Model selection for Runge's function

A Comprehensive Comparison of Regression and Optimization Methods

Students: Hugo Hilde, Natallia Danilchanka

University of Oslo

(Dated: October 5, 2025)

This paper investigates various regression methods and their performance in approximating Runge's function, $f(x) = \frac{1}{1+25x^2}$, which is known for its oscillatory behavior when fitted with high-degree polynomials. As many real-world problems exhibit erratic behaviour, a thorough investigation of Runge's phenomenon provides valuable insights for model selection in practical applications.

We implemented and compared Ordinary Least Squares (OLS), Ridge, and LASSO regression techniques using polynomial models up to degree 15. Parameters were estimated using both closed-form solutions (OLS and Ridge) and gradient descent optimization. Five gradient descent variants were evaluated: standard GD, momentum, AdaGrad, RMSprop, and ADAM, implemented with both batch and stochastic approaches.

Model performance was assessed through the bias-variance decomposition framework, with generalization ability evaluated using bootstrap resampling and k-fold cross-validation. Our results demonstrate that OLS achieves near-zero MSE for clean data at polynomial degrees above 8, while noisy data exhibits clear overfitting beyond degree 7-8, with test MSE leveling off at approximately 0.015, while adaptive learning rate methods (especially ADAM) demonstrate superior performance compared to standard gradient descent.

I. INTRODUCTION

In the fast-changing world of Machine Learning, the myriad of models available is ever expanding. These models can be broadly divided into regression (minimizing distances between estimated and true functions) and classification (assigning samples to predefined classes). A natural question that arises is what model should be used for different problems. To address this question we focus on regression and seek models that can estimate the function

$$\frac{1}{1+25x^2}$$

devised by the famous German mathematician Carl Runge. The Runge function presents a notorious challenge in function approximation due to the so-called Runge phenomenon — oscillations that occur when using high-degree polynomial interpolation. This makes it an ideal test case for comparing regression methods and understanding the bias-variance tradeoff, as it exposes the strengths and limitations of different approaches to model complexity.

Given the overwhelming amount of possible models to choose from, we return to the roots of the ML field, historically dominated by linear regression. This family of methods is characterized by its linear nature: we make a (rather) strong assumption about the underlying relationship in the data, namely that the target value can be written as a linear combination of the variables. The task at hand is then to find the coefficients θ that minimizes the distance between the true output value y and our estimated output $\hat{y}$.

While modern and fancier methods have largely come to dominate the field of machine learning, the tried-and-true linear regression remains compelling for its interpretability, low computational cost, and solid foundation for extensions such as polynomial feature representations.

We deploy the Ordinary Least Squares estimate of the optimal parameters θ, as well as two famous regularization methods called Ridge and Lasso. In the modern age of ML, where billions of parameters are estimated iteratively, the closed-form solution for parameter estimation found by OLS and Ridge is quite a sight for sore eyes. We also show how to proceed when a closed-form solution is not obtainable, implementing Lasso regression using the iterative method of gradient descent.

Furthermore, we derive the bias-variance decomposition, a fundamental result in statistics, showing beautifully the innate tension between a model's overfitting (closeness to the training points) and generalizability (how the model performs on unseen data).

Lastly, we address the situation where data is scarce (i.e. most real-world situations). This introduces the technique of resampling, here exemplified by two powerful methods: Bootstrapping and Cross-Validation.

This paper is organized as follows: Section 2.A presents the theoretical framework for OLS, Ridge, and Lasso regression, gradient descent optimization. Section 2.B describes our implementation and experimental setup. Section 3 presents results comparing the different methods on the Runge function, as well as a demonstration of the bias-variance decomposition. In section 4 we discuss the findings and their implications.

II. METHODS

A. Method 1/X

First off, the question of whether to include stochastic noise to the Runge function is a pressing one. This can have a significant effect on our data experiments and analysis. Therefore, in the preliminary parts of the code, we fit the polynomials both on the noisy data and its "clean" counterpart. We will return to what we found and how we proceeded in the results section. The first part of our analysis is fitting polynomials with methods that use analytical solutions, such as Ordinary Least Squares (OLS) and Ridge re-

gression. Then, we move on to approximating with a set of gradient and Lasso methods to find an appropriate model to fit our data. In total, we implement five different optimization rules for gradient descent:

- Regular gradient descent, which uses constant learning rate
- Momentum-based gradient descent, an update rule which balances previous gradients with the current gradient
- AdaGrad, an adaptive learning rate method that scales the learning rate inversely with the square root of the sum of all past squared gradients
- RMSProp, a method that addresses AdaGrad's diminishing learning rates by using an exponentially decaying average of past squared gradients
- ADAM, an update rule which balances the properties of momentum and adaptive learning rates (combining momentum with RMSProp)

A natural question that arises when estimating the model parameters θ for OLS and Ridge is why we should bother to use gradient descent, since in these cases we have the closed-form solution for the optimal parameters. However, these methods are useful when the number of samples (n) or the number of covariates (p) grows large. Since we would need to calculate the inverse of the matrix $(X^TX)^{-1}$, an operation demanding $O(p^3)$ flops, we see that this route is not always feasable.

Further, study the Bias-Variance tradeoff as s function of model complexity. This is a question we face every time we want to find optimal parameters for the model, but need to choose to balance between giving the model more variability and lower bias, or opposite.

Additionally, we explore resampling techniques such as Bootstrap and k-fold cross-validation for cases where we have a limited number of data points.

B. Implementation

The first order of business in any Machine Learning process is to familiarize oneself with the data. In our case, the data is generated from Runge's function (with noise):

$$\frac{1}{1+25x^2}+\epsilon$$

, where $\epsilon \sim (0, 0.1)$ Our features are x-values sampled from the uniform distribution on the interval $[-1, 1]$. This domain of x presents the ML-practitioner with a dilemma; usually scaling and standardizing the features are mandatory. However, since $|x| \leq 1$, the polynomial features that we use to fit the models will never grow out of bounds.

We find the closed-form solutions for estimation of θ for OLS and Ridge:

For **Ridge Regression**, we minimize:

$$L(\theta) = ||y - X\theta||^2 + \lambda||\theta||^2$$

Expanding and taking the derivative with respect to θ:

$$\frac{\partial L}{\partial \theta} = -2X^T y + 2X^T X\theta + 2\lambda\theta = 0$$

Rearranging:

$$(X^T X + \lambda I)\theta = X^T y$$

Therefore:

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

where $\lambda = 0 \implies$ **OLS** (since $(X^T X)^{-1} X^T y$ is the standard OLS solution).

We leverage these formulas in the first part dealing with analytically solvable setups.

If we replace the penalty term of Ridge regression, $\lambda||\theta||^2$ (L2-loss) with $\lambda||\theta||$, (L1-loss), we no longer have a closed-form solution, since the derivative of our cost function is discontinuous at 0. Thus, we implement iterative methods in order to estimate the optimal parameters θ.

This is where the gradient descent methods described earlier comes in handy, and we implement all 5 for both Lasso and the regular Ridge/OLS regression. Lastly we utilize a technique called Stochastic Gradient Descent (SGD), or more correctly in our case called Mini-batch Gradient Descent. In this algorithm, we decrease computational cost by calculating gradients in batches of a given size. The calculation of the full gradient can become very compute intensive with a large number of samples and/or features. This is why Mini-batch Gradient Descent chooses m ¡ n samples at random and computes the gradient for these m samples. This gradient might be less than optimal, though for a correctly chosen m, it might converge quickly and be significantly faster than full-batch gradient descent. By using these mini-batches, we can also balance **Explorations** and **Exploitation**: We add a stochasticity to the calculation of the gradients, leading to a randomness in how we move in the search landscape, preserving a degree of exploration. At the same time, by having a not too small batch, our gradient will likely be close to that of the full batch, and we exploit the information from the gradient.

The difference between SGD and Mini-batch Gradient descent is as follows:

- Stochastic Gradient Descent (SGD): Uses 1 sample per update (batch size = 1)
- Mini-batch Gradient Descent: Uses a small batch per update ($1 < m < n$)

C. Use of AI tools

The AI is a very convenient and widely used tool in the life of the modern researcher. It helps to speed up code development and debugging, as well as search for the root cause of errors without deep diving into not all the time well well-maintained documentation on various Python li-

braries.

As can be seen in this example: [1], it is enough to pass the code and the error that one gets during compilation to get suggestions to resolve issues. On average, it can require a few iterations and a basic understanding of what one needs to achieve to get the issue fixed, and not create another. Because chat ChatGPT is a probabilistic model [2] that tries to predict the most likely output based on input data. No one can blindly believe that it will provide a correct answer.

Additionally, it was used to create and adjust a plot and a combination of plots that were used for analysis. It very conveniently adjusts existing plots by adding grids, titles, or helps to space the legend to avoid interference with visualized data.

III. RESULTS AND DISCUSSION

We implemented the models incrementally, expanding the implementation of OLS, Ridge and Lasso with various gradient descent methods. First, We present the results from the initial implementation, before presenting the full comparison of all models across multiple hyperparameter configurations.

In the first step of the project, we generated x-values drawn from the uniform distribution on the interval $[-1, 1]$. We explored them with and without scaling (as shown in the figure: 1)

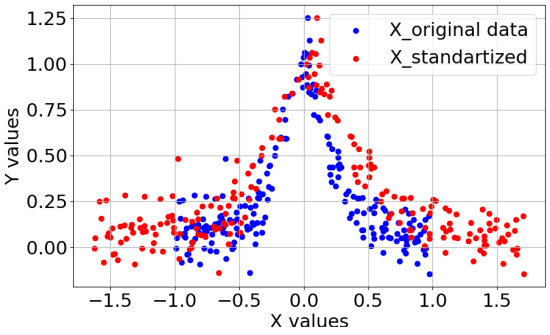

Figure 1: Sample data

The impact of scaling in this case is minimal, since the data is naturally centered around 0. We did scale the original x vector, but in reality, this did not impact our models. In the next step, we have split the data into a train and a test set with a 4/5 to 1/5 proportion, respectively.

The first method we applied to the dataset was the Ordinary Least Squares(OLS) algorithm, where we fit polynomials from degree 1 to 15. From this we calculated the Mean Square Error (MSE) (fig.: 2) and R^2 (fig.:3)for the train and test data respectively:

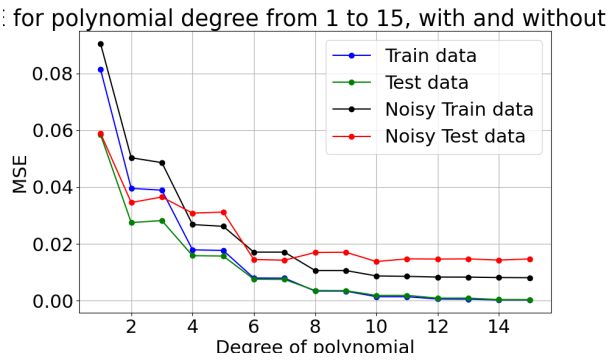

Figure 2: OLS MSE

For MSE and lower-degree polynomials, we observed that, depending on the seed and the

number of variables, the test set can perform better than the training set. That can be explained by the synthetic nature of data, and that the noise is small to distort the relationship between dependent and independent variables.

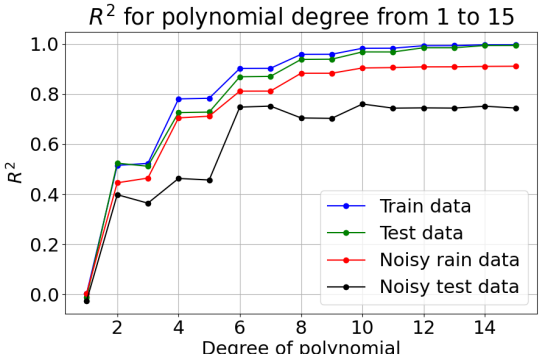

Figure 3: OLS R^2

An important observation from the plots (fig.: 2) and (fig.: 3) is how the added noise in the original function influences the results. Even though the MSE stabilizes at very low levels even for the noisy test data, the R^2-score revealed something else: the added noise introduces variance that is not fully explained by our linear fits. This observation was key for understanding the strengths and limitations of our models.

Another interesting manifestation of the Runge's phenomenon is shown in the plot of how the coefficients θ change as the degree of polynomials increases. This picture, 4, highlights that the solution became unstable as the degree of polynomials grew large, indicating clear overfitting.

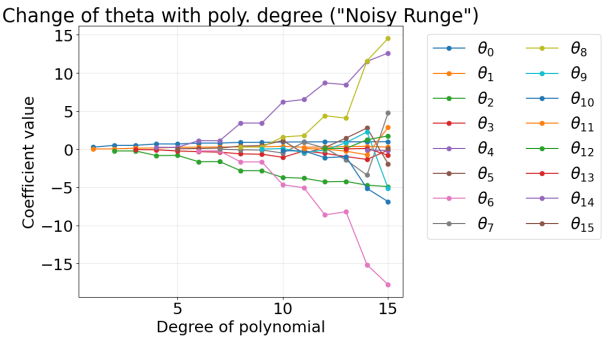

Figure 4: theta OLS

The next method we considered was Ridge regression method. We set up a grid search not only for polynomials of different degrees(the same as in OLS from 1 - 15) but also different regularization parameters to see how penalizing theta can affect our prediction accuracy.

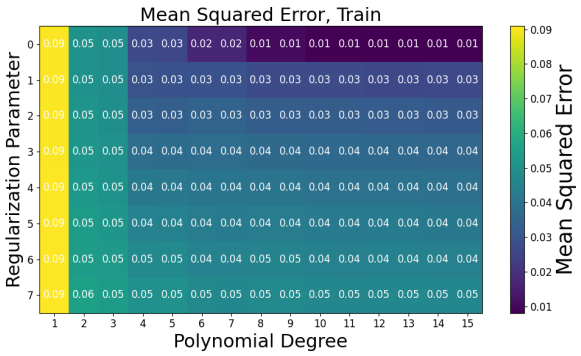

Figure 5: MSE Ridge Train data

Interestingly, as shown in both the plots for predicting on the train set (fig.:5) and test set (fig.:5), that the lowest MSE was found when the regularization parameter $\lambda = 0$, transforming the Ridge regression method into the previous OLS model. It could be due to the fact that our parameters are relatively small in OLS, and lowering them does not improve the predictive

capabilities of the model.

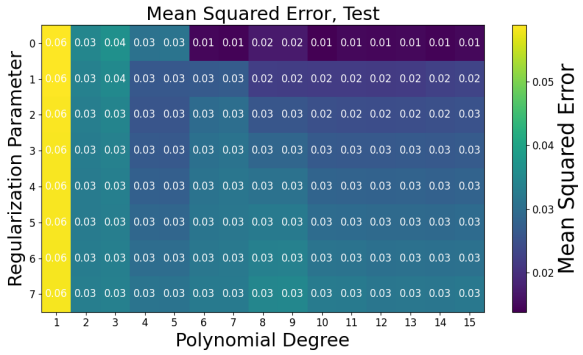

Figure 6: MSE Ridge Test data

A. Iterative methods

In the remaining part of our paper, we left the realm of closed-form solutions and implemented iterative methods. First, we implemented the standard gradient descent methods for estimating θ, both for OLS and Ridge. In this method, the learning rate is constant, and in the case when we set the learning rate to the reciprocal of the maximum eigenvalue of the Hessian matrix ($\frac{2}{n}((X^TX) + 2\lambda I$, the amount of iterations needed for proper convergence for degree 5 averaged around 9000. (maybe mention that we manually double checked that the thetas found by gradient descent is the same as for the closed-form solution)

The learning rate greatly influenced the convergence, if we set it manually, for the same degree as above, the differences between learning rates of 0.1, 0.01 and 0.001 are dramatic; leading to overflow, convergence after 20 000 iterations and convergence after 200 000 iterations respectively.

Therefore we implemented adaptive update rules, methods that leverage properties of the search landscape to dynamically adjust the learning rate.

We implemented all 4 dynamical methods discussed in previous sections, which yielded the following results:

First, we performed a search over different hyperparameters (degree of polynomials, λ for regularization and initial step sizes) on all 5 gradient descent methods. Again we saw that $\lambda = 0$ was the best value for minimizing the MSE, collapsing the Ridge model into Ordinary Least Squares. In figure 7, we can see how the ADAM algorithm dominated our results.

Top 10 Ridge regression configurations:

optimizer	degree	lambda	learning_rate	test_mse
ADAM	6	0.0	0.01	0.014617
ADAM	6	0.0	0.05	0.014744
ADAM	6	0.0	0.10	0.014794
ADAM	6	0.0	0.50	0.014843
ADAM	7	0.0	0.01	0.014972
ADAM	7	0.0	0.05	0.015042
ADAM	7	0.0	0.10	0.015086
ADAM	7	0.0	0.50	0.015121
ADAM	8	0.0	0.01	0.017367
ADAM	8	0.0	0.05	0.017399

Figure 7: Gradient Descent for OLS and Ridge

This highlights the ability of this algorithm to escape local minima. In our code, we printed convergence statement for all methods, which revealed that the other methods converged/got stuck in sub-optimal solutions.

Then we implemented a new regression method, the Lasso. As mentioned earlier, this

method has no closed-form solution, so we leveraged the GD methods implemented earlier. The same grid search yielded the results in figure 8.

Top 10 Lasso regression configurations:

optimizer	degree	lambda	learning_rate	test_mse
lasso_adagrad	11	0.1	0.500	0.029801
lasso_ADAM	3	0.1	0.050	0.038580
GD_lasso	3	0.1	0.100	0.038683
GD_lasso	2	0.1	0.100	0.038776
lasso_GD_with_momentum	3	0.1	0.050	0.038937
lasso_RMSprop	2	0.1	0.001	0.038961
lasso_RMSprop	3	0.1	0.001	0.038985
lasso_adagrad	3	0.1	0.050	0.038993
lasso_adagrad	2	0.1	0.050	0.038994
lasso_GD_with_momentum	2	0.1	0.010	0.039000

Figure 8: Gradient Descent for Lasso Regression

Interestingly, here the choice of optimizer method was less crucial. We had an outlier on top (AdaGrad with polynomial degree 11), but the clear overall trend was that lower degrees of freedom and a small λ produced the best solutions. There might be many reasons for why AdaGrad performed so well with exactly those configuration, but since it was a lone outlier, we deemed it statistically non-significant and not as a valid endorsement of this optimizer configuration.

Lastly we implemented all previous methods with an optional use of Stochastic Gradient Descent (SGD). We ran a comprehensive grid search over all hyperparameters, including mini-batches of size 16, 32 and full batch. The table 9 summarizes all possible model configurations, combining all 5 update rules, different gradient for OLS/Ridge and Lasso along with the other hyperparameters.

TOP 10 OVERALL CONFIGURATIONS

model	degree	lambda	learning_rate	batch_size	test_mse
ADAM_Ridge	7	0.0	0.10	full	0.014318
ADAM_Ridge	6	0.0	0.10	full	0.014487
Momentum_Ridge	6	0.0	0.01	32	0.014516
ADAM_Ridge	6	0.0	0.01	32	0.014707
ADAM_Ridge	7	0.0	0.10	32	0.014730
ADAM_Ridge	7	0.0	0.01	32	0.015216
RMSprop_Ridge	6	0.0	0.01	16	0.015463
ADAM_Ridge	7	0.0	0.01	16	0.015467
RMSprop_Ridge	7	0.0	0.01	16	0.015637
ADAM_Ridge	7	0.0	0.01	full	0.016206

Figure 9: All possible configurations, including SGD

We found that the optimal degree of our Polynomial Features was 6 or 7, striking an optimal balance between bias and variance. Again, ADAM dominated, proving its capabilities on our dataset. For our relatively simple dataset, regularization never helped the models improve their generalizability, clearly shown by the lambda column in figure 9, indicating that all features are needed for best predictions.

B. Bias variance

To effectively minimize errors, it's crucial to understand what makes up those errors. Let's take a closer look at error decomposition to break down the different parts of it. By doing this, we can identify which elements we can change and control and which ones we simply can't alter.

If we take the expectation of the squared error:

$$\mathbb{E}\big[(\mathbf{y} - \tilde{\mathbf{y}})^2\big]$$

Substituting $\mathbf{y} = f(x) + \epsilon$ and $\tilde{y}$ with $\hat{f}(x)$:

$$= \mathbb{E}\big[\big(f(x) + \epsilon - \hat{f}(x)\big)^2\big]$$

Expanding the square and using that the expectation is a linear operator:

$$\mathbb{E}\big[(\mathbf{y}-\tilde{\mathbf{y}})^2\big] = \mathbb{E}\big[(f(x)-\hat{f}(x))^2\big] + 2\mathbb{E}\big[(f(x)-\hat{f}(x))\epsilon\big] + \mathbb{E}[\epsilon^2] \quad (1)$$

If we assume that $\mathbb{E}[\epsilon] = 0$, the second term vanishes, since we can factor out the $\mathbb{E}[\epsilon]$ in term two (due to the linearity of expectations). Further, we have that $\mathbb{E}[\epsilon^2] = \sigma^2$ by the definition of the variance.

Further, let's take a look at the first term in (1):

$$\mathbb{E}\big[(f(x)-\hat{f}(x))^2\big]$$

Subtracting and adding $\mathbb{E}[\hat{f}(x)]$:

$$= \mathbb{E}\big[\big(f(x)-\mathbb{E}[\hat{f}(x)] + \mathbb{E}[\hat{f}(x)]-\hat{f}(x)\big)^2\big]$$

Expanding the square and using that the expectation is a linear operator yields 3 terms:

$$\mathbb{E}\big[(f(x)-\mathbb{E}[\hat{f}(x)])^2\big] \quad (2)$$

$$2\mathbb{E}\big[(f(x)-\mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)]-\hat{f}(x))\big] \quad (3)$$

$$\mathbb{E}\big[(\mathbb{E}[\hat{f}(x)]-\hat{f}(x))^2\big] \quad (4)$$

In (3), we see that term 2 can be written as (since $\mathbb{E}[X]$ = constant and $\mathbb{E}$[constant] = constant):

$$\mathbb{E}\big[\mathbb{E}[\hat{f}(x)]-\hat{f}(x)\big] = \mathbb{E}[\hat{f}(x)]-\mathbb{E}[\hat{f}(x)] = 0$$

which causes (3) to equal 0.

Hence, equation (1) now takes on the form:

$$\begin{aligned}\mathbb{E}\big[(\mathbf{y}-\tilde{\mathbf{y}})^2\big] &= \mathbb{E}\big[(f(x)-\mathbb{E}[\hat{f}(x)])^2\big] \\ &\quad + \mathbb{E}\big[(\hat{f}(x)-\mathbb{E}[\hat{f}(x)])^2\big] + \sigma^2 \\ &= \underbrace{\mathbb{E}\big[(f(x)-\mathbb{E}[\hat{f}(x)])^2\big]}_{\text{Bias}^2} + \underbrace{\mathbb{E}\big[(\hat{f}(x)-\mathbb{E}[\hat{f}(x)])^2\big]}_{\text{Var}} \\ &\quad + \underbrace{\sigma^2}_{\text{Var}[\epsilon]} \quad (1)\end{aligned}$$

Switching back to the original notation:

$$\mathbb{E}\big[(\mathbf{y}-\tilde{\mathbf{y}})^2\big] = \mathbb{E}\big[(f(\mathbf{x})-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \mathbb{E}\big[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \sigma^2 \quad (2)$$

If we now approximate the unknown function f with the target value y, we get the final expression:

$$\begin{aligned}\mathbb{E}\big[(\mathbf{y}-\tilde{\mathbf{y}})^2\big] &= \underbrace{\mathbb{E}\big[(\mathbf{y}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big]}_{\text{Bias}^2} \\ &\quad + \underbrace{\mathbb{E}\big[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big]}_{\text{Var}} + \underbrace{\sigma^2}_{\text{Var}[\epsilon]} \\ &= \text{Bias}^2[\tilde{\mathbf{y}}] + \text{Var}(\tilde{\mathbf{y}}) + \sigma^2 \quad (3)\end{aligned}$$

In this process, we find that our error can be broken down into three main components: the bias and variance of the function ($\tilde{\mathbf{y}}$), as well as the variance associated with error or noise. While we have little control over the noise, we can influence the bias and variance by adjusting the function ($\tilde{\mathbf{y}}$) and fine-tuning its parameters. It can help to achieve an optimal balance between bias and variance, which will help us minimize the mean squared error (MSE).

Next, we explore this knowledge on our so far best performing model Adam for 2 variables:

degree of freedom and number of sample data, to see the connection and how MSE, bias, and variance change with a change in the parameters. We are using the bootstrap technique to perform the analysis.

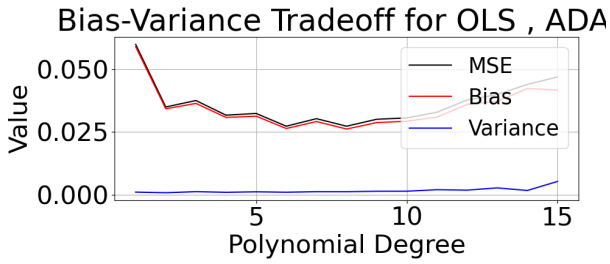

Figure 10: As a function of polynomial degree

The search for optimal polynomial degree was executed for n = 200 samples. As can be seen in figure 10, we have best MSE for polynomials from 6 to 8, where after we receive an increase in both Bias and Variance that can be a result of overfitting the data to the training sample.

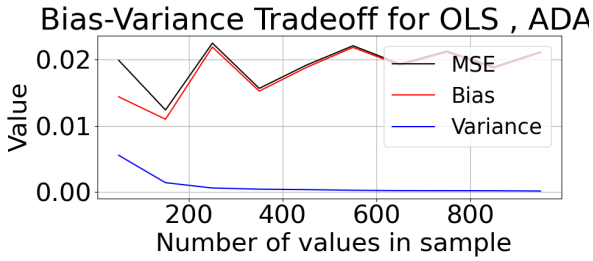

Figure 11: As a function of sample size

When exploring the Bias-Variance tradeoff as a function of the number of data points, we choose to have a degree of polynomial equal to 7 as one of the optimal degrees from our analysis above. As can be seen in figure 11, the variance drops and stays low when we reach 250 points, where bias fluctuates around 0.02 with decreasing variance when the number of points increases.

To compare the performance of the bootstrap method, we test the k-fold cross-validation method and explore three models: OLS, Ridge, and Lasso, using the sklearn library with 5 folds and lambda parameters of 0.01 for Ridge and Lasso.

In the figure 12, they are denoted by postfix _cv and the performance of the Adam algorithm is tested by bootstrap.

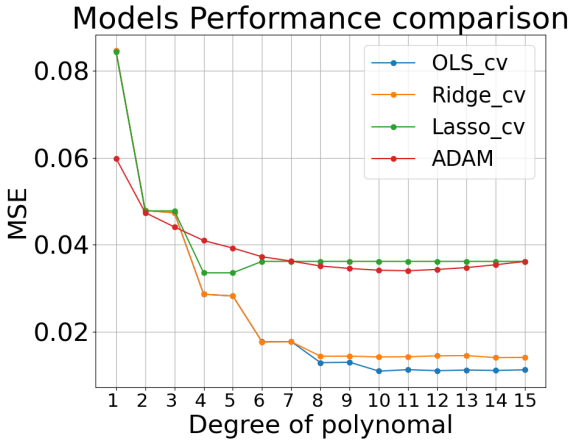

Figure 12: Performance comparison

It can be seen that the performance of the models is quite similar and comparable. Adam and Lasso, performing with a higher MSE. This can be caused by using fixed learning rates and lambdas that are sub-optimal.

```
MSE:
   n-degree      OLS_sk      Ridge_sk      Lasso_sk       ADAM
-------------------------------------------------------------------
['    1     ' '0.08459949' '0.08459946' '0.08440058' '0.05986980']
['    2     ' '0.04787296' '0.04787290' '0.04778525' '0.04738711']
['    3     ' '0.04734407' '0.04734344' '0.04779052' '0.04408232']
['    4     ' '0.02863428' '0.02863299' '0.03354596' '0.04097090']
['    5     ' '0.02822162' '0.02821924' '0.03355053' '0.03924068']
['    6     ' '0.01768704' '0.01767208' '0.03618559' '0.03724079']
['    7     ' '0.01774420' '0.01770792' '0.03618559' '0.03624136']
['    8     ' '0.01289905' '0.01436881' '0.03618551' '0.03511627']
['    9     ' '0.01298990' '0.01440038' '0.03618551' '0.03454867']
['    10    ' '0.01095228' '0.01420723' '0.03618566' '0.03414799']
['    11    ' '0.01128860' '0.01426223' '0.03618566' '0.03402679']
['    12    ' '0.01104757' '0.01446738' '0.03618549' '0.03433049']
['    13    ' '0.01120986' '0.01452438' '0.03618549' '0.03475514']
['    14    ' '0.01110730' '0.01405902' '0.03618549' '0.03540785']
['    15    ' '0.01125377' '0.01410719' '0.03618548' '0.03617098']
```

Figure 13: Performance of the models

It is interesting to see that with the Bootstrap method, we have more than double the increase in MSE compared to the single run we explored previously, which can be seen in the Figure 7

With more elaborate parameter selection, we possibly can achieve a lower error rate for the sample test. Meanwhile, there is always a risk of over-training the data.

IV. CONCLUSION

- State your main findings and interpretations
- Try to discuss the pros and cons of the methods and possible improvements
- State limitations of the study

  In our research, we systematically investigated a diverse range of models along with various tuning parameters in an effort to identify the optimal-performing model. Although practical constraints prevent us from exploring all potential combinations of models and their corresponding tuning parameters, we have infinitely many values for each parameter.

  The second point that can be improved in our research is the estimation of the Bootstrap error. Despite the fact that the Bootstrap method tends to give a higher MSE than we have for the single train-test run, it still potentially overestimates the accuracy. From the book [3, p. 251] we get to know that direct estimation of MSE for the bootstrap method can be too optimistic, as the probability of having the same data in the test and train set due to method restriction is about 0.632, as can be seen in calculation (5).

$$\Pr\{\text{observation } i \in \text{ bootstrap sample } b\} = 1 - \left(1 - \frac{1}{N}\right)^N \approx 1 - e^{-1} = 0.632. \quad (5)$$

  In our study, we do not take this into account, and thus, our ability to predict correct output can be overestimated.

- Try as far as possible to present perspectives for future work

  In the next iteration of the project, we propose to investigate k-fold cross-validation on the gradient methods implemented in the initial phase. This approach will require the development of custom cross-validation methods, allowing us greater flexibility and indepen-

dence from the constraints imposed by the scikit-learn library.

During the course of the project, we extensively utilized the code derived from the following study notes: [4], [5],[6],[7],[8], and [9].

---

[1] Natallia Danilchanka, "Example of AI use," Sept. 2025. accessed Sep. 23, 2025.

[2] Ivan Belcic, Cole Stryker, "What is GPT (generative pretrained transformer)?," Sept. 2025. accessed Sep. 23, 2025.

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

[4] M. Hjorth-Jensen, "Week 34: Introduction to the course, logistics and practicalities." `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week34.html`, accessed Sun. 5 Oct, 2025.

[5] M. Hjorth-Jensen, "Week 35: From ordinary linear regression to ridge and lasso regression," n.d. `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week35.html`, accessed Sun. 5 Oct, 2025.

[6] M. Hjorth-Jensen, "Week 36: Linear regression and gradient descent." `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week36.html`, accessed Sun. 5 Oct, 2025.

[7] M. Hjorth-Jensen, "Week 37: Gradient descent methods." `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week37.html`, accessed Sun. 5 Oct, 2025.

[8] M. Hjorth-Jensen, "Week 38: Statistical analysis, bias-variance tradeoff and resampling methods." `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week38.html`, accessed Sun. 5 Oct, 2025.

[9] M. Hjorth-Jensen, "Week 39: Resampling methods and logistic regression." `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week39.html`, accessed Sun. 5 Oct, 2025.