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 7, 2025)

Abstract

Regression methods are among the most fundamental methods for machine learning. However, classical methods such as Ordinary Least Squares (OLS) are often subject to overfitting and poor generalization, particularly in polynomial regression. This project investigates how regularization and optimization techniques mitigates these issues, using the Runge function as a test case for overfitting. We implement and compare OLS, Ridge and Lasso regression, along with gradient-based optimization methods including Momentum, ADAgrad, RMSprop, and ADAM. Model performance is evaluated across different parameters using the mean squared error (MSE) and the coefficient of determination (R^2). The main results are that regression models are prone to overfitting the Runge function, particularly OLS. Ridge dampens this effect. For the closed-form solutions, we find that Ridge achieved around 0.001-0.070 better R^2 score compared to OLS. Additionally, we find that the different parameters for the optimizer and gradient in gradient descent are key in fitting a model. However, our numerical models still yielded an MSE more than three times higher than the closed-form OLS solution, suggesting that the numerical optimization may have been restricted by an insufficiently broad search space. More advanced methods like neural networks may be needed for a model with lower MSE.

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, linear regression and other simple techniques 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 trade-offs 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

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

The OLS model is highly sensitive to noise in the data. When the model complexity increases, for example when using high degree polynomial features, the θ-values can become large values in order to fit the training data closely. This way it captures noise instead of the underlying pattern, which can lead to poor generalization for unseen data [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

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

By including the penalty term $\lambda\|\theta\|_2^2$ it penalizes large parameter values, and thus reduces overfitting. Larger λ creates stronger regularization by shrinking the coefficients towards zero [6].

3. *Lasso regression*

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

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

Contrary to Ridge, the L_1 penalty can set coefficients exactly to zero, leading to the possibility of eliminate features. By minimizing the cost/loss function, it is not possible to derive a closed form solution. This is because the L_1 norm is not differentiable at zero. To find the optimal parameters, iterative optimization methods are required instead [7]. 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 [8]. 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 $\{x_i\}_{i=1}^n$ [9]:

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

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 $k = \{1, 2, \dots, \frac{n}{M}\}$.

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 [10]. 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 [11]. 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:

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

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

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 Gausian noise with mean 0 and variance 0.1 to the data.

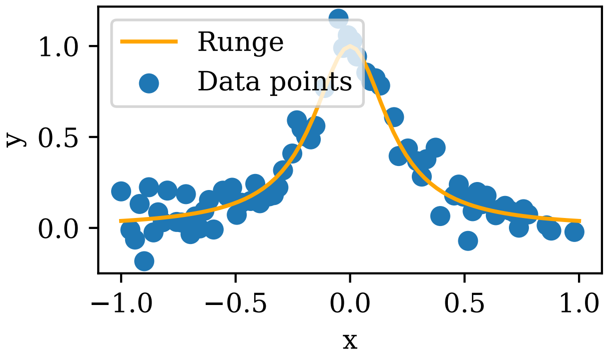

Figure 1: Train data used for the Runge function plotted with the Runge function. Normalized noise with variance 0.1 is added. With a sample of 100 points and a train-test split with ratio 0.8, 80 points remain for training.

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 [12]. 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 selveral tunable parameters that influence training performance. 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 0.001 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. This is to prevent overtraining when convergence is reached.

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 if not explicitly stated otherwise.

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 [13][14].

3. *Closed-form OLS and Ridge*

We first implement OLS and Ridge regression for the closed-form solutions. As a display of the general tendency of the Runge function to get overfitted, especially for OLS, an OLS model is used on different size samples of our data across different model complexities. We used the anaytical OLS solution in order to minimize error from other sources and tested samples of size 20 and 80 (the whole train data set). Then we compared the error of our predictions of the test data with the error of the predictions for the train data. This is to see how the size of the data set affects the bias-variance decomposition.

Following, the MSE for Ridge regression for polynomial degrees 2, 10, 14 and 18 is plotted across different lambda values to study the dependence on lambda. The reason to only use even degrees is that the Runge function is even, i.e. it is symmetric around the y-axis, so even-degree polynomials naturally respect this symmetry, while odd-degree polynomials would introduce unnecessary asymmetry.

4. *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 and R^2 was computed on the test data and the final MSE and R^2 was logged.

The gradients tested were OLS, Ridge and Lasso and the optimizers tested were ADAM, ADAgrad, no optimizer (Simple), momentum and RMSprop.

We chose two models to be 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.

5. *Validation with resampling techniques*

We implemented the resampling technique bootstrapping in order to decompose the observed MSE into bias and variance, as derived in equation 18. In addition, we validated and expanded upon the performance metrics quantified in Section 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.

6. *Programming tools*

All code relevant for this project has been written in the programming language Python (version later than 3.12). We relied primarily on Numpy [15] and Scikit-Learn [12], among others libraries. 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. Additionally, Pandas was used for creating Latex code [16] and Matplotlib [17] was used for plotting functions. GitHub was used for version control and as a cloud-based repository, ensuring collaboration and reproducibility of the code [18].

7. *Code structure*

The code implementation is organized around a gradient descent class which includes the main training loop. In this class one can select a gradient from a class containing all gradients and an optimizer from a class containing all optimizers. The gradients implemented are OLS, Ridge and Lasso while the optimizers implemented are ADAM, ADAgrad, Momentum, No optimizer (Simple), RMSprop. Additionally cross-validation is implemented separately. The overall structure is shown in Listing 1. In addition, several other functions and methods are implemented for data manipulation and visualization.

Listing 1: Code structure of the Gradient Descent implementation

```
Class GradientDescent
  def setGradient(gradient: OLS | Ridge | Lasso)
  def setOptimizer(optimizer: ADAM | ADAgrad |
                      Momentum | Simple | RMSprop)
  def train()
  def evaluate()

 def CrossValidation() (separate function)
```

8. *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. These were calculated using Scikit-Learns OLS, Ridge and Lasso classes. As shown in I the R^2 score for OLS and Ridge were identical (up to reported four decimals), while for Lasso the score differed by under 0.020. This verifies that our closed-form expressions were implemented correctly and the correctness of the gradient updates for the L1 penalty term.

Table I: Comparison of R^2 for Scikit-learn's built-in implementations of OLS, Ridge and Lasso with our analytical solutions for OLS and Ridge and our implementation of Lasso (10 features)

Regression	R^2(Scikit-learn)	R^2(Our implementation)
OLS	0.7604	0.7604
Ridge	0.7929	0.7929
Lasso	0.6427	0.6594

C. Use of AI tools

During the implementation of the code Artificial intelligence such as ChatGPT were used for error correction and debugging [19]. 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. ChatGPT has also been used for some revision in the creation of this report, see the file 'ChatGPT.pdf'.

III. RESULTS AND DISCUSSION

A. Results

The main purpose of this article is to explore different models 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 and the R^2 closest to 1 on the test data.

1. *Results from closed form OLS and Ridge*

Figure 2 and Figure 3 illustrates how a model trained using the OLS closed form solution on 20 datapoints starts to overfit as the polynomial degree increases. The figures show that the MSE for the test data increases drastically from around polynomial degree 9, while the R^2 score worsens after degree 12.

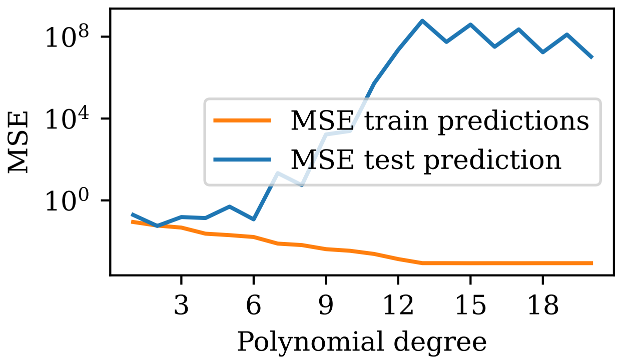

Figure 2: Train and test MSE for OLS closed form solution with 20 data points as training set.

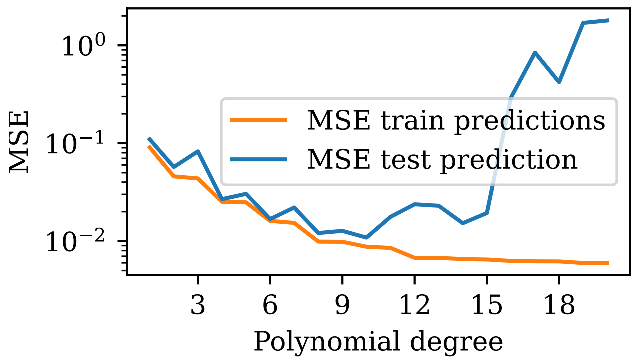

Figure 4: Train and test MSE for OLS closed form solution with 80 data points as training set.

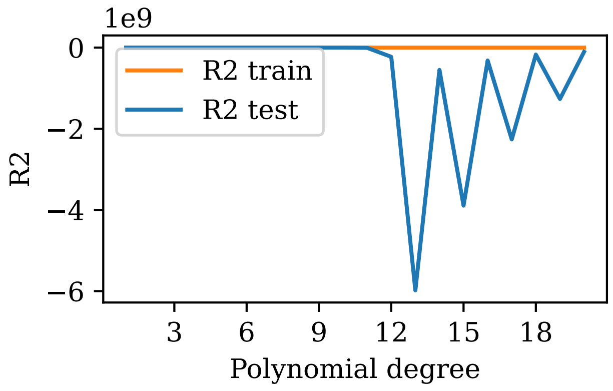

Figure 3: Train and test R^2 for OLS closed form solution with 20 data points as training set.

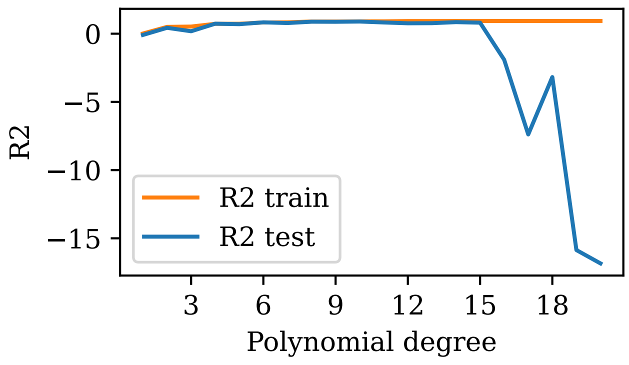

Figure 5: Train and test R^2 for OLS closed form solution with 80 data points as training set.

Similarly, Figure 4 and Figure 5 shows how a model trained using the OLS closed form solution on 80 datapoints also overfits for higher polynomial degrees. Both the MSE and R^2 scores worsen significantly from polynomial degree 15. However, comparing the results, the model trained on 80 datapoints achieves better MSE and R^2 scores than the model trained on 20 datapoints. Importantly, the models trained on 80 data points seem to allow for a higher model complexity before showing signs of overfitting.

Figure 6 and Figure 7 show the theta values for different polynomial degrees, using OLS, with 20 and 80 datapoints respectively. Colours follow a Viridis scale, where darker shades corresponds to lower-degree polynomials, and lighter (yellow) shades to higher-degree polynomials. The coefficients grow in magnitude with increasing polynomial degree, indicating potential divergence and numerical instability.

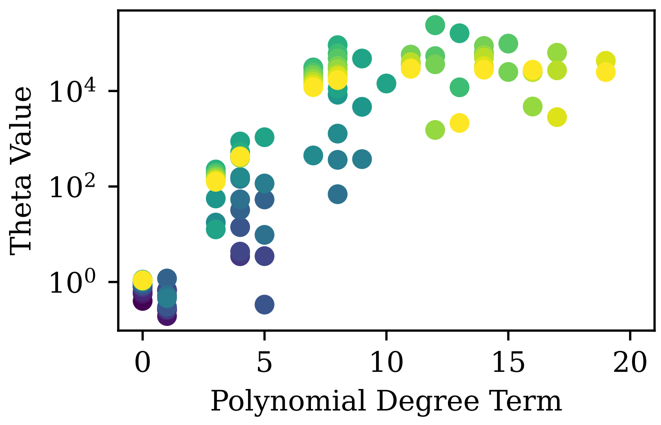

Figure 6: Coefficients of the fitted polynomial plotted for analytical OLS models. The models are trained on 20 datapoints. Models of varying complexities are trained. The points on the plot follow a Viridis colour scale, with the darker colours being the coefficients of polynomial models with lower degrees.

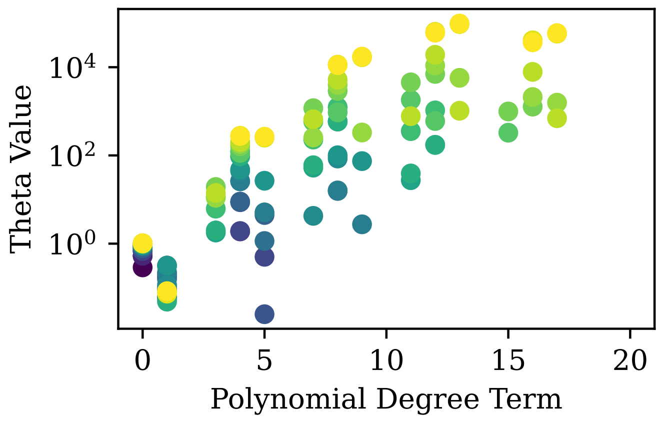

Figure 7: Coefficients of the fitted polynomial plotted for analytical OLS models. The models are trained on 80 datapoints. Models of varying complexities are trained. The points on the plot follow a Viridis colour scale, with the darker colours being the coefficients of polynomial models with lower degrees.

Figure 8 shows the variantion of the MSE for different lambda values. The best MSE is achieved for lambda values below 10^{-2}. However for polynomial degree 18, the MSE score increases rapidly for lambda under 10^{-8}, indicating overfitting.

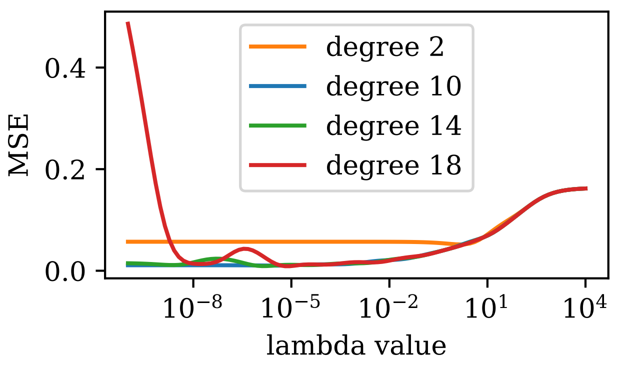

Figure 8: MSE of the closed-form Ridge regression for polynomial degrees 2, 10, 14, and 18 across different λ values. Models are trained on 80 points.

In Table II the R^2 scores obtained from the closed-form solutions of OLS and Ridge regressions models across different polynomial degrees is shown. The R^2 score for the OLS model improves until degree 14, but afterwards it is subject to severe overfitting as the negative R^2 scores for degree 16 and 18 show. On the other hand, the R^2 score for Ridge, improves until degree 6 before maintaining a relative consistent value.

Table II: R^2 score from closed form Ridge and OLS solutions. Ridge regression is performed with a lambda value of 0.01.

Poly. deg.	2	4	6	12	14	16	18
OLS	0.0432	0.7335	0.8330	0.7636	0.8486	-1.9065	-3.1804
Ridge	0.0433	0.7336	0.8370	0.8604	0.8543	0.8435	0.8314

2. *Results from gradient descent*

Table III shows R^2 scores of gradient descent models with varying complexity using the gradients of OLS and Ridge regression with different learning rates. All models trained with a learning rate of 0.01 have poor predictive power, signifying that the solutions have not converged.

Table III: R^2 scores for gradient descent implementation with different learning rates.

Poly. deg.	2	4	6	12	14
Learning rate $\eta = 0.01$					
OLS	-2.0047	-6.5051	-7.7508	-13.421	-21.462
Ridge	-4.8664	-3.1413	-9.5130	-28.306	-38.152
Learning rate $\eta = 0.1$					
OLS	0.3971	0.5534	0.4033	0.5046	0.5335
Ridge	0.4726	0.4735	0.4764	0.3022	0.5169
Learning rate $\eta = 0.5$					
OLS	0.4870	0.5534	0.5312	0.5003	0.5050
Ridge	0.4867	0.4735	0.4889	0.4173	0.4890

The main experiments were conducted on OLS, Ridge and Lasso regression with the optimizers ADAM, ADAgrad, Momentum, RMSprop and no optimizer, with 0 to 15 features. The resulting MSE and R^2 score are shown in figures 9, 10 and 11.

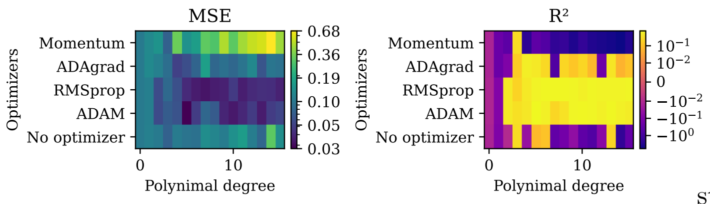

Figure 9: (OLS) Heatmap of MSE and R^2 value for different gradient descent optimizers and polynomial degrees. Yellow boxes correspond to good scores, while dark boxes correspond to weaker scores.

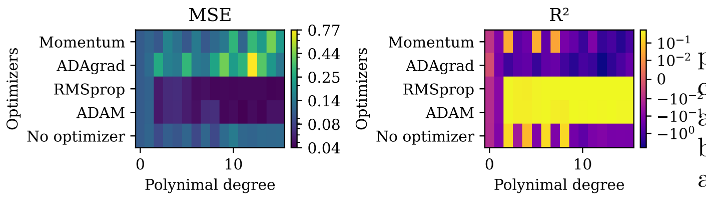

Figure 10: (Ridge) Heatmap of MSE and R^2 value for different gradient descent optimizers and polynomial degrees. Yellow boxes correspond to good scores, while dark boxes correspond to weaker scores.

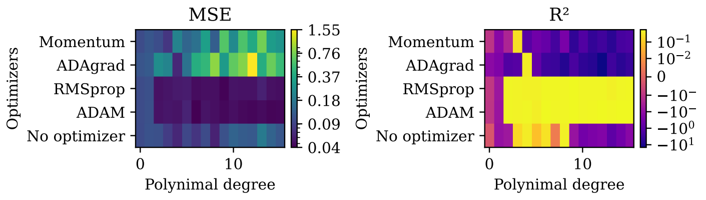

Figure 11: (Lasso) Heatmap of MSE and R^2 value for different gradient descent optimizers and polynomial degrees. Yellow boxes correspond to good scores, while dark boxes correspond to weaker scores.

We see that for figures 9, 10 and 11, RMSprop and ADAM seem to give the lowest MSE and highest R^2 score across all three methods. It also seems that for Ridge regression, we clearly see how the function is better fitted with even-degree polynomials, clear for the Momentum and no optimizer in 10. The error scores seem to get worse as we increase the polynomial degree for Momentum and ADAgrad, while becoming better for RMSprop and ADAM.

Table IV: Comparison of final MSE and R^2 for non-stochastic and stochastic gradient methods (RMSprop, polynomial degree of 6). Learning rate of 0.05 is used in all cases, and training is restricted to maximum 100 epochs.

Regression	**Non-stochastic**		**Stochastic**	
	MSE	R^2	**MSE**	R^2
OLS	0.041	0.590	0.041	0.597
Ridge	0.049	0.511	0.056	0.438
Lasso	0.058	0.428	0.057	0.436

In IV we observe that the difference between non-stochastic gardient descent and stochastic gradient descent, is rather small. The difference is largest for Ridge regression, where the MSE value for SGD is 0.007 larger than for the non-stochastic, and the R^2 score is 0.073 lower than that of the non-stochastic.

3. *Bias-variance tradeoff*

Figure 12 illustrates the bias-variance trade-off for a polynomial model fitted to the Runge function. The decomposition of MSE into bias and variance is plotted against the degree of the model. Although quite unstable, there is a clear tendency for the variance to increase and the bias to decrease as model complexity increases.

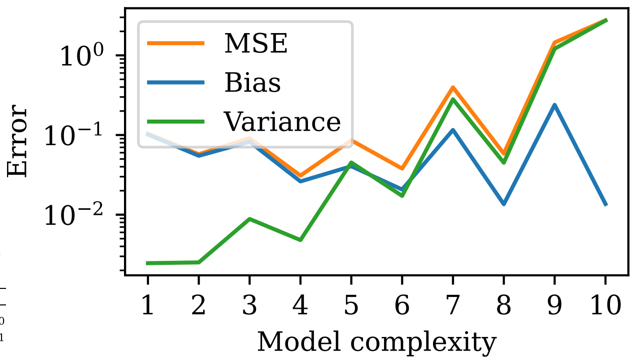

Figure 12: Bias-variance decomposition of a closed form OLS model with varying polynomial degrees. Figure is produced with 500 bootstrap samples for each model. Error is plotted logarithmically because of large numerical differences in values.

Figure 13 shows this decomposition for a fixed model complexity against varying sizes of training data. The model is a 10th degree polynomial. The figure shows that the bias remains relatively stable regardless of the number of data points it is trained on. However, the variance shows a clear tendency to decrease as the training sample increases in size.

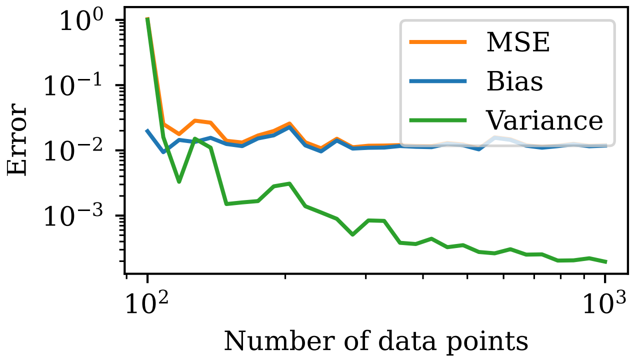

Figure 13: Bias-variance decomposition of a degree 10 polynomial trained with closed form OLS. The decomposition is plotted against the size of the dataset. The model is trained on 80% of the data, and tested on the remaining 20%. The figure is produced with 100 bootstrap samples for each data sample. The Error is plotted logarithmically because of large numerical differences in values.

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

Table V: Bias–variance trade-off for RMSprop with different gradients and 6 features

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

4. Cross validation

Figure 14 15 16 compares the mean MSE with standard deviation of the cross validation and the bootstrap resampling methods. Generally MSE decreases with number of features. The standard deviation is generally smaller for bootstrap that cross validation.

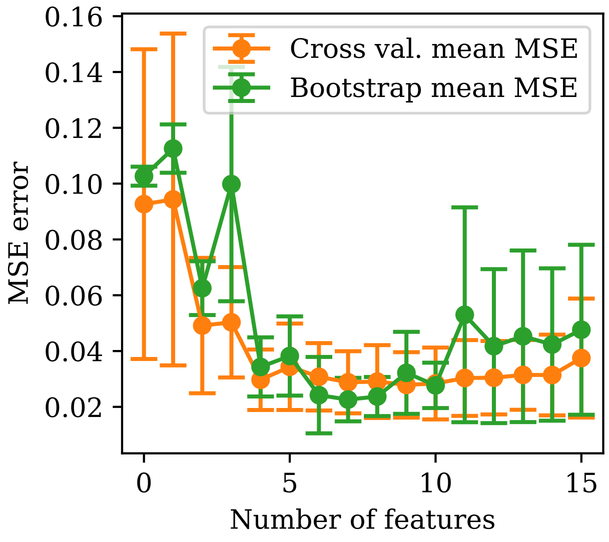

Figure 14: MSE $\pm$ standard deviation of cross validation with 10 folds and of bootstrap with 50 samples for RMSprop with decay 0.99 and gradient OLS with parameter 0.01 with polynomial degree 0 to 15. Learning rate 0.05.

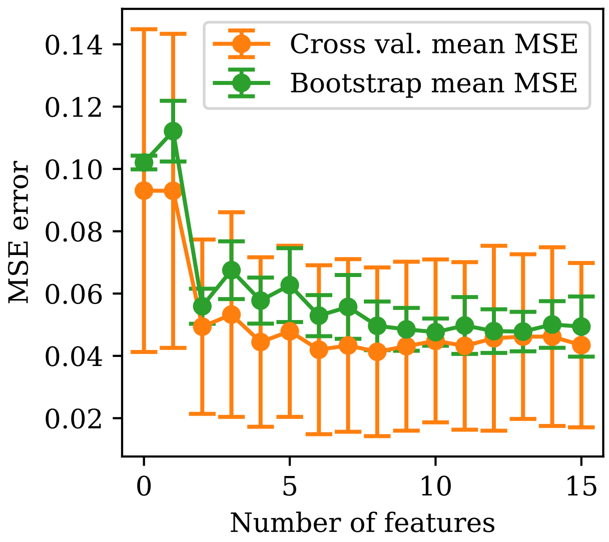

Figure 15: MSE $\pm$ standard deviation of cross validation with 10 folds and of bootstrap with 50 samples for RMSprop with decay 0.99 and gradient Ridge with parameter 0.01 with polynomial degree 0 to 15. Learning rate 0.05.

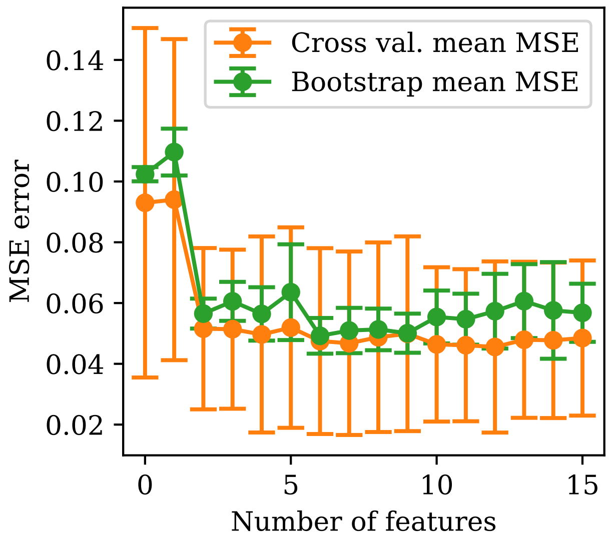

Figure 16: MSE ± standard deviation of cross validation with 10 folds and of bootstrap with 50 samples for RMSprop with decay 0.99 and gradient Lasso with parameter 0.01 with polynomial degree 0 to 15. Learning rate 0.05.

B. Discussion

1. *Overfitting and the Runge function*

Figures 2, 3, 4, and 5 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 figures 2 3 4 and 5, as well as figures 6 and 7. They show that overfitting occurs more with a model trained on fewer data points and more features. Hence, a way to reduce overfitting is by reducing the number of features or training on more data. In our case, both ways improves the MSE and R^2 scores signinficantly.

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 2, 3, 4 and 5 show that polynomial models of odd degrees perform systematically worse, also confirmed by Figure 12. 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 even degree.

2. *Comparison of Closed-Form for OLS and Ridge Regression*

The general take-away from our experimentation with OLS models on the Runge function, is that it works great for polynomials of low degrees. This can be seen in Figure 2, 3, 5, and 5. Even when increasing the number of training points, OLS will still overfit if the model is complex enough. This can be explained by the steady increase observed in the theta values, as seen in Figure 6 and 7 and discussed in the last section. The penalty term included by Ridge regression is a reasonable way to counteract this overfitting.

Figure 8 shows the effect of a varying lambda term for various polynomial models, which penalizes the largest parameters, and forces them to stay on the same scale. The effect of this extra term is unnoticable for smaller degrees, at least when lambda is smaller than 0.01. However, for polynomial models of large degrees, the effect of the penalty term is quite significant, even for very low values of lambda. A degree 18 polynomial, which drastically overfits with OLS, produces predictions comparable to degree 10 and 14 with a lambda value of 10^{-8}. This is confirmed by Table II. Here, OLS and Ridge perform comparably on degrees between 2 and 14. On degrees 16 and 18, however, OLS shows significant signs of overfitting, with R^2 values of -1.9065 and -3.1804 respectively. The R^2 values for Ridge with these degrees are comparable to the other values found with Ridge. This shows that Ridge regression is less suceptible to overfitting than OLS.

It is, however, important to choose the lambda value carefully. Figure 8 also shows that as lambda increases, so does the general performance of the model. With high enough lambda values, the muting effect of the penalty term will be so strong that all models perform equally well. In our case, a lambda value of 10 ensures that both 2nd and 18th degree polynomials predict with the same MSE.

3. *Gradient Descent Approaches with OLS and Ridge*

We now shift our focus from analytical to numerical solutions. Solutions to OLS and Ridge regression are parameters that are particularly well positioned to be solved numerically, as they have analytical solutions to be compared to. This offers better control over solution quality.

Table III shows that naive attempts at gradient descent perform significantly worse than our analytical solutions. With a learning rate of 0.01, the poor R^2 score suggests that neither gradient descent model reach a convergent solution. With higher learning rates, the solutions become better, with R^2 scores ranging between 0.4 and 0.5, whereas the analytical solutions have scores ranging between 0.7 and 0.9. Our further discussion on gradient descent will focus on how one can close this gap.

4. Comparison of Gradient Descent optimizers

For all three methods, RMSprop and ADAM generally yield the lowest MSE and highest R^2 as the polynomial degree increases as can be seen in Figures 9, 10, and 11. This suggests that these adaptive optimizers handle the complex curvature of the Runge function more effectively. This behaviour indicates a partial mitigation of overfitting, as the optimizers stabilize the training process even for higher-degree polynomials. In contrast, ADAgrad performs poorly across all models, but especially for Ridge and Lasso, likely due to its rapidly diminishing learning rate.

For Ridge and Lasso, performance remains better for even-degree polynomials, which aligns with the even symmetry of the Runge function. This pattern is especially noticeable for the no-optimizer baseline in the R^2 heatmap of Figure 10.

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.

5. Stochastic gradient descent

Table IV shows that the difference between non-stochastic, and stochastic gradient descent is generally small. Both methods converge to similar final MSE and R^2 scores, likely because the dataset is moderate in size and RMSprop that was used for the comparison stabilizes the stochastic updates. The largest discrepency appears for Ridge regression, where stochastic updates slightly increase the MSE and decrease the R^2, possibly due to the interaction between L_2 regularization and the noise introduced by the stochastic updates. Overall, these results suggests that for this problem, SGD provides comparable performance to full-batch gradient descent.

6. Bias-variance tradeoff

Another way to quantify overfitting is by analyzing a model through the lens of the bias-variance trade-off, as shown in Figures 12 and 13. Bias and variance represent two competing sources of error that together determine the MSE.

In general, a less complex model has limited capacity to represent the true underlying function. Because it cannot adapt to fine details in the data, it tends to produce averaged predictions. This systematic deviation from the true function is what we quantify as bias. Less complex models are therefore biased toward the mean, as they cannot capture more intricate relationships.

As model complexity increases, the bias typically decreases, but variance increases. Complex models are more flexible and can learn "patterns" in the noise, leading to unstable predictions that vary between datasets. Of course, these patterns do not exist, and will therefore not make predictions any better. In fact, it usually makes predictions worse. This is the essence of overfitting, and can be seen in 12, where the variance grows for higher polynomial degrees.

In contrast, Figure 13 shows that as the number of data points increases, both the MSE and bias remain relatively stable while variance decreases. With more data, the model generalizes better and becomes less sensitive to noise, effectively reducing variance without increasing bias.

7. Empirical estimation through resampling

Our results on model performance gain a new dimension when we consider the empirical standard deviation of the test MSE. This is shown in table **??**. 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 empirical 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 admitting 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 data points 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.

8. Further research

The main way to improve the model for the Runge function would be to expand the search space. One could further consider different learning rates, as briefly explored in Figure 8, and different 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 therefore 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 points.

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 Ridge shows that one can only achieve an R^2 score 0.8604 for the number of features tested, at least for OLS (Table II). Similar results probably hold for both OLS 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 study we analyzed the Runge function, using various linear regression methods, including OLS, Ridge, and Lasso, as well as gradient-descent methods. Gradient descent was tested with multiple optimizers (ADAM, ADAgrad, RMSprop, Momentum, and no optimizer), and the gradients for OLS, Ridge and Lasso were used. Stochastic gradient descent was applied to the best performing models. We also used bootstrapping and cross-validation to assess model robustness and explore the bias-variance trade-off.

Our results confirm the well known behaviour of the Runge function: increasing model complexity does not necessarily improve performance. High-degree polynomials are prone to overfitting, which is evident from rising variance and deterioating test performance, while low-degree models are biased towards the mean. The bias-variance analysis and resampling methods further reinforce these findings, showing that variance decreases with more training data, and that cross-validation provides more stable MSE estimates than bootstrapping for this dataset using Ridge regression.

Among the methods tested, Ridge consistently outperformed OLS and Lasso for the number of features considered, as it is robust to overfitting. Gradient descent with RMSprop achieved results close to the analytical solutions, but performance was still limited by fixed hyper parameters and the inherent difficulty of the Runge function. Stochastic gradient descent performed similarly to full-batch gradient descent indicating that adaptive optimizers can stabilize training even with moderate-sized datasets.

Overall, this study demonstrates the challenges of fitting a polynomial to the Runge function, and highlights the importance of regularization, optimizer choice and dataset size in controlling overfitting. While linear regression methods can achieve reasonable approximations, the fundamental limitations of polynomial interploration suggest that more flexible approaches, such as neural networks, may be needed for significantly better performance. Future work could explore broader hyperparameter searches, alternative optimization stratigies, larger or higher-quality datasets, and non-polynomial models to improve predictive accuracy.

[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] M. Hjorth-Jensen, *Lecture notes week 35: From ordinary linear regression to ridge and lasso regression* (2025), URL `https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week35.ipynb`.

[6] M. Hjorth-Jensen, *Lecture notes week 36: Linear regression and gradient descent* (2025), URL `https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week36.ipynb`.

[7] Wikipedia contributors, *Lasso (statistics)* (2025), [Online; accessed 25-September-2025], URL `https://en.wikipedia.org/wiki/Lasso_(statistics)`.

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

[9] M. Hjorth-Jensen, *Lecture notes week 37: Gradient descent methods* (2025), URL `https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week37.ipynb`.

[10] I. Castro, D-Lab UC Berkely (2021), URL

https://dlab.berkeley.edu/news/beginner%E2%80%99s-guide-bootstrap.

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

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

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

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

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

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

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

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

[19] OpenAI, *Chatgpt: Ai language model* (2025), accessed: 2025-10-06, URL https://chat.openai.com/.