Exploration of Regression with the Runge Function
FYS-STK3155 - Project 1

Max Følstad-Andersen, Hallvard Hareide, Yang Liu, William Schjerve Moe
University of Oslo
(Dated: October 6, 2025)

Regression methods are some of the most fundamental methods for machine learning. However, the classical methods such as Ordinary Least Squares (OLS) are often subject to overfitting and poor generalization. The Runge function illustrates this, where interpolation using higher degrees of polynomial does not improve the fit. To tackle these problems, we analyze the regularized regression techniques Ridge and Lasso, in addition to resampling methods such as Bootstrap and cross-validation.

In this project, OLS, Ridge and Lasso regression were first implemented followed by gradient decent methods such as momentum, ADAgrad, RMSprop and ADAM. Using these methods the model performance was evaluated and analyzed for different polynomial degrees. To measure the model, mean squared error (MSE) was used.

The main results are that the gradient descent model which best predicted our data was a 6th degree polynomial trained with the gradient of OLS and the optimizer RMSprop. However, this model predicted with three times higher MSE than a closed form OLS model. This can mean that the search space for our numerical approximation should have been wider.

Our findings show that the Runge function is inherently prone to overfitting. The quantifications of this, and methods to avoid it, should be considered in other projects to enhance model performance.

I. INTRODUCTION

Regression methods are among the fundamental building blocks for machine learning. Their simplicity and systematic approach make the methods widely used for modeling relationships between variables. However, the simplest techniques such as linear regression, face multiple challenges such as overfitting and poor performance when applied to complex or limited datasets [1]. Thus, regularized versions such as Ridge and Lasso regression, in addition to resampling techniques such as Bootstrap and cross-validation are used to address these issues. By using these methods, it is also possible to get an insight into the tradeoffs between bias and variance [2].

In this project, the focus is on the Runge function, which is a function that shows the Runge Phenomenon 1. Carl David Tolme Runge showed that polynomial interpolation of this function using increasing degrees does not improve the fit, but instead the error often increases [3]. This despite the fact that there exists a sequence of polynomials that converge uniformly to the Runge function [4]. As it turns out interpolation polynomials is not such a sequence.

The Runge Function:

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

By fitting polynomials of different degrees using OLS, Ridge and Lasso regression, the performance of the model will be evaluated for the different complexities. Moreover, gradient decent methods, including momentum, ADAgrad, RMSprop and ADAM, will be implemented as iterative alternatives to the closed-form solutions. Finally, resampling techniques such as Bootstrap and cross-validation will be explored to understand the bias-variance tradeoff.

This report is divided into four sections and organized as follows. Section II: Methods describes the theoretical backgorund and implementation details of the used models. Section III: Results and Discussion presents, analyzes and discusses the obtained results for different methods, polynomial degrees, data sizes and parameters. Section IV: Conclusion concludes our main findings and summarizes their implications.

II. METHODS

A. Regression methods

The goal of regression is to predict the relationship between variables. This is done by fitting an unknow function $f(x)$ to a known dataset, which consists of inputs $x^T = [x_0, x_1, .., x_{n-1}]$ and outputs $y^T = [y_0, y_1, .., y_{n-1}]$. For the setup, it is assumed that the output data that is supposed to be modelled can be represented as

$$y = f(x) + \epsilon \tag{2}$$

Here f is a continious function and $\epsilon \sim N(0, \sigma^2)$ represent noise.

To approximate $f(x)$, a model of polynomial degree p is used and gives the prediction

$$\tilde{y} = X\theta \tag{3}$$

In (3) X is the design matrix, where each row is a data point and each column is a feature of that data. It

is common to also include an intercept column, which tries to account for constant effects in the data that are not dependent on anything else. Furthermore, $\theta^T = [\theta_0, \theta_1, .., \theta_{n-1}]$ are unknown parameters. The aim is to find the parameter values that gives the best model quality, which can be measured by for example the *Mean Squared Error* measure. To find these θ-values, the approach is to minimize a cost/loss function. There are numerous such functions and in this project we consider the three variants OLS, Ridge and Lasso.

1. Ordinary least squares

In OLS the cost/loss function is defined as the mean squared error (MSE) between the predicted and true values.

$$C_{OLS}(\theta) = \|y - X\theta\|^2 \tag{4}$$

Minimizing this function with regard to θ, the solution obtain is

$$\hat{\theta}_{OLS} = (X^T X)^{-1} X^T y \tag{5}$$

2. Ridge regression

Ridge regression is based on OLS and introduces a regularization hyperparameterλ as a L_2 penalty term. Thus, the ridge cost/loss function is

$$C_{Ridge}(\theta) = \|y - X\theta\|^2 + \lambda\|\theta\|_2^2 \tag{6}$$

and optimal parameters

$$\hat{\theta}_{Ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{7}$$

3. Lasso regression

Lasso regression is also based on OLS, but introduces a L_1 penalty term instead. The Lasso cost/loss function is

$$C_{Ridge}(\theta) = \|y - X\theta\|^2 + \lambda\|\theta\|_1 \tag{8}$$

By minimizing the cost/loss function, it is not possible to derive a closed form solution. This is because... To find the optimal parameters, iterative optimization methods are required instead. In this project gradient descent will be used to implement Lasso regression.

4. Gradient descent

Gradient descent (GD) is an iterative optimisation algorithm that updates model parameters in the direction of the negative gradient of the cost function. The goal is to minimise the cost function, such as the MSE in OLS and Ridge regression. For an arbitrarily chosen starting point, the slope will be steeper, but as we update the parameters, the steepness will gradually descend until it reaches the lowest point on the curve, known as the point of convergence [5]. To do this, we require a direction and a learning rate.

The learning rate, α, is the size of the steps taken to reach the minimum, and is fixed throughout the iteration. Typically this is a small value, that gradually becomes smaller as we reach the point of convergence. High learning rates will result in larger steps, and may thus reach this point earlier, but it also risks overshooting the minimum. Smaller learning rates will achieve higher precision, but will need more iterations.

To find the direction of the GD, one uses the gradient of a cost function. Finding this gradient can be computationally expensive for large datasets. One solution to this is to calculate the gradient using small subsets of the data called "mini-batches," more on this later. The cost function can be, for example, the same as in OLS or Ridge, as it is in our case. It continuously iterates, moving along the direction of the steepest descent until it is close to zero, at which point the model will stop learning. One of the difficulties with GD is knowing whether one has found the true global minimum, or simply a local minimum or a saddle point - both cases where the gradient will be zero.

There are several methods to address this issue. One of the methods we have implemented in this project is called Stochastic Gradient Descent (SGD). It runs a training epoch for each example within the dataset, updating the training example's parameters one at a time. In most cases, as in ours, only a subset of the examples are chosen through mini-batching. The underlying idea comes from an observation that the cost function can almost always be written as a sum over n data points $\{x_i\}_{i=1}^n$ (Week 37):

$$C(\theta) = \sum_{i=1}^{n} c_i(\mathbf{x}_i, \theta) \tag{9}$$

Hence, the gradients can be computed as a sum over i-gradients:

$$\nabla_\theta C(\theta) = \sum_{i}^{n} \nabla_\theta c_i(\mathbf{x}_i, \theta) \tag{10}$$

The stochasticity is introduced by only taking the gradient on a subset of the data, called mini-batches. For n data points and M mini-batches, there will be a total of $\frac{n}{M}$ mini-batches, denoted as B_k, for $k = \{1, 2, \ldots, \frac{n}{M}\}$.

The frequent updates from SGD can offer more detail and speed, but can also be less computationally efficient, when compared to another method we have also implemented; Batch Gradient Descent. The principle of Batch Gradient Descent is to compute the gradient using only a

small subset of the examples, i.e. update the parameters using say a couple of thousand examples, taken from an entire training set of millions.

5. *Resampling methods*

Resampling techniques are statistical methods used to assess the accuracy and reliability of models by repeatedly drawing samples from a dataset. Two widely used methods in machine learning are bootstrapping and cross-validation, which we have implemented in this study.

Bootstrapping generates many "resampled" datasets by sampling with replacement from the original dataset, allowing estimation of variability, confidence intervals, and sampling error, without assuming the specific population distribution [6]. In essence, we can repeat our experiment a number of times for statistical robustness, without needing to generate new data, but simply by reshuffling what we already have.

Cross-validation on the other hand, partitions the data into subsets or folds to iteratively train and test the model. Each fold serves as a test once, and the averaged results provide an unbiased estimate of the predictive performance of the model. The method we have implemented is known as K-folds cross-validation, which splits the training set into k smaller sets. The performance measure reported by k-fold cross-validation is then the average of the values computed by each individual k-fold [7]. The distribution of the performances can also be used to compute empirical estimates of standard deviation. K-fold cross-validation can be computationally expensive, but does not waste too much data.

Resampling techniques also allow for a practical foundation in understanding what is known as the bias-variance trade-off. By repeatedly evaluating a model on resampled datasets or folds, we can observe how changes in the model complexity affect the error, which is directly related to the decomposition of the MSE into bias and variance. See below how this can be derived from the cost function:

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] \tag{11}$$

We use that $y = f(x) + \epsilon$. Plugging this in we get

$$\mathbb{E}[(y-\tilde{y})^2] = \mathbb{E}[(f(x)+\epsilon+\tilde{y})^2] \tag{12}$$

Now add and subtract $\mathbb{E}[\tilde{y}]$

$$= \mathbb{E}[(f(x) - \mathbb{E}[\tilde{y}] + \mathbb{E}[\tilde{y}] - \tilde{y} + \epsilon)^2] \tag{13}$$

Expanding

$$= \mathbb{E}\Big[(f(x)-\tilde{y})^2 + (\mathbb{E}[\tilde{y}]-\tilde{y})^2 + \epsilon^2 + 2f(x)\mathbb{E}[\tilde{y}](\mathbb{E}[\tilde{y}]-\tilde{y}) + 2(f(x)-\mathbb{E}[\tilde{y}])\epsilon + 2(\mathbb{E}[\tilde{y}]-\tilde{y})\epsilon\Big] \tag{14}$$

The terms with ϵ vanish, and so do the cross terms because we have $\mathbb{E}[\mathbb{E}[\tilde{y}] - \tilde{y}] = \mathbb{E}[\tilde{y}] - \mathbb{E}[\tilde{y}] = 0$
We are thus left with

$$\mathbb{E}[(y-\tilde{y})^2] = \mathbb{E}[(f(x) - \tilde{\curvearrowright})^2 + (\mathbb{E}[\tilde{y}]-\tilde{y})^2 + \epsilon^2] \tag{15}$$

Using the given terms for bias 16 and variance 17:

$$\text{Bias}[\tilde{y}] = \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] \tag{16}$$

$$\text{var}[\tilde{y}] = \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] = \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2. \tag{17}$$

we have that

$$\mathbb{E}[(y-\tilde{y})^2] = Bias[\tilde{y}] + Var[\tilde{y}] + \sigma^2 \tag{18}$$

The bias term shows how much the average prediction deviates from the actual function. A high bias would thus indicate that the model is systematically wrong, for example due to underfitting. The variance term measures how much the model's predictions deviate from the mean of the model's predictions, and is typically related to overfitting The noise term is an irreducable error from the randomness in the data.

B. Implementation

1. *The data*

The data analyzed were 100 points sampled from the Runge function. The points were linearly distributed between -1 and 1. We also added normally distributed noise with variance 0.1.

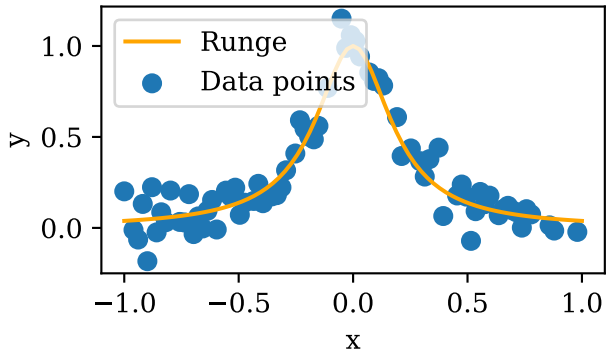

Figure 1: Train data used for the Runge function plotted with the Runge function.

Before training any models, we used a number of different data preprocessing procedures. To place features on a comparable scale, we scaled our data using `normalize` from Scikit-learn. Furthermore, the data was also split into a testing set and training set, with the use of Scikit-learn's function `train_test_split`. The training data consisted of 80 % of the original data, while the test data consisted of the remaining 20 %.This allows us to evaluate the model on unseen data, which is essential for assessing generalisation performance. This is done, as the model may achieve near-perfect performance on training data due to overfitting, but not perform well on unseen data.

2. Constraints

When implementing linear regression, there are a lot of parameters which one is free to set, and which will impact the effectiveness and result of the training. In order to focus on the various gradients and optimizers, we have fixed these parameters. The following is a summary of the constraints we have put on our models.

The gradients we have taken into consideration are the gradients for OLS, Ridge and Lasso regression. Both Ridge and Lasso require a regularization parameter, the parameter used was 0.01 in all experiments for both Ridge and Lasso.

All gradient descent implementations must pick a number of iterations, epochs, which the training should stop at. We fixed it to 100. However, if the MSE of the test data and the predicted data changed less than the learning rate divided by 1000 for 10 consecutive epochs, we would stop training prematurely.

The optimizers used were ADAM, RMSprop, ADAgrad, momentum, as well as an implementation without an optimizer. With the momentum optimizer, the momentum parameter was fixed to 0.9. The decay parameter of RMSprop was fixed to 0.99. For ADAM, the decay parameter β_1 was fixed to 0.9 and β_2 to 0.999.

All the optimizers require an initial learning rate, in all experiments we used an initial learning rate of 0.05.

The exact values we have picked for these constraints are somewhat arbitrary, but all values are close to the values that are standard for implementation. The optimizer parameters are, for example, the default values that pytorch's gradient descent uses [8][9].

3. Gradient descent

In order to test the different gradients and optimizers in gradient descent, we ran tests for each combination and logged the results. For each experiment, the MSE was computed on the test data and the final MSE was logged.

The gradients tested were OLS, Ridge and Lasso and the optimizers tested were ADAM, ADAgrad, no optimizer (Simple), momentum and RMSprop. All combinations of these gradients and optimizers were tested for polynomials of degree 2, 4 and 6. The reason to only use even degrees is that the Runge function is even, i.e. it is symmetric around the y-axis.

The two best-performing models were trained with stochastic gradient descent to refine the models and try to improve the MSEs. The resampling of stochastic gradient descent was done with the *resample* function of Scikit-learn with randomization parameter set to 1.

4. Validation with resampling techniques

As a display of the general tendency of the Runge function to get overfitted, we train a model on different size samples of our data across different model complexities. We train a model using the analytical solutions of OLS in order to minimize error from other sources. We test samples of both size 20 and 80 (the whole train data set), and compare the error of our predictions of the test data with the error of the predictions of the train data. This is to see how the size of the data set affects the bias-variance decomposition.

We also implement the resampling technique bootstrapping in order to decompose the observed MSE into bias and variance, as derived in equation 18.

We also use resampling techniques to validate and expand on the performance we quantified in II A 5. We use 5-fold cross-validation to quantify the expected MSE and standard deviation. We also perform a bias-variance decomposition for this model.

5. Programming tools

All code relevant for this project has been written in the programming language Python (version later than 3.12). We relied on libraries such as Numpy [10] and Scikit-Learn [11], among others. Scikit-Learn was used both as a benchmark for our implementations of OLS, Ridge and Lasso, and Gradient Descent, and to provide tools for model evaluation such as k-folds cross validation. Aditionally, Pandas was used for creating Latex code [12] and Matplotlib [13] was used for plotting functions. GitHub was used for version control and as a cloud-based repository, ensuring collaboration and reproducability of the code [14].

6. Code structure

The code revolves around gradient descent. The main training loop of gradient descent is implemented as a class. In this class one can pick a gradient from class containing all gradients and an optimizer from the class containing all optimizers. The gradients implemented are OLS, Ridge and Lasso. The optimizers implemented are

ADAM, ADAgrad, Momentum, No optimizer (Simple), RMSprop. Additionally cross-validation is implemented separately.

7. *Demonstration and benchmarks*

In order to validate the performance of the best models, the closed form solutions of OLS and Ridge and the numerically calculated optimal parameters of Lasso were used. The optimal parameters were calculated using Scikit-Learns OLS, Ridge and Lasso classes.

C. Use of AI tools

During the implementation of the code Artificial intelligence such as ChatGPT were used for error correction and debugging. In particular, AI were used for help in clarification of error messages. The immediate feedback accelerated the debugging phase which potentially saved a lot of time spent on finding and correcting minor issues in the code. Code for plotting and the ADAM optimizer were partially written by ChatGPT, in the rest of the code, no AI-generated code is used.

III. RESULTS AND DISCUSSION

A. Results

The main purpose of this article is to find a model to approximate the Runge function. We focus on using polynomial functions like OLS, Ridge and Lasso. The search space is quite limited since we use strict limitations for parameter values. The main target is to train a model on the training data that gives the smallest MSE on the test data.

1. *Results from gradient descent*

The main experiments were run on OLS, Ridge and Lasso with the optimizers ADAM, ADAgrad, no optimizer, momentum and RMSprop with 2, 4 and 6 features. The final MSE of the experiments with the different gradients, optimizers and number of features are shown in table I II III. The two parameter combinations with the lowest MSE is OLS-RMSprop with 6 features at a MSE of 0.0350 and OLS-RMSprop with 4 features at a MSE of 0.0420.

Table I: Final test MSE for gradient descent. Showing gradient-optimizer combinations with 2 features.

	OLS	Ridge	Lasso
No optimizer	0.0870	0.0742	0.0582
ADAM	0.0547	0.0501	0.0488
RMSprop	0.0541	0.0495	0.0489
ADAgrad	0.0628	0.1174	0.2501
Momentum	0.1239	0.0680	0.0957

Table II: Final test MSE for gradient descent. Showing gradient-optimizer combinations with 4 features.

	OLS	Ridge	Lasso
No optimizer	0.1129	0.0562	0.1073
ADAM	0.0591	0.0589	0.0499
RMSprop	0.0420	0.0482	0.0574
ADAgrad	0.0820	0.2255	0.2428
Momentum	0.0738	0.0796	0.0666

Table III: Final test MSE for gradient descent. Showing gradient-optimizer combinations with 6 features.

	OLS	Ridge	Lasso
No optimizer	0.0741	0.1059	0.0934
ADAM	0.0515	0.0522	0.0478
RMSprop	0.0350	0.0452	0.0439
ADAgrad	0.0997	0.2512	0.3791
Momentum	0.1176	0.0759	0.0873

The model with the lowest MSE, i.e. OLS with RMSprop and 6 features is plotted in figure 2 against the Runge function. For the model, the points plotted are the test data.

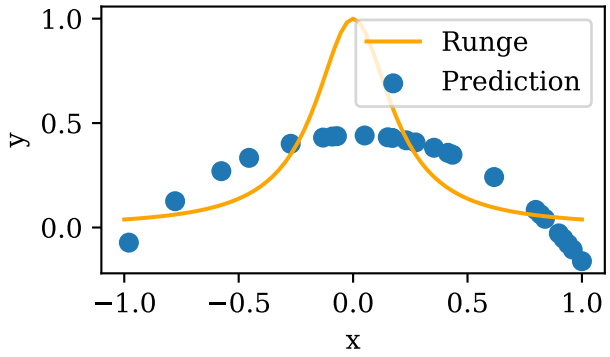

Figure 2: RMSprop with OLS and 6 features plotted with the Runge function.

The models with the lowest MSEs, i.e. OLS with RMSprop and 6 and 4 features were retrained with stochastic gradient descent and the same parameters. The final MSE was 0.0350 for the model with 6 features and 0.0355 for the model with 4 features. This means no decrease in MSE for the model with 6 features and a 18 % improvement for the model with 4 features.

2. *Benchmark values*

The MSE of the test data was computed on the closed form solution of OLS and Ridge and the optimal value for Lasso with the model complexities 2, 4 and 6. The MSEs are shown in table IV. The smallest MSE value is 0.0163 which is 3 times smaller than the smallest MSE obtained with gradient descent.

Table IV: Test MSEs from closed form solutions. Gradient plotted against number of features.

	2	4	6
OLS	0.0542	0.0262	0.0163
Ridge	0.0541	0.0263	0.0205
Lasso	0.0478	0.0478	0.0478

3. *Overfitting*

Figure 3 and 4 shows the model overfitting with higher polynomial degree. The model model trained on only 20 data points overfits more than the model trained on 80 datapoints.

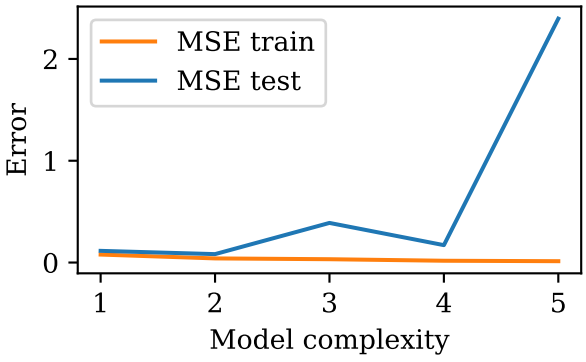

Figure 3: Train and test MSE form closed for soluton with 20 data points.

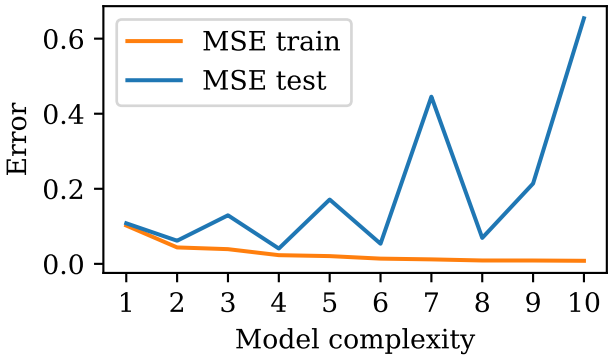

Figure 4: Train and test MSE for closed for soluton with 80 data points.

4. *Bias-variance tradeoff*

Figure 5 illustrates the bias-variance trade off for a fitted model to the Runge function. The model fitted is the closed form solution of OLS.

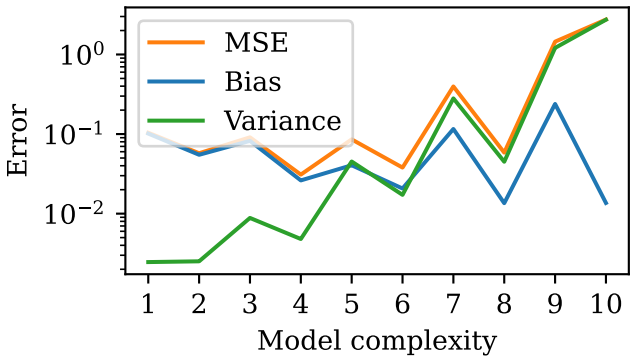

Figure 5: Bias-variance trade off for closed form solution of OLS.

The bias-variance trade of for RMSprop with 6 features for different gradients is shown in table V.

Table V: Bias-variance trade of for RMSprop with different gradients and 6 features.

	MSE	Variance	Bias
OLS	0.0217	0.0048	0.0169
Ridge	0.0493	0.0028	0.0464
Lasso	0.0468	0.0042	0.0426

5. *Cross validation*

Table VI shows the emperical MSE mean and standard deviation computed using bootstrap and cross validation,

for RMSprop, OLS and 6 features, compared with the test MSE.

Table VI: MSE for the test data, MSE mean and standard deviation with bootstrap and cross validation with RMSprop, OLS and 6 features.

	Test MSE	Bootstrap MSE	Cross val. MSE
OLS	0.0350	0.0216±0.0691	0.0316±0.0125
Ridge	0.0452	0.0493±0.0532	0.0539±0.0192
Lasso	0.0439	0.0467±0.0647	0.0487±0.0198

B. Discussion

1. Comparison of gradient descent models

The main result is that OLS with RMSprop with 6 features performed the best followed by OLS with RMSprop with 4 features.

No optimizer generally had the highest MSE and RMSprop generally had the lowest MSE. It is expected that a gradient descent with optimizers outperforms a gradient descent without optimizer. Furthermore momentum also performed a lot worse than the other optimizers, this again is due to the fact that momentum is a lot less sophisticated than the other methods.

In many cases OLS performed better than Ridge and Lasso, this may be caused by Ridge and Lasso generally being less accurate than OLS while imporving numerical stability. In the case of the Runge function for the number of featured tested, numerical instability was not a problem.

When we compare the lowest MSE obtained; 0.0350 with the MSE of the closed form solution; 0.0163, gradient descent gets outperformed. This is probably caused by us doing some strict limitations. In all the experiments we used the same fixed values for initial learning rate and epoch. We used a fixed algorithm to determine early stopping and in all the optimizers we used fixed parameters. Additionally we used a fixed randomization seed.

A way to improve the models could be to expand the search space and consider more initial learning rates and different parameters for the optimizers. Since the seed was set the poor results could also be caused by chance and other seeds could have performed way better. Different seeds should also be explored for future research.

2. Stochastic gradient descent

With RMSprop OLS and 6 features, stochastic gradient descent did not improve the MSE. In the case of 4 features it improved the MSE by 18 %. This increase is substantial, but it unfortunately cannot be guaranteed as shown by the model with 6 features. But stochastic gradient descent could improve a lot of results and future research should consider training models both with and without stochastic gradient descent.

3. Overfitting and the Runge function

Figure **??** shows that although train data can be fitted to perfection, a more complex model will predict test data worse due to overfitting.

The Runge function is continuous, and thus there exists a sequence of polynomials that converge uniformly to it [4]. But, as shown by Carl David Tolmé Runge, interpolations of this polynomial do not have this property of converging uniformly [3]. In many cases, the sequence does not converge at all, and the error increases with the polynomial degree. Hence, the Runge function is very prone to overfitting. This is the reason we used a small number of features and also the reason why we do not expect the MSE to be 0 for any of the models.

We can detect that a model is overfitted if the predictions of the train data improves, but the predictions on the test data stagnates or worsens. This is clearly displayed in figure 3 and 4. They show that overfitting occurs more with a model trained on fewer data points and more complex models. Hence, a way to reduce overfitting is by reducing the number of features or training on more data. Since our models already are quite simple, being between 2 and 6 features, the main way of reducing overfitting is by training on more data.

In addition to showing overfitting, the figures also confirm what we assumed at the outset. When chosing what parameters to vary, we argued that fitting polynomials of even degrees would give better models. The oscillation observed between even and odd degrees in figures 3, 4 and 5 confirm this assumption, and show that polynomial models of odd degrees perform systematically worse. All these models are trained using a closed form expression for the parameters given by OLS. We suspect that training models using Ridge or Lasso regression will mute this effect. This is because these models apply a feature size penalty to all features. This penalty could reduce unimportant features, such as those of odd degree, effecively turning a model of odd degree into one of a degree one less.

4. Bias-variance tradeoff

Another way to quantify overfitting is by analyzing a model through the lens of the bias-variance trade-off. This is displayed for polynomials fitted to the Runge function in figure **??**. Here, bias and variance play dual parts in how they affect the MSE. In general, a less complex model will have a smaller range of values they can express. Less complex models will be less affected by noise for the same reason, as it cannot be fully fit to reproduce the noise. This results in a systematic tendency

to output averaged results, and is what is quantified by bias. Less complex models are biased towards the average because they lack the complexity of predicting complicated relationships.

On the other hand, when models become too complex, they become prone to overfitting. This is because the increased complexity is used to learn "patterns" in the noise. Of course, these patterns do not exist, and will therefore not make predictions any better. In fact, it usually makes predictions worse. Models trained on slightly different datasets will predict very different results. This tendency is captured and quantified by variance. More complex models become trained to fit the noise, and their predictions will therefore vary depending on the dataset at hand.

5. Emperical estimation through resampling

Our results on model performance gain a new dimension when we consider the emperical standard deviation of the test MSE. This is shown in table VI. For each of the three models, the test MSE, the mean MSE measured with bootstrap, and the mean MSE measured with cross-validation, are all different, but lie within the respective emperical standard deviations. The ordering of the models in terms of MSE is also preserved across all estimates, with OLS admitting the lowest MSE, and Ridge admititng the highest.

The standard deviation for the estimates of cross validation are far lower than those for bootstrap. This can be because bootstrap samples points with replacement, and since our dataset is small, bootstrap samples might be unrepresentative of our original data set. Cross-validation on the other hand, partitions the data into train and test differently for each fold. Since it uses all the data every time, it could be more likely to be representative. We also only do 5-fold cross-validation, which means our standard deviation is computed with 5 data points. It is not too unlikely that these 5 datapoints turn out very similar. However, since this result is consistent across all the different gradients, we can conclude that cross-validation gives more accurate estimates of MSE.

6. Further research

The main way to improve the model for the Runge function would be to expand the search space. One could consider different learning rates and randomization seeds. Additionally future research should look into using different parameters for the optimizers and the Ridge and Lasso gradients. One could also look into the expanded use of stochastic gradient descent. Our results show that increasing the number of features may lead to overfitting and should hence be avoided. Hence the search space for this parameter does not necessarily need to be changed. Furthermore future research could look into improving the data, we used 100 data points with some noise. Improved data quality would improve the results. In our case this is just a case of generating more data. Lastly one could look into more advanced machine learning methods. Since polynomials have been proved hard to fit to the Runge function, due to sequences of interpolation polynomials not converging, but rather diverging with higher model complexity. One could look into using Neural networks which can be fitted to more general functions since it is non-polynomial. The MSE of the closed form solution of OLS shows that one can only achieve an MSE 0.0163 for the number of features tested, at least for OLS. Similar results probably hold for both Ridge and Lasso. Hence to achieve a close to perfect model of the Runge function, simple methods like OLS, Ridge and Lasso are not sufficient.

IV. CONCLUSION

In this article we have tried to approximate the Runge function with different regression methods. The focus have been on Ordinary Least Squares (OLS), Ridge and Lasso regression. Models using the gradients from these methods were trained with gradient descent and evaluated using the closed form solution of OLS and Ridge and the optimal parameters for Lasso. With the gradient descent the optimizers Adam, ADAgrad, RMSprop and momentum were used as well as gradient descent without optimizer. Additionally stochastic gradient descent was used on the two best performing models. Finally the resampling techniques bootstrapping and cross-validation were used to the model robustness and the bias-variance trade-off.

The model with the best performance was RMSprop with OLS and 6 polynomial features achieving a Mean Square Error (MSE) of 0.0350. Though this result was around 3 times higher than the closed form solution of OLS. This indicates that a larger search space should have been used were more parameters were altered. Furthermore, stochastich gradient descent improved the MSE with 18 % in one of the cases and could be used with other models to improve results.

The results showed the well known behaviour of the Runge function; increased model complexity does not necessarily lead to a better model. The bias-variance decomposition also showed the same effect with the model overfitting with higher model complexity.

Bootstrap and cross validation supported the findings. Showing that OLS consistently performed better than Ridge and Lasso. Cross-validation got better standard deviation estimates than bootstrapping, suggesting it is the more reliable resampling method for this problem.

Overall, this study shows that fitting a polynomial to the Runge function is difficult, but gradient descent can achieve MSE scores close to the closed form solution. One could imporve performance by changing parameters but the room for improvement is limited due to the nature

of the Runge function. Where higher model complexity does not necessarily improve performance. Hence more complex approaches may be needed to achieve better results. This could, for example include methods like deep Neural Networks.

[1] S. Kumar and V. Bhatnagar, Journal of Intelligent Systems and Computing (2022), URL https://en.wikipedia.org/wiki/Regression_analysis.

[2] 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.

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

[4] T. L. Lindstrøm, *Spaces - An Introduction to Real Analysis* (American Mathematical Society, 2010).

[5] IBM, IBM Think (N/A), URL https://www.ibm.com/think/topics/gradient-descent.

[6] I. Castro, D-Lab UC Berkely (2021), URL https://dlab.berkeley.edu/news/beginner%E2%80%99s-guide-bootstrap.

[7] Scikit-learn, *Cross-validation: evaluating estimator performance* (N/A), URL https://scikit-learn.org/stable/modules/cross_validation.html.

[8] URL https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[9] URL https://docs.pytorch.org/docs/stable/generated/torch.optim.RMSprop.html.

[10] URL https://numpy.org/.

[11] URL https://scikit-learn.org/stable/.

[12] T. pandas development team, *pandas-dev/pandas: Pandas* (2020), URL https://doi.org/10.5281/zenodo.3509134.

[13] J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).

[14] URL https://github.com.