The implementation of Runge's function in Machine Learning
FYS-STK4155 - Project 1

Malene Fjeldsrud Karlsen, Birk Tinius Skogsrud and Elias Bjørgum Micaelsen-Fuglesang
University of Oslo

(Dated: October 7, 2025)

Machine Learning is an important and exciting field within computational science focusing on learning pattern recognition and the prediction of data. This report focuses on the fundamentals of Machine Learning through mathematical and statistical aspects of producing a dataset and creating algorithms that finds the underlying relationship of the data. Using Runge's function, the goal is to understand how well various regression methods fit this one-dimensional polynomial, with the implementation of Ordinary Least Squares, Ridge Regression, and Lasso Regression. Furthermore, we will introduce methods of Gradient Descent and momentum to enhance the learning, and resampling methods to reproduce data sets efficiently and profitably. The goal is to gain a deeper understanding of the fundamentals of Machine Learning and compare and analyze how these algorithms and methods work. To solve this, the report will contain a numerical solution of the varying methods and a results and discussion section to discuss and highlight what we can learn from these algorithms. Some key takeaways from the report are that Runge's function is a polynomial with characteristics that are not easy to fit using any of the mentioned methods. All results give high variance and low bias for higher degrees of polynomials. Although with higher sample sizes and introducing a regularization parameter, we find the Ridge Regression to be the better-suited method in this project.

I. INTRODUCTION

Using Runge's function, we will, in this study, explore how we can use Machine Learning to best fit the data of this one-dimensional polynomial. To generate a machine learnt model for the given data, this report will focus on the fundamentals of Machine Learning. Using a supervised learning method, we will create algorithms for various regression methods: the Ordinary Least Squares, Ridge Regression, and Lasso regression, allowing us to compare the outcomes and decide on which one of the regressions suits our data the best [1]. To achieve this, we must identify possible sources of errors in our model, such as overfitting the model with too complex data or underfitting our data with too little data. Overfitting will not allow our model to capture the underlying patterns of our dataset, while underfitting our data does not provide enough information for our model to learn the real behavior of the provided data points.[1] To find these sources of error and navigate through the challenges of fitting our polynomial, we have performed a statistical analysis by calculating the Mean Squared Error and the R2-score to further evaluate the fit and study how well our model performs. The Mean Squared Error and R2-score, also known as a cost function and metrics respectively, allow us to evaluate the errors in our model by comparing our predicted data with the real data, IE., the 'truth'. Finding the parameters that suit our data is achieved through the regression algorithms, while gradient descent and optimization methods such as AdaGrad, Momentum, RMSProp, and ADAM are used to find the parameters that minimize the difference between our predicted values and our real data. [2] Finding the minimum of our cost functions provides us with valuable information about the ideal parameters to use when fitting the polynomial to create better predictions.

This report aims to provide a better understanding of how to increase efficiency in our created model and how to navigate through the challenges of overfitting, underfitting, high variance, and high bias through the bias-variance tradeoff. [1] The paper will also include methods on how to resample our datasets using the Bootstrap method and cross-validation to ensure our model understands and adapts to the variability of the dataset while producing new datasets in a profitable manner. Using resampling methods is a process that ensures the model can be used in the future for predicting the relationships and correlations found for the data in new datasets. In the Methods Section (II) of this paper, we will provide a mathematical explanation of the methods used, as well as the numerical solutions. All results will be discussed and reviewed to compare the methods used and their efficiency, as well as their ability to fit our given data in Section III and IV of the report. The challenges faced in the numerical solutions will be presented if they affect our model output and critically reviewed for further use.

Furthermore, we will conclude on our most important findings in section V and what to focus on in future projects of a similar nature. Using the results and conclusions of this paper, the idea is to gain a better understanding of the fundamentals of Machine Learning and to implement the same algorithms for mathematical functions of higher orders and higher dimensions. Gaining a good understanding of our current dataset will allow us to work with larger datasets in the future that contain more information about physical properties and their underlying correlations. [1] The minimization and optimization algorithms used throughout the report are a part of the main ingredients for all Machine Learning algorithms, both for lower and higher complexity cases.

Understanding the basics will therefore help in ensuring reliable and reproducible results in the future.

II. METHODS

A. OLS Regression

OLS is a form of linear regression which is defined as the minimization of the L_2-norm of the response y_i and the predictor $\tilde{y} = X\theta$:

$\min_\theta \|X\theta - y_i\|_2^2 = \min_\theta \sqrt{\sum_{i=1}^n (x_i^T\theta - y_i)^2}$

[3]

For our case: y_i are the resulting values of Runge´s function for our x_i-values, X is the matrix containing our x_i-values where each column contain the x-values for exponents 1,2,3,...,16, and θ is the matrix containing the coefficients multiplied with the x-values to make our model fit with the response.

By using the OLS method, we find the values for θ that minimize the L_2-norm [3]. I.e. we want to find the values for theta which minimizes the sum of the square difference between y_i and $X\theta$ for all our data points.

The expression above can be rewritten as a minimization problem of a matrix equation, which is called the cost-function $C(\theta)$:

$C(\theta) = 1/n((y - X\theta)^T(y - X\theta))$

By differentiating the cost-function and setting it equal to zero, we find the optimal θ, $\hat{\theta}$:

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

$\hat{\theta} = (X^TX)^{-1}X^Ty$

Multiplying $\hat{\theta}$ with X, we get the output of the OLS model: $\tilde{y}$. To compare y with the output of the OLS model, $\tilde{y}$, we take the mean-squared-error of these two.

[1]

B. Ridge Regression

For Ridge regression we add a regulator, λ, to the OLS equation:

$\hat{\theta}_{Ridge} = \arg\min_\theta(\|X\theta - y_i\|_2^2 + \lambda\|\theta\|_2^2)$

With Ridge regression, we constrain the magnitude of θ [3]. I.e we force the values of θ. If we choose a small value for λ, $\hat{\theta}_{ridge}$ will be similar to $\hat{\theta}_{OLS}$, but if we choose a large value for λ, $\hat{\theta}_{ridge}$ will shrink towards 0. [4]

Again we rewrite the expression above as a matrix equation:

$C(X, \theta) = ((y - X\theta)^T(y - X\theta)) + \lambda\theta^T\theta$

Where we get the optimal parameters $\hat{\theta}_{ridge}$:

$\hat{\theta}_{ridge} = (X^TX + \lambda I)^{-1}X^Ty$

Which means that when X is orthogonal, we have the following relationship:

$\hat{\theta}_{Ridge} = \frac{1}{1+\lambda}\hat{\theta}_{OLS}$

[3]

So $\hat{\theta}_{Ridge}$ will be smaller than $\hat{\theta}_{OLS}$ since $\lambda > 0$.

C. LASSO regression

With LASSO regression, we use the L_1-norm for the regularization term, instead of the L_2-norm as we used in Ridge regression:

$\hat{\theta}_{LASSO} = \arg\min_\theta(\|X\theta - y_i\|_2^2 + \lambda\|\theta\|_1)$

[3]

To extend on the difference between Ridge and Lasso regression: The regulator term for Ridge regression strongly affects the large θ-values, but the values closer to zero are not as affected, meanwhile, Lasso does not shrink the large θ-values as strongly, but the values close to zero are more affected. [4]

However, we can´t solve the equation above analytically. This is because of the L_1-regularizer, which can not be differentiated for $\theta_j = 0$ [3]. Therefore we have to use a numerical method, as for instance, the gradient descent methods explained in the next section.

D. Gradient descent

A basic method for gradient descent is:

$\theta_{t+1} = \theta_t - v_t$, where we update our θ-values from an initial θ, θ_0, with:

$v_t = \eta_t\nabla_\theta C(X, g(\theta))$, where η_t is the learning rate, which controls the size of the step along the gradient, and $C(X, g(\theta)) = \sum_{i=1}^n e_i(x_i, \theta)$ is the cost function, where e_i is the mean squared error for linear regression. [3] If we choose η_t to be small, the method will reach a local minimum of the cost function, but this can be time consuming when running the code. However, if the learning rate, η_t, is too big, the algorithm can become unstable. [3]

There are some limitations to this simple method for gradient descent. For example this method finds the local minima and that the local minima might be mistakenly found because of rugged datasets. Also, this method is sensitive to initial values - The local minima we find depend on the chosen initial values. This is also the case for other variants of gradient descent. As mentioned above, the learning rate can also lead to instability or be time-consuming when running the code. [3]

E. Stochastic gradient descent

Stochastic gradient descent is a variant of the gradient descent method where we use gradient descent for multiple batches. These batches are usually much smaller than the dataset, which means this can be more efficient than using the basic method for gradient descent [3]. Also, we are more likely to find the absolute minima using stochastic gradient descent [3]. The equation for stochastic gradient descent is similar to gradient descent:

$\theta_{t+1} = \theta_t - v_t$, but for stochastic gradient descent:
$v_t = \eta_t \nabla_\theta C^{MB}(X, g(\theta))$, where the cost function is:
$\nabla_\theta C(X, g(\theta)) = \sum_{i \in B_k}^{M} \nabla_\theta e_i(x_i, \theta)$
[3]
Where we see that i only run through the M batches B_k, compared to gradient descent, where we sum up for all the n data points.

F. Momentum

Stochastic gradient descent usually includes a momentum term, which weights the previous step:
$\theta_{t+1} = \theta_t - v_t$ with:
$v_t = \gamma v_{t-1} + \eta_t \nabla_\theta C(X, g(\theta)$
[3]
γ is the momentum parameter and is valued between 0 and 1 [3]. As seen in the equation above, the momentum parameter decides how much we weight the previous time step.

Momentum is useful because it increases speed through flat regions and smooths out oscillations [3].

G. AdaGrad

The main concept of AdaGrad is to update the learning rate η_t, which is more effective for sparse data and features with large differences in magnitude [5]. This is how the learning rate is updated:
$\eta_t = \frac{\eta}{\sqrt{G_t + \epsilon}}$, where η is a small constant (referred to as the global learning rate), G_t is the sum of the squared gradients from t=1 to t=t, and ϵ is a small constant to avoid division by zero [5].

We see η_t decreases as G_t increases for each step, which helps stabilizing the training.

We update the parameters with:
$\theta_{t+1} = \theta_t - \eta_t g_t$, where $g_t = \nabla_\theta C(X, g(x))$
AdaGrad is useful for models that benefit from a more stable optimization process, but for models where we want a learning rate which doesn´t descend as quickly, RMSprop and ADAM are preferred. [5]

H. RMSprop

For RMSprop, we update the parameters with:
$\theta_{t+1} = \theta_t - \eta_t \frac{g_t}{\sqrt{s_t + \epsilon}}$
where $s_t = \beta s_{t-1} + (1-\beta) g_t^2$ where $s_t = E[g_t^2]$ and β controls the averaging time.
[3]
In contrast to the previous methods, here we take the running average of the square of the gradient. This prevents the learning rate from decreasing too fast, which can be a problem when using AdaGrad. [6]

I. ADAM

For ADAM, we keep the running average of both the first momentum (gradients), m_t, and second momentum (squared gradients), s_t [3]. s_t is the same as for RMSprop, but we use a different $\beta = \beta_2$. The first momentum is similar to the second momentum, but for the first momentum, we don´t square the gradient, and we use $\beta = \beta_1$: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$, where $m_t = E[g_t]$ [3]
ADAM also corrects for bias since we are using a running average for the first two moments: $\hat{m}_t = \frac{m_t}{1-\beta_1}$ and $\hat{s}_t = \frac{s_t}{1-\beta_2}$
[3]
Using ADAM, the parameters are updated with:
$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{s}_t + \epsilon}}$ [3]

J. Bias-variance trade-off and resampling techniques

Bias and variance is important for measuring the accuracy of the algorithms used. Bias measures the difference between the expectation value of the estimator and the true value: $Bias^2 = \sum_i (f(x_i) - E_\mathcal{L}[\hat{g}_\mathcal{L}(x_i))^2]$ Where $\hat{g}_\mathcal{L}$ is the predictor and $mathcalL$ is the dataset [3]

High bias gives large errors for both training and testing data, and can lead to underfitting [7].

The variance measures how much the estimator fluctuates: $Var = \sum_i E[(\hat{g}_\mathcal{L}(x_i) - E_\mathcal{L}[\hat{g}_\mathcal{L}(x_i))^2]]$ [3]

The variance can be viewed as the spread of the data, and a model with high variance will overfit [7].

We can show that the expectation value of the squared error of the difference between the response, $y = f(x_i) + \epsilon$, and the predictor, $\hat{g}_\mathcal{L}(x_i) = \tilde{y}$ is equal to the sum of the bias, the variance and the variance of the noise:

We have the expression for the MSE (using $f(x_i) = f$:
$\frac{1}{n}\sum(y - \hat{g}_\mathcal{L}(x_i))^2 = E[(f + \epsilon - \tilde{y})^2]$
Which we can rewrite as:
$E[f^2 + f\epsilon - f\tilde{y} + f\epsilon + \epsilon^2 - \epsilon\tilde{y} - f\tilde{y} - \epsilon\tilde{y} + \tilde{y}^2]$
$= E[f^2 + 2f\epsilon - 2\tilde{y}\epsilon + \epsilon^2 - 2f\tilde{y} + \tilde{y}^2$
$= E[f^2 + 2\epsilon(f - \tilde{y}) + \epsilon^2]$
$= E[(f - \tilde{y})^2 + 2\epsilon(f - \tilde{y}) + \epsilon^2]$
$= E[(f - \tilde{y})^2] + E[2\epsilon(f - \tilde{y})] + E[\epsilon^2]$
And since ϵ and $(f - \tilde{y})$ we can write:
$E[2\epsilon(f - \tilde{y})] = E[2\epsilon]E[(f - \tilde{y})]$ where $E[2\epsilon] = 0$
And since $E[\epsilon^2] = \sigma^2$ we now have:
$E[(f - \tilde{y})^2] + \sigma^2$
$= E[f^2 - 2f\tilde{y} + \tilde{y}^2] + \sigma^2$
$= E[f^2] - E[2f\tilde{y}] + E[\tilde{y}^2] + \sigma^2$
We have that: $Var[\tilde{y}] = E[\tilde{y}^2] - E[\tilde{y}]^2$
$Var[\tilde{y}] + E[\tilde{y}]^2 = E[\tilde{y}^2]$ which we can plug into the expression:
$E[f^2] - E[2f\tilde{y}] + var[\tilde{y}] + E[\tilde{y}]^2 + \sigma^2$
$= f^2 - 2fE[\tilde{y}] + E[\tilde{y}]^2 + var[\tilde{y}] + \sigma^2$ since $E[2f] = 2f$
$= (f - E[\tilde{y}])^2 + var[\tilde{y}] + \sigma^2$

$= E[(f - E[\tilde{y}])^2] + var[\tilde{y}] + \sigma^2$ since $(f - E[\tilde{y}])$ is constant. $E[(f - E[\tilde{y}])^2] = bias[\tilde{y}]$ so now we have our final expression:

$bias[\tilde{y}] + var[\tilde{y}] + \sigma^2$

The bootstrap method is used for assessing accuracy. With bootstrap, the data is resampled by drawing random values from the training dataset (with replacement) and placed into a new dataset. If the training set contains n values, the new dataset should also contain n values. There are made multiple of these datasets. The model can than be fit to these new datasets, which can further be examined. [8]

Cross-validation is a widely used method for prediction error. Here, we will look at K-fold cross-validation. For K-fold cross validation, we split the data into K sets of the same size, where we fit the model to all but one of the sets. For each data-set, the prediction error is also calculated. This is done K times, where the subsets are reshuffled, and then we take the average of these prediction error estimates. The prediction error estimates might be larger than the actual error if the size of the datasets are too small. [8]

K. Implementation

Implementation of the regression methods

To implement the Ordinary Least Squares regression analysis, we have created an algorithm that defines Runge's function including noise $\varepsilon \sim \mathcal{N}(0, 1)$ and creates a design matrix X consisting of 16 polynomial degrees without the intercept 1 and with three different values for the size of the dataset. Before performing the OLS analysis, the design matrix and function are split into two subsets, consisting of randomly chosen training data and test data, with sizes of 4/5 and 1/5, respectively. Furthermore, we have centered the training data for our design matrix X and our response variable y, as well as the test data from X. From there, we implement the regression analysis as our own algorithm 2. The values for the mean squared error and the R2 score are then predicted using the existing features in Scikit learn [9], then plotted in Figure 1 and 2. The Scikit learn function takes in the true y values as the first argument and the predicted y values as the second argument. The predicted y values ($\tilde{y}$) are calculated as: $(X_{centered} \cdot \Theta_{params}) + mean(y)$ for all implementations in this report.

To visualize the theta parameters found in the OLS regression, we have stored the resulting thetas in an initialized array with the form $p \times len(p)$, where p is the chosen polynomial degree of 15. The results are then plotted 3.

Algorithm 1: Creating Design Matrix X

```
Input: Data vector (x), polynomial degree (p),
       boolean $includeIntercept$
Output: Design matrix $X$
$n \leftarrow$ length of $x$ ;
if $includeIntercept = True$ then
    Initialize $X$ as an $n \times (p + 1)$ matrix of zeros;
    for $i \leftarrow 0$ to $p$ do
        $X[:, j] \leftarrow x^j$;
    end
end
else
    Initialize $X$ as an $n \times p$ matrix of zeros;
    for $j \leftarrow 1$ to $p$ do
        $X[:, j - 1] \leftarrow x^j$;
    end
end
```

Algorithm 2: Ordinary Least Squares (OLS) regression

```
Input: Design matrix $X \in \mathbb{R}^{n \times p}$, target vector
       $y \in \mathbb{R}^n$
Output: Coefficient vector $\theta \in \mathbb{R}^p$
Compute the normal equation:
$\theta \leftarrow (X^T X)^{-1} X^T y$
Return $\theta$
```

The implementation of the Ridge Regression and the Gradient Descent uses the same general implementation of code as in the OLS. The calculation of the Ridge Regression is based on the same method as in 2, but also includes a regulator λ as discussed and shown in the method section.

For the Ridge Regression, we run the iteration to calculate the parameters theta for both the polynomial degree and for three different values of λ. Storing this two-dimensional data in a matrix allows us to visualize the MSE and R2 scores by plotting two-dimensional heatmaps. We have also chosen to scale the data for the Ridge Regression, instead of centering it as for the OLS. This is further discussed in section IV.

For the LASSO regression due to the non-differentiability of the L_1 regularization term at zero. It was implemented only using gradient-based optimization methods.

Implementation of Gradient Descent

Moving on from the regression methods, the Gradient Descent is implemented by running the Ridge and OLS regression method with three different values for the learning rate η, and multiplying them by the gradient descent values (GD). Then we update the parameters θ and plot the results.

To later update the learning rate, we run the Gradient Descent code again, but this time we introduce four new methods for updating the parameters effectively. We introduce the optimizers: Momentum 3, Adam , AdaGrad 4, and RMSProp 5. In these methods, λ is held fixed at 0.1, and η has a starting value of 0.01 for all optimization methods. The value for ϵ is chosen as a small value to prevent possible matmul errors in Python from trying to calculate mathematical expressions with zero.

We also, for OLS, Ridge, and LASSO plotted convergence curves of the training MSE and the final coefficient values using the same initial learning rate $\eta = 0.01$ using the different optimization schemes.

Algorithm 3: Momentum Optimization

```
Input: Momentum = 0.3, $v$ = list(zeros),
       $\eta = 0.01$
$v = \text{momentum} \cdot v + \eta \cdot$ Gradient Descent (GD);
$\theta_{params} = \text{GD} - v$;
```

Algorithm 4: AdaGrad Optimization

```
Input: $\epsilon = 1e-8$, $\eta = 0.01$, Adagrad = gradient $\cdot$
       gradient
$GD_{new} = \text{GD} \cdot \eta / (\epsilon + \sqrt{\text{AdaGrad}})$;
$\theta_{params} -= GD_{new}$;
```

Algorithm 5: RMSProp Optimization

```
Input: $\epsilon = 1e-8$, $\beta = 0.9$, $\eta = 0.01$, RMSProp
       $= \beta \cdot \text{RMSProp} + (1-\beta) \cdot \text{GD} \cdot \text{GD}$
$GD_{new} = \text{GD} \cdot \eta / (\epsilon + \sqrt{\text{RMSProp}})$;
$\theta_{params} -= GD_{new}$;
```

Lastly, the ADAM optimization is a combination of the Momentum and the RMSProp optimizers, creating two moving averages (moments) that provide a weighted update of 3 and 5 [10].

Stochastic Gradient Descent implementation

The Stochastic Gradient-based implementation for OLS, Ridge, and LASSO regression follows the same general template as gradient descent but employs *mini-batches*. Feature columns are centered for OLS, while they are standardized using z-score normalization for Ridge and LASSO, with the intercept term added afterward. At each epoch, the training indices are shuffled and partitioned into mini-batches b of size $|b| = B$. For each batch (X_b, y_b), the subgradient is computed and the parameters are updated using a fixed learning rate $\eta = 0.01$. One parameter snapshot is recorded per epoch for use in convergence plots.

For the LASSO regression, it is important to note that when validating against `scikit-learn`'s implementation (`sklearn.linear_model.Lasso`), the regularization parameters differ by a scaling factor. Our objective is defined as

$$\min_{\theta} \ \frac{1}{n} \|X\theta - y\|_2^2 \ + \ \lambda \|\theta\|,$$

where n is the number of samples and the intercept term is not penalized. In contrast, `scikit-learn` defines the LASSO objective as

$$\frac{1}{2n} \|X\theta - y\|_2^2 + \alpha \|\theta\|_1.$$

To ensure equivalence between the two formulations, we set

$$\alpha = \frac{\lambda}{2},$$

so that the regularization strengths are consistent across implementations[9].

K-fold cross validation implementation

We estimate MSE as a function of model complexity using K-fold cross-validation with $K = 25$. For each dataset size $N \in \{200, 300, 400\}$. The index set $\{1, \ldots, N\}$ is partitioned once into K disjoint folds with shuffling and a fixed seed and the same fold partition is reused for all polynomial degrees $p \in \{1, \ldots, 17\}$. For a given degree p and fold k, we train on the union of the $K-1$ non-k folds and test on fold k. The design matrices are constructed from polynomial features of degree p, then centered to obtain $X_{\text{tr}}^{(c)}$ and $X_{\text{te}}^{(c)}$. The intercept column is appended after centering. Parameters are estimated by closed-form OLS on the training fold and for each fold we compute the training and test mean squared errors (MSE). These are averaged over all K folds to obtain, for each degree p, the fold-averaged training and test MSE.

Bootstrap implementation

The bootstrap implementation follows a resampling-based procedure to estimate the variability of MSE as a function of model complexity. For each polynomial degree p, we perform $B = 120$ bootstrap resamples by sampling the training indices with replacement. For every resample, we construct centered design matrices $X_{\text{tr}}^{(c)}$ and $X_{\text{te}}^{(c)}$ using the training subset only, fit the Ordinary Least Squares model in closed form and compute both the training and test MSE. The MSEs are then averaged over all B resamples for each degree.

L. Use of AI tools

In this project, we have used ChatGPT for help with the following:

- Used to help with the implementation of code. The numerical solutions are all created by the authors of this project, but ChatGPT has been used to increase efficiency in handling errors in the Python code and making plots.

- Used to ask questions about uncertainties in the project description.
- The website Grammarly has been used as a tool for avoiding spelling mistakes. Only the free version has been used, meaning it helps with spelling mistakes, but it does not rewrite or improve the contents of the report.

III. RESULTS

A. OLS Regression

For the OLS regression, we have plotted the Mean Squared Error and the R2 scores for the training and test data. Figure 1 shows the MSE values for the OLS Regression for 15 polynomial degrees, with varying sizes of datasets labeled as n in the legends. Figure 2 shows the R2 scores.

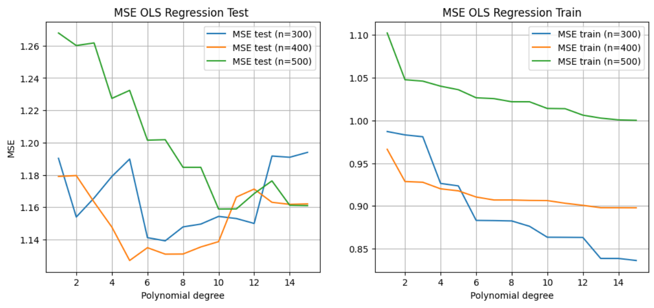

Figure 1: Mean Squared Error (MSE) for the Ordinary Least Squares Regression. The legends show the varying number of data points n. The left figure shows the test data, and the right figure displays the training data.

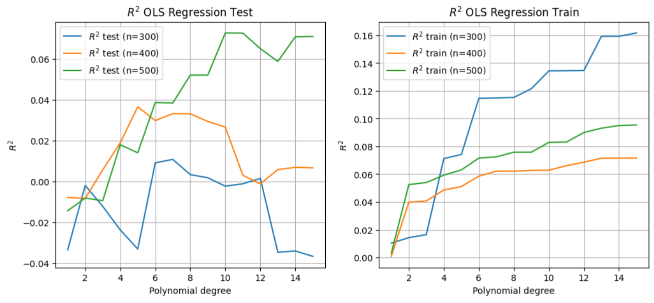

Figure 2: The R2 score for the Ordinary Least Squares Regression. The legends show the varying number of data points n. The left figure shows the test data, and the right figure displays the training data.

Furthermore, we have plotted the coefficients θ for the OLS regression as a scatter plot to visualize the divergence of the data points for the polynomial degrees. The main goal for this plot is to understand the data and to better explain the results found in Figures 1 and 2, which will be discussed later in the paper.

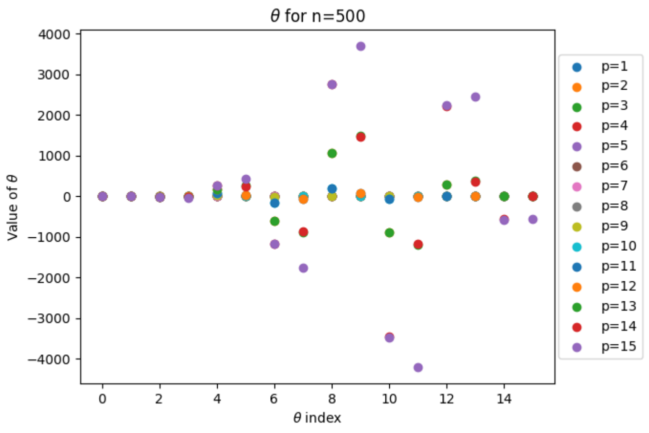

Figure 3: θ´s for each polynomial degree up to 15 for the Ordinary Least Squares Regression. P in the legend contains θ's for each degree.

B. Ridge Regression

For the Ridge Regression, we are storing the results in a matrix and plotting them as a two-dimensional heatmap, with varying λ on the abscissa and the polynomial degree as the ordinate. The colorbar of the heatmap shows the MSE values in Figure 4, and the values of the R2 scores in Figure 5. The coefficients θ are also printed in the code to ensure that the code works well and to study how the data points diverge. In the discussion, we will consider how the varying λ affects our results.

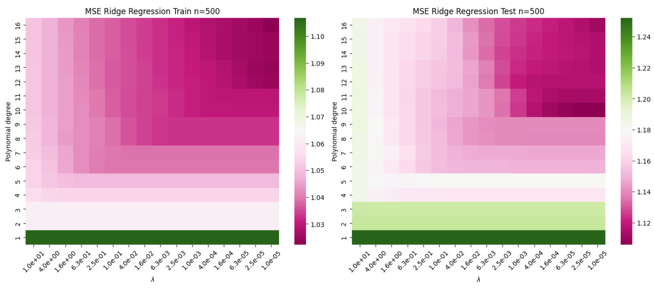

Figure 4: Two heatmaps showing the training data to the right and the test data to the left for the Ridge Regression. The colorbar contains information about the MSE-values for varying λ's on the abscissa and the polynomial degree on the ordinate.

C. Gradient Descent

For the gradient descent, the first two graphs visualize how the learning parameter η affects the results, by

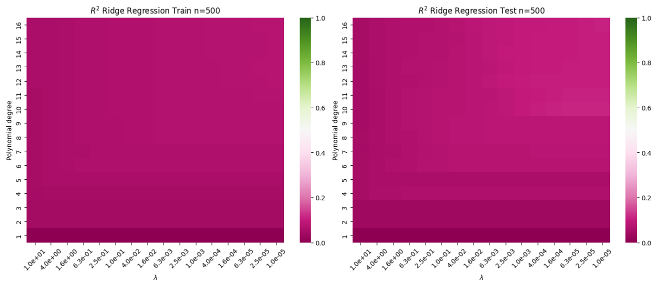

Figure 5: Two heatmaps showing the training data to the right and the test data to the left for the Ridge Regression. The colorbar contains information about the R2 scores for varying λ's on the abscissa and the polynomial degree on the ordinate.

plotting the MSE values and the R2 scores in Figure 6, for the gradient descent of the OLS. Figure 7 contains the same information, but for the gradient descent of the Ridge regression. The data size is kept at N = 500 for figures: 5, 6, 7, 8, 9. Though it is important to include the training data, we decided to only plot the test data for this part of the report to understand how well the model performs when introduced to new and unseen data.

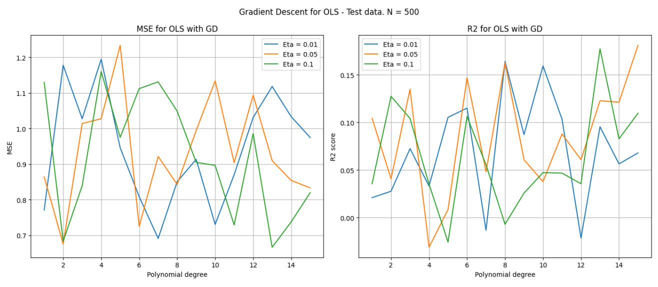

Figure 6: The MSE values for the Ordinary Least Squares Regression for varying learning rates η on the left figure. The right figure shows the R2 scores. All the data in Figure 6 is based on the test data.

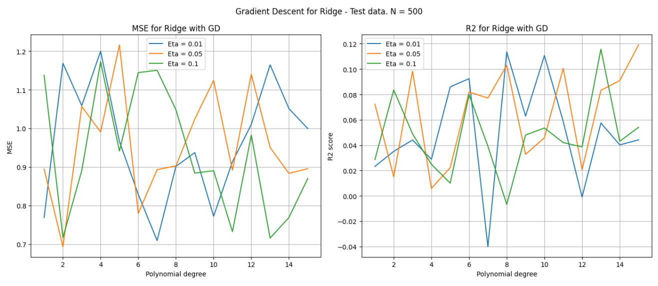

Figure 7: The MSE values for the Ridge Regression for varying learning rates η on the left figure. The right figure shows the R2 scores. All the data in Figure 7 is based on the test data.

In Figure 8, the results are plotted for the OLS gradient descent, using the adaptive learning features introduced in the methods section of the paper and labeled in the legend of the plot. The result are repeated for the Ridge gradient descent in Figure 9.

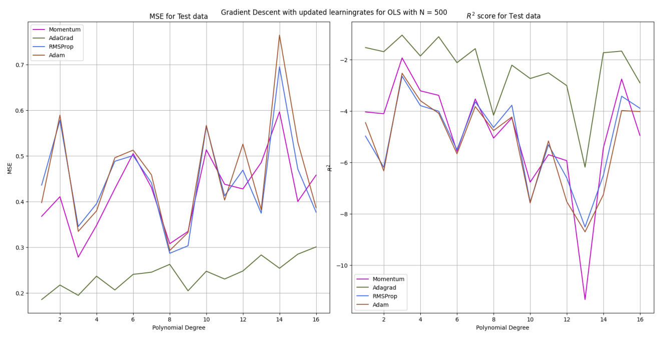

Figure 8: The MSE values for the OLS Regression using optimization methods shown in the legends. The right figure shows the R2 scores. All the data in Figure 8 is based on the test data. The initial learning rate value is set as $\eta = 0.01$

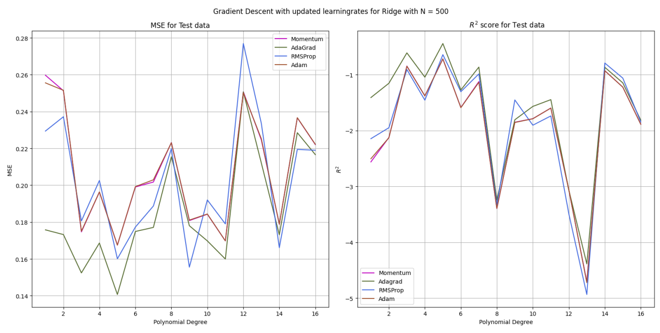

Figure 9: The MSE values for the Ridge Regression using optimization methods shown in the legends. The right figure shows the R2 scores. All the data in Figure 9 is based on the test data. The initial learning rate value is set as $\eta = 0.01$

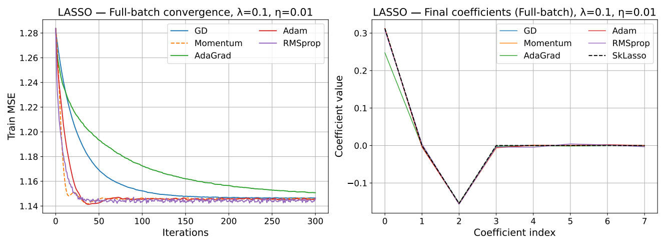

Figure 10: Lasso regression with full-batch optimization ($N = 1000$, $\lambda = 0.1$, polynomial degree $p = 7$, 300 iterations, initial step size $\eta = 0.01$). Left: convergence of Train MSE for the following full-batch schemes plain gradient descent GD, Momentum, AdaGrad, RMSprop, and Adam. Right: final coefficients after optimization, compared with those obtained from scikit-learn's Lasso.

As shown in 10, Momentum, Adam and RMSprop converge the fastest with only minor oscillations, plain GD reduce the training error more slowly, and AdaGrad

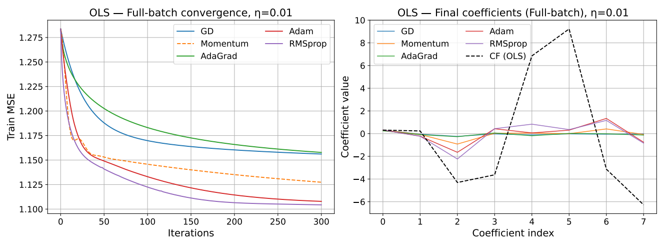

Figure 11: Ordinary Least Squares with full-batch optimization ($N = 1000$, $p = 7$, 300 iterations, initial step size $\eta = 0.01$). Left: Train MSE convergence for full-batch methods plain GD, Momentum, AdaGrad, RMSprop, and Adam. Right: final coefficients obtained after training, compared with the OLS closed-form solution.

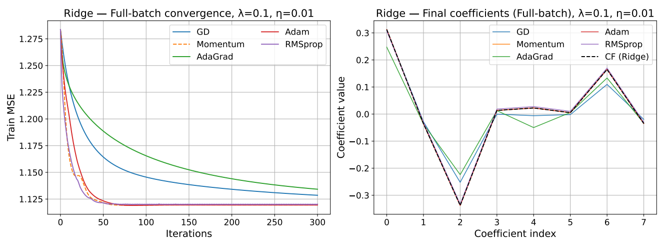

Figure 12: Ridge regression with full-batch optimization ($\lambda = 0.1$, $\eta = 0.01$, 300 iterations). Left: Train MSE convergence for full-batch schemes plain GD, Momentum, AdaGrad, RMSprop, and Adam. Right: final coefficients after training, compared with the closed-form Ridge solution.

is the slowest. The resulting Lasso coefficients closely match scikit-learn's implementation (`SkLasso`) when using $\alpha = \lambda/2$, with Adagrad looking to be the least accurate.

In Fig. 11, RMSprop performs best overall, showing a smooth and stable convergence as it flattens near the minimum, while Adam also achieves a comparably low training error. Momentum, plain GD, and AdaGrad converge more slowly and end up with higher error levels. Notably, Momentum initially converges at a rate similar to Adam and RMSprop, but its progress slows in the later stages. The estimated coefficients, however, differ noticeably from the closed-form OLS solution.

Figure 12 presents the results for Full batch Ridge regression. Adam, RMSprop and Momentum converge quickly and stabilizes at training MSE around 1.125, whereas AdaGrad and GD fail to fully converge in the allotted iterations. The learned coefficients align well with the closed-form Ridge solution overall, with Adam, RMSprop and Momentum showing the closest agreement.

D. Stochastic gradient descent

In Figure 13, plain SGD, Adam, Momentum and RMSprop reach low training error quickly but exhibit noticeable epoch-to-epoch noise. This is especially the case for Momentum which exhibits large oscilliations after the in-

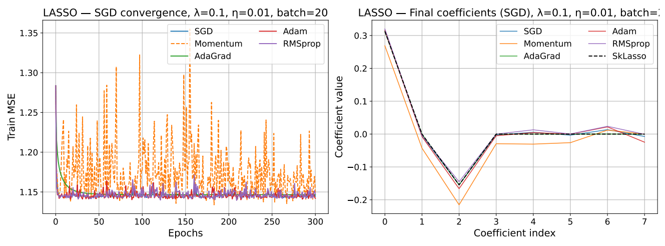

Figure 13: Lasso regression with stochastic optimization ($N = 1000$, $p = 7$, $\lambda = 0.1$, mini-batch size 20, 300 epochs). Left: Train MSE convergence for stochastic gradient descent variants plain SGD, SGD with Momentum, SGD–AdaGrad, SGD–RMSprop, and SGD–Adam. Right: final coefficients obtained after training, compared to scikit-learn's Lasso.

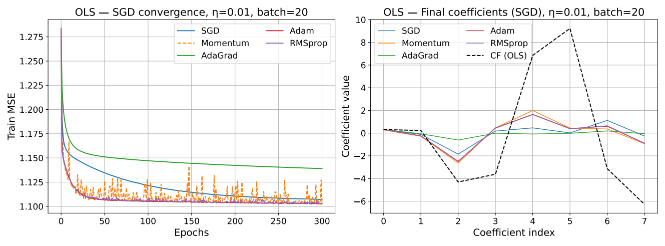

Figure 14: Ordinary Least Squares with stochastic optimization ($p = 7$, 300 epochs, learning rate $\eta = 0.01$, mini-batch size 20). Left: convergence of stochastic gradient descent variants plain SGD, SGD with Momentum, SGD–AdaGrad, SGD–RMSprop, and SGD–Adam. Right: final coefficients obtained after training, compared with the OLS closed-form solution.

tital epochs, while AdaGrad progresses more slowly and appears nearly noise-free. After 300 epochs, the Lasso coefficients agree well with `SkLasso` when using $\alpha = \lambda/2$, with Momentum yielding the least accurate estimates.

Figure 14 shows the results for SGD applied to OLS. RMSprop and Adam achieve the fastest reduction in training error and reach the lowest values. Momentum follows Adam and RMSprop closely, but exhibits pronounced oscillations and lacks fully stable convergence. Plain SGD converges more slowly, while AdaGrad is significantly slower and fails to reach convergence within the allotted epoch budget. The final coefficients obtained from all gradient-based methods deviate noticeably from the closed-form OLS solution, indicating generally poor agreement with the analytical reference.

Following Figure 15, which presents the results for SGD applied to Ridge regression, all methods except AdaGrad and Momentum converge rapidly. Adam, SGD and RMSprop exhibit small but persistent fluctuations, much less pronounced than those observed for Momentum. Plain SGD oscillates less, while AdaGrad shows no visible oscillations but converges more slowly. The learned coefficients are generally very close to the closed-form Ridge solution across methods, with Momentum showing the weakest agreement.

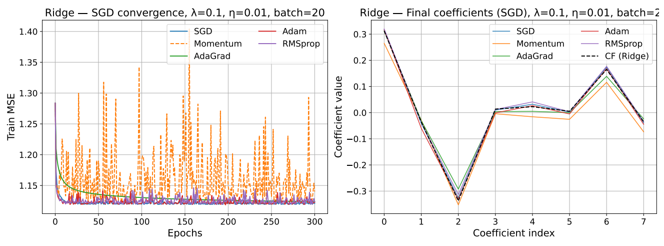

Figure 15: Ridge regression with stochastic optimization ($\lambda = 0.1$, $\eta = 0.01$, 300 epochs, mini-batch size 20). Left: Train MSE convergence for stochastic gradient descent variants plain SGD, SGD with Momentum, SGD–AdaGrad, SGD–RMSprop, and SGD–Adam. Right: final coefficients obtained after training, compared with the closed-form Ridge solution.

E. Bootstrap

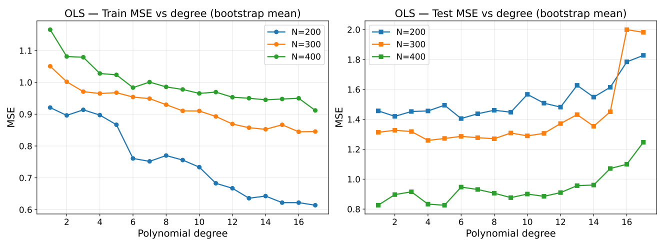

Figure 16: The plots illustrates the results of the bootstrap mean analysis. The left panel displays the training MSE for Ordinary Least Squares (OLS) with sample sizes $N = 200$, 300, and 400, while the right panel shows the corresponding test MSE. Both metrics are evaluated across polynomial degrees p ranging from 1 to 17.

Figure 16 presents the mean training and test mean squared error (MSE) as functions of polynomial degree for dataset sizes $N = 200, 300$, and 400, estimated using the bootstrap resampling method. In the left panel, the mean training MSE decreases nearly monotonically with increasing polynomial degree for all dataset sizes, as expected when model flexibility increases. The decrease is steepest for $N = 200$, which also achieves the lowest training MSE across most degrees. The curves for $N = 300$ and $N = 400$ are closer together, with $N = 400$ exhibiting slightly higher mean training MSE across the entire range of degrees. In the right panel, the mean test MSE initially decreases for all datasets, reaching a minimum around degrees $p =$ 4-6, before gradually increasing as the model complexity grows. Beyond $p \approx 14$, the mean test MSE rises more sharply. The lowest overall test errors occur for $N = 400$, followed by $N = 300$ and $N = 200$. At higher degrees ($p > 15$), the mean test MSE for $N = 300$ overtakes that of $N = 200$, while $N = 400$ maintains the lowest and most stable test error across all degrees.

F. Cross validation

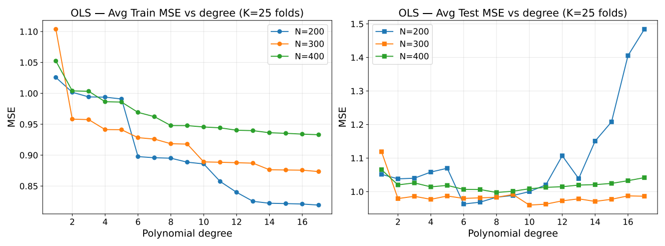

Figure 17: The plots illustrate the results of the K-fold cross-validation analysis. The left panel displays the training MSE for Ordinary Least Squares (OLS) with sample sizes $N = 200$, 300, and 400, while the right panel shows the corresponding test MSE. Both metrics are evaluated across polynomial degrees p ranging from 1 to 17.

Figure 17 shows the results of the K-fold cross-validation analysis. For the training MSE, all curves show a consistent downward trend as the polynomial degree increases. The training error decreases rapidly in the beginning for the lower polynomial degrees for $N = 200$ and $N = 300$. $N = 200$ achieves the lowest overall MSE values across nearly all degrees. $N = 400$ exhibit in general higher training MSE with a smoother decrease. The test MSE values remain relatively similar across all dataset sizes up to approximately $p = 12$, with $N = 400$ showing a smoother and more gradual change across polynomial degrees. Beyond $p \approx 11$, the curve for $N = 200$ begins to rise sharply. The final test MSE for $N = 200$ is notably higher than those for $N = 300$ and $N = 400$, which remain more similar in magnitude throughout the degree range.

IV. DISCUSSION

In Figure 1 of the OLS Regression analysis, we find that the test data for the MSE sinks effectively until it reaches the polynomial degree of 10 for all three values of n, before it starts increasing. Simultaneously, the training data decreases in value, indicating that the training data is adapting to the noise of the dataset rather than the real correlations of the dataset. Knowing that MSE consists of three terms: variance, bias, and variance of the noise, this relationship indicates too high model complexity, resulting in high variance and low bias.

The same effect of overfitting is seen in Figure 2 showing the R2 scores. The decrease in R2 scores for the test data indicates the model is performing progressively worse as the polynomial degree exceeds 10. It is worth mentioning that the MSE scores obtained when using n = 500 provide a better R2 score, indicating that increasing the dataset has a positive effect on the OLS Regression.

Comparing the results with the heatmap from the Ridge Regression 4 and 5, which includes various values for λ, indicates that the Ridge Regression is a better fit for our polynomial for higher values of n. While lower values of n give the same results as for the OLS, a higher value of n, however, shows that the values of MSE get lower for both the training and test data as the polynomial degree increases and the value of λ increases. This is a good indication that, as the regularization parameter puts a higher penalty on the parameters of a larger dataset, the model is performing better. The R2 scores for the Ridge Regression still remain low, and there are clear signs of overfitting for this regression as well, but it can be argued that the Ridge Regression helps reduce the high-degree polynomial fit of the Runge‘s function. One difference in the calculations of the OLS and the Ridge Regression, is that we have decided to not only center the data for the Ridge Regression, but scale it by dividing the centered data by the standard deviation. This is an important change as the Ridge Regression's regularization parameter performs better on scaled data, where all the variables are kept on a comparable scale.

Figure 3 illustrates the overfitting, as the estimated θ coefficients from the OLS Regression are stable for the lower polynomial degrees, but "explode" as the degree increases, which indicates instability as the values diverge from zero. The penalty induced by λ in the Ridge Regression will make these values decrease towards zero again, which is a good solution for preventing the same instability.

Now that we have looked at the OLS and Ridge, we will consider how the implementation of Gradient Descent impacts the MSE and R2 score values. Figures 6 and 7 illustrate that the different values used as the learning rate η impact the scores. A learning rate set at $\eta = 0.0.1$ and $\eta = 0.05$ gives lower values for the MSE scores, and higher values for the R2 score than a higher value of η. This result is somewhat expected because the lower step size creates a more numerically stable convergence of the coefficients. In reality, however, we know the limitations of this small step size increase the cost and reduce the efficiency, as it needs to do more iterations because of the smaller step size. Another limitation of the learning rate is that there is only one representing all coefficients of the dataset, even though the coefficients may vary in size and direction, which is not ideal.

A solution to these limitations is to include optimization methods for updating the learning rates, as shown in Figures 8 and 9. The optimization methods update the learning rate based on the previous iterations, as explained in the methods section. By comparing these methods, one can see that the different optimizations help to stabilize the model and lower the values of the MSE. However, the R2 scores have significantly decreased, indicating a poor performance of the model, likely due to the noise of the dataset. When removing the noise, the R2 scores significantly increase in value, especially for the Ridge Regression. The Ridge Regression seems to adapt even better to a function without noise, though the fluctuations show that the complexity of the polynomial still produces results of higher variance and low bias. One can argue that neither the OLS nor the Ridge method is ideal due to the given polynomial degree and polynomial complexity. The fluctuations appear more significant in Figure 8 for the OLS method, so we could argue that the Ridge Regression is still the better choice of the two.

The AdaGrad (Green line) proves to be the best optimization method, as it lowers the MSE-values and have the best, though not ideal, R2 scores for both the OLS Regression and the Ridge Regression. The performance of the different optimizers is more even for the Ridge Regression, while for the OLS model, we observe large variations between the resulting values and scores.

Because the gradient descent is an expensive method for larger datasets, dividing the data into smaller subsets is ideal for performing the gradient descent in a profitable manner. By using stochastic gradient descent, we reduce the cost while also reducing the risk of the gradient descent converging to the local minima, rather than the global minima. We see in Figures 11 and 14 for the gradient descent of OLS and the Stochastic gradient descent of OLS, respectively, that the gradients converge faster for the stochastic descent because of the use of subsets rather than the whole dataset. The same behavior is found when repeating the process for the Ridge and the Lasso regression.

For the Lasso and Ridge regressions, the stochastic gradient descent leads to more fluctuations than what is found in the regular gradient descent method. This is likely due to us working with individual training examples, so each iteration (epoch) is made with new data, generating more noise and fluctuations. This is more evident for the Lasso and Ridge results than the OLS results.

When writing the methods section, we were prepared for the stochastic gradient descent to predict less accurate results than the standard gradient descent, based on the literature used for preparing the theory. It is therefore worth mentioning an interesting result, as we find the coefficient values θ to be more accurate in Figure 15 with the stochastic gradient descent for the Ridge Regression, than in Figure 12 for the standard gradient descent.

This could be due to a number of reasons, but a plausible theory is that dividing the data into smaller subsets makes it easier for the algorithm to escape a saddle point of a local minima, which is known to be a limitation of the gradient descent method. Another plausible theory might be that the added noise in the stochastic method might help in producing well-predicted data. For future projects, it might be interesting to further study the accuracy between the two methods.

For the boostrapping method, the resulting values for the MSE in Figure 16 appear similar to Figure 1 in shape and form, and still show clear signs of overfitting and higher variance in our data. This makes sense, as Bootstrapping does not reduce the MSE values, but rather reinforces the knowledge we have already obtained from the previous Figures by introducing stability and reliability to our results. This is confirmed because the training data decreases in MSE value as it adapts to the noise of the dataset, while the test data starts increasing near a polynomial degree of 10, as seen in all our previous results.

For the cross-validation results, however, we see a few changes and interesting trends in our results found in Figure 17. When the size of the dataset is larger than 300, we see that the training data keeps decreasing in MSE values, while the test data does not display the same spike as seen in earlier Figures for the OLS. This is as expected, as the training of independent subsets reduces the variance of the error. This is an indication that the cross-validation reduces overfitting to a certain degree. For lower values of n, there is still a sudden increase in the test data, displaying a higher variance in the dataset. That being said, the MSE values remain relatively high for all sample sizes, so it is still not an ideal polynomial fit. However, it could be interesting to run the cross-validation for even larger datasets (larger n), to see how this would impact our results further. It is important to acknowledge the limitations of the resampling methods, as using resampling can be computationally expensive over time by creating multiple datasets and performing the same recalculations.

V. CONCLUSION

To conclude, all the results presented in this report indicate that Runge's function is not an easy polynomial to fit. All the methods used through regression or gradient descent show the same trend - low R2 scores and higher values for the MSE. This indicates that the model is performing poorly and that the error between the predicted values and the true values of the dataset remains relatively high throughout the report. Runge's function, used with a polynomial degree of 15 or higher, shows that there is too much complexity in our data. It is also worth mentioning that the noise in our dataset also impacts the results, as we have added noise to our function following a normal standard distribution. From the start, we have tried removing the noise to evaluate whether the less-than-ideal results are based on our function alone, or if the noise is contributing actively in increasing the error and worsening the performance of the model. The noise has a big impact on our results, and our errors decrease and our R2 score performs better when the noise is removed. Removing the noise is not a realistic fix, though, as a real dataset will naturally introduce noise and fluctuations. We have therefore chosen to keep the noise in all our results to reflect on how well our methods will work on real life data.

The key takeaways from this report are that the introduction of a regularization parameter λ improves the polynomial fit by shrinking the coefficients towards zero, reducing the variance, but at the same time increasing the bias. Therefore, the Ridge Regression is the better fit for our polynomial function. The errors also decrease when we increase the sample size. Introducing the stochastic gradient descent also makes the error converge faster to the local minima of the cost function, and even produces more accurate results than using the standard gradient descent for the Ridge gradient descent. Resampling methods support our findings for the gradient descent and regression methods by producing the same results for the OLS function as in the start of the report. It provides us with statistical verification that the polynomial fit of the Runge's function consists of high variance and lower bias. Therefore, we conclude that in reality, none of these models provides a good fit for the Runge's function and are therefore not ideal methods to use on real-life data with the same characteristics.

VI. LINK TO GITHUB REPOSITORY

The report and all Python scripts are accessible through the following link: `https://github.com/Malenefk/MachineLearning_GroupBME/tree/main/Project1`

[1] M. Hjorth-Jensen, *Computational Physics Lecture Notes 2025* (Department of Physics, University of Oslo, Norway, 2015), URL `https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/chapter1.ipynb`.

[2] GeeksforGeeks, *Momentum based gradient optimizer - ml* (2025), URL `https://www.geeksforgeeks.org/machine-learning/ml-momentum-based-gradient-optimizer-introduction/`.

[3] W. C. D. A. G. R. . R. C. Mehta, P., Boston University - Physics department (2024).

[4] K. T. L. S. . M. B. Fahrmeir, L., *Regression - Models, Methods and Applications* (Springer, 2013).

[5] GeeksforGeeks, *Adagrad optimizer in deep learning* (2025), URL `https://www.geeksforgeeks.org/machine-learning/intuition-behind-adagrad-optimizer/`.

[6] GeeksforGeeks, *Rmsprop optimizer in deep learning* (2025), URL `https://www.geeksforgeeks.org/deep-learning/rmsprop-optimizer-in-deep-learning/`.

[7] GeeksforGeeks, *Bias-variance trade off - machine learning* (2025), URL `https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/`.

[8] 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`.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Journal of Machine Learning Research **12** (2011).

[10] GeeksforGeeks, *How to implement adam gradient descent from scratch using pyton* (2025), URL `https://www.geeksforgeeks.org/machine-learning/how-to-implement-adam-gradient-descent-from-scratch-us`