Regression models
FYS-STK3155 - Project 1

Herman Nissen-Sollie
University of Oslo
(Dated: October 7, 2025)

Regression models form a cornerstone of statistical learning, providing tools for modeling relationships between variables and predicting unseen outcomes. In this project, we systematically compare three fundamental regression methods—Ordinary Least Squares (OLS), Ridge, and LASSO—using synthetic data generated from the Runge function. The study explores both closed-form and gradient-based implementations, including Gradient Descent (GD), Stochastic Gradient Descent (SGD), and adaptive optimizers such as Adam and RMSProp. Model performance is evaluated under varying noise levels, dataset sizes, and polynomial complexities, with additional analyses based on resampling techniques (bootstrap and cross-validation).

The results confirm the theoretical bias–variance trade-off: increasing model complexity first improves, then degrades generalization. Ridge regression reduces overfitting through regularization, while LASSO achieves sparsity and slightly improved robustness. Gradient-based training attains comparable accuracy to closed-form solutions when the learning rate is well tuned, and Adam emerges as the most stable optimizer. Resampling methods yield consistent model selection and illustrate the balance between bias and variance. Overall, the findings bridge theoretical understanding and practical implementation of regression, offering clear guidance for model choice and optimization in noisy settings.Code to reproduce all results is available at: `https://github.com/pushing-py/Project_1`.

I. INTRODUCTION

Machine Learning (ML) is a field concerned with developing algorithms that can automatically learn patterns from data [1]. Unlike traditional programming, where explicit rules are coded, ML algorithms improve their performance by identifying structures in datasets. patterns that are often too complex for humans to detect directly. This ability is particularly valuable given the vast amounts of data that can now be processed efficiently by modern computational methods.

Regression analysis is one of the fundamental approaches in ML, with widespread applications in fields such as medicine and economics[2]. The primary objective of regression models, as well as many other ML algorithms, is to learn from existing data and predict outcomes for unseen cases. More specifically, regression aims to capture the relationship between a set of independent variables and a dependent variable, allowing new predictions to be made when new inputs are provided. However, this approach comes with challenges, including how to handle noise in the data and how to manage the computational cost of the algorithms when applied to large datasets. Choosing and applying a model is therefore not a "one-size-fits-all" task, but rather a matter of understanding the nature of the data and balancing different compromises to obtain satisfactory results.'

In this report, we focus on three regression models: Ordinary Least Squares (OLS), Ridge regression, and LASSO regression. These models are applied to data generated from the Runge function, a classical test case in numerical analysis due to its sensitivity to polynomial approximation[3]. We compare the performance of the different regression methods, analyze how hyperparameters affect their predictions, and explore optimization techniques such as gradient descent and resampling (bootstrap and cross-validation). We also investigate how different levels of noise influence the models' behavior and performance.

The goal of this study is to develop a deeper understanding of regression models, connect theoretical insights to practical results, and critically assess the methods and choices made throughout the analysis.

Section II will cover the three regression models, and introduce machine learning methods relevant for the regression task. We also cover the dataset, how we have implemented the model. There is also an AI declaration. In Section III we will go through the results of our analysis, and discuss the results critically, and discuss our theoretical understanding can be linked to our results. In the Section IV we will try to share our main findings, and point towards future work.

II. METHOD

A. Regression models in machine learning

Regression models aim to describe the relationship between a response variable Y and one or more predictors X [2]. In the simplest case, we assume a linear relationship between two variables, as expressed in Function 1, where x denotes the input and e the error term:

$$Y = \beta_0 + \beta_1 x + e \tag{1}$$

To fit such a model, the dataset is typically divided into training and test sets. The model is fitted on the training data, and its predictive ability is evaluated on

unseen test data. Normally you would use about 15-30 percent of the dataset for testing.

Polynomial regression is the task of fitting a polynomial function to a dataset. The higher the polynomial, the better it can weigh in each point of the dataset. The problem with polynomial regression comes in when there is noise in the data, resulting in that when you divide a dataset into training and testdata, a model that fits the training data perfectly won't do so with the testdata. Therefore you do need a function which is generalized to such a degree that it will fit well to the unseen data aswell. Higher polynomials will allways fit the trainingdata better, but this is not the case for the testdata.

This phenomenon is described as *the bias-variance tradeoff*. Bias means error between the model prediction, and the actual value. High-bias models are not able to find a good pattern in the dataset. Low bias can come with the cost of variance, meaning that the model is sensitive to new data. Variance is measured by calculating the difference in accuracy on the training data and testdata. High variance means the model performs much better on the training data than the testdata. The assummtion made is that the error of a Function 2

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

can be expressed by the mean squared error which is the sum of the bias squared, variance and the noise

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \text{Bias}^2[\tilde{y}] + \text{var}[\tilde{y}] + \sigma^2 \tag{3}$$

The numerical solution for finding the bias and variance is explained further in the Appendix A.

1. Model Evaluation Metrics

To evaluate the performance of regression models, two commonly used metrics are the Mean Squared Error (MSE) and the coefficient of determination (R^2). The MSE measures the average squared difference between the observed values y_i and the model predictions $\hat{y}_i$, and is defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

A lower MSE indicates that the model's predictions are closer to the actual values. The R^2 score, on the other hand, quantifies the proportion of variance in the dependent variable that is explained by the model, and is given by

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where $\bar{y}$ is the mean of the observed values. An R^2 value close to 1 indicates a strong explanatory power, while a value near 0 suggests that the model performs no better than a simple mean prediction.

2. Ordinary Least Squares (OLS)

Ordinary Least Squares (OLS) represents the simplest regression approach. The method estimates the parameters β by minimizing the sum of squared errors between observed values Y and model predictions $f(X)$:

$$RSS(\beta) = \sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{4}$$

$$f(x_i) = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \tag{5}$$

The optimal coefficients are found by minimizing the residual sum of squares. Setting the derivative equal to zero leads to the so-called normal equations:

$$X^T X\hat{\beta} = X^T y \tag{6}$$

If $X^T X$ is invertible, this leads to the closed-form OLS solution:

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{7}$$

OLS models may, however, suffer from high variance, meaning the model fits the training data well but performs poorly on test data. This problem can be alleviated by applying shrinkage methods, which deliberately reduce the size of the coefficients. This sacrifices some fit on the training data but can improve generalization to new data.

3. Ridge Regression

Ridge regression is one shrinkage method. It extends the OLS cost function with an additional penalty term:

$$\hat{\beta}^{ridge} = \arg\min_{\beta}\left\{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\right\} \tag{8}$$

The first part of the function is identical to OLS, while the second part adds the penalty $\lambda\sum\beta_j^2$. For $\lambda = 0$, Ridge gives the same solution as OLS. As λ increases, large coefficients are penalized more heavily (quadratically), shrinking them toward zero.

The closed-form solution for Ridge coefficients is:

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{9}$$

where I is the $p \times p$ identity matrix. Ridge is especially useful when predictors are highly correlated, since OLS can produce unstable, excessively large coefficients that cancel each other out.

In the cases where the $X^T X$ term is not invertible, there is method for getting the pseudo-inverse of the identity matrix. The pseudo-inverse provides a minimum-norm least-squares solution. This must be used when some of the variables are linearly dependent, or there are more variables than observations. Pseudo-inverse is allways possible, it can be very computationally heavy for larger dataset, and can give unstable valuse when there are a lot of noise in the dataset.

4. LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO) is a shrinkage method similar to Ridge, but it uses the sum of absolute values of the coefficients as the penalty instead of the sum of squares:

$$\hat{\beta}^{lasso} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{10}$$

A key property of Lasso is that it can shrink some coefficients exactly to zero, effectively performing variable selection. This is particularly advantageous when dealing with many predictors, as irrelevant variables can be excluded from the model.

Unlike Ridge, Lasso does not have a closed-form solution. The problem arises because the L_1 penalty $|\beta_j|$ is not differentiable at zero, making it impossible to solve with standard derivative methods. Instead, numerical estimations has to be used to find the optimal coefficents. One example of this is gradient decent.

5. Scaling

In many cases, input data can vary greatly in magnitude. This often occurs when combining different types of measurements in a model, such as air pressure and temperature. Such differences can strongly affect model performance, especially in methods that include a penalty term. This is because the regularization term penalizes large coefficient magnitudes directly, and those magnitudes depend on the scale of the features.

Ordinary Least Squares (OLS) regression, on the other hand, automatically rescales coefficients and is therefore not directly affected by feature scaling. However, scaling often leads to smoother, faster, and more stable training, without changing the relationships between features and the target.

A common scaling method is **standardization**, where each feature is transformed to have mean = 0 and standard deviation = 1. This does not impose any upper or lower limits on feature values, unlike the **min–max scaler**, which rescales features to a fixed range such as [0, 1]. **Robust scaling** also centers data around zero, like standardization, but it uses the median and interquartile range instead of the mean and standard deviation—making it less sensitive to outliers.

For regression methods, the standard scaler is generally the most appropriate choice, since these models assume features that are roughly centered and standardized. The robust scaler may be useful for extremely noisy datasets, while min–max scaling is more common for models where distance or magnitude directly matters, such as neural networks.

6. Gradient Descent and Stochastic Gradient Descent

One cannot always get the exact solution, as is the case for LASSO-regression, but this problem also is common for Ordinary Least Squares (OLS) and Ridge regression, since getting an exact solution is only possible when $X^T X$ is invertible. When this is not possible, one alternative is to solve the problem using **Gradient Descent (GD)**.

Gradient Descent uses a loss function to optimize the coefficients. The *mean square error* is the standard loss function for regression tasks, but in principle, any performance metric can be used[1]. The goal is to minimize this loss. The procedure works as follows:

1. Compute the derivative of the loss function.
2. Initialize the coefficients with arbitrary values and calculate the loss (e.g., the sum of squared errors for all points in the dataset).
3. Update the coefficients by subtracting the derivative multiplied by a constant called the *learning rate*.
4. Repeat the process with the new coefficients until the step size is close to zero or the maximum number of iterations is reached.

The learning rate decides how big each step should be for the gradient decent method. Optimizing this parameter is essential to get the most out of gradient descent. With to small of a learning rate, your algorithm uses unnescery long time to find the optimal values. With to large learning rate, your alghorithm will make the coefficent explode. The optimal learning rate takes steps that are large enough on the areas that are not of interest, and when the degree of slope starts to get smaller it makes the steps small enough to nicely converge to the optimal coefficent values. The standard way is to use a fixed learning rate, and this has a large impact on the results. There are however, methods of adjusting the learning rate under training. Allthough the stepsize will shrink automatically towards where the derivative is smaller, this can make us have to have a very small learning rate to be sure we are getting the optimal coefficients. Using methods like *momentum*, *ADAgrad*, *RMSprop* and *ADAM*

- **momentum**: Introduces a velocity term that accumulates past gradients to accelerate learning in relevant directions and dampen oscillations. The parameter update rule is $v_t = \gamma v_{t-1} + \eta \nabla_\theta L(\theta)$ and $\theta \leftarrow \theta - v_t$, where γ is the momentum term.

- **ADAgrad**: Adapts the learning rate for each parameter individually, scaling the step size inversely proportional to the historical sum of squared gradients. This allows larger updates for infrequent features and smaller updates for frequent ones.

- **RMSprop**: Modifies AdaGrad by introducing an exponential moving average of squared gradients to prevent the learning rate from shrinking too much. The update rule uses $E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2$.

- **ADAM**: Combines the benefits of Momentum and RMSProp by maintaining running averages of both the gradients and their squared values, and applying bias corrections. It is widely used for its fast convergence and stability.

For large datasets, using standard GD can be very slow and may exceed available memory. In this case, **Stochastic Gradient Descent (SGD)** is preferred. Instead of calculating the loss over the entire dataset at once, SGD estimates it using only a small batch of data points (in the extreme case, a single data point).

The batch size is usually constrained by the available memory, but it is often recommended to choose a *power of two*, as such sizes have been shown to be more computationally efficient.

SGD splits the dataset into batches, where the number of batches and the number of data points in each batch is determined by the chosen batch size. Gradient descent is then performed on each batch in sequence, updating the coefficients after each one. Since the way the dataset is split can affect the outcome, the data is typically reshuffled between epochs to ensure that the batches are presented in different random orders.

7. Resampling Methods

Resampling methods are essential in modern statistics and machine learning because they allow us to assess model performance, estimate variability, and make the most of limited data. Instead of relying on a single train–test split, resampling repeatedly draws new samples from the data and refits the model, giving more reliable estimates of prediction error and parameter uncertainty.

Two widely used approaches are cross-validation and the bootstrap. Cross-validation splits the dataset into multiple folds, training on a subset and testing on the remaining fold, in order to estimate test error and guide model selection. The bootstrap, on the other hand, generates new datasets by sampling with replacement from the original data, which makes it possible to estimate the variability, bias, and confidence intervals of model parameters without relying on strong distributional assumptions. Together, these methods are powerful tools for both evaluating and improving statistical learning models.

8. Implementation

We tested Ordinary Least Squares (OLS), Ridge, and LASSO regression models on the same dataset. Initially, standard OLS and Ridge regressions were implemented using the pseudoinverse of the normal equations matrix ($X^T X$) to obtain the closed-form solutions. Subsequently, gradient descent was applied to all three regression methods, followed by the introduction of the optimization techniques discussed in Section II A 6. Stochastic Gradient Descent (SGD) was then implemented using the same optimization strategies.

In addition, we evaluated the use of cross-validation and bootstrap resampling to estimate model performance. The bootstrap analysis was performed only for the OLS model due to computational constraints. For each iteration, both predictive performance and the qualitative behavior of the algorithms were analyzed. All models were tested on the same dataset, with systematic variations in dataset properties to study how each method responds under different conditions.

For OLS and ridge regression the methods where tested both with and without scaling, the scaling used was the standard scaler from the scikit learn library, and we also used the train test function from scikit learn [**?**]. Throughout the rest of the conducted analysis the standard scaler was used.

The hyperparameters were not exhaustively optimized for each method. Instead, all models were evaluated over a broad range of parameter values to capture near-optimal performance while maintaining interpretability. This approach allows for a more transparent comparison between methods, even though it comes at the expense of an in-depth optimization discussion.

B. Dataset

The dataset is synthetically generated from the Runge function. We would like to test how the model performs when the data is under different levels of noise, and also check the impact of the dataset size. This is done using the same random state each time, to only measure the impacts of these factors and not totally different distributions, visualized in Figure 1.

The Runge function is a classical example in numerical analysis for illustrating the limitations of polynomial interpolation. Carl Runge showed that interpolating

$$f(x) = \frac{1}{1+x^2}, \quad x \in [-1, 1], \tag{11}$$

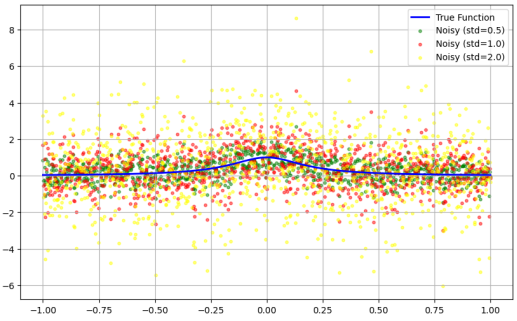

Figure 1: Dataset with same distribution, generated with three different levels of noise using standard deviation 1000 number of data points. The blue line is the Runge function.

with equally spaced nodes leads to large oscillations near the interval boundaries as the polynomial degree increases. This effect, known as *Runge's phenomenon*, arises because the interpolation error

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \qquad (12)$$

grows significantly near the endpoints, where the product term becomes large. In other words, the interpolation polynomial becomes unstable at the edges due to the lack of surrounding data points that would otherwise constrain its shape.

Although this is not directly a machine learning problem, it highlights an important principle relevant for regression: increasing model complexity (e.g., using higher-degree polynomials) does not guarantee better approximation. Instead, it may introduce instability and overfitting, especially in regions with sparse data support such as the boundaries of the input domain.

C. AI declaration

AI has been used to polish the language in my writing, where i have gone trough some drafts of certain sections of the text, given them as prompts and implemented some of the LLM's response. AI has also been used to help with Latex, scince I'm not very profient in this. I have also used AI to help with coding, since it is good at detecting the errors and sometimes make my code more readable and easier to use.

III. RESULTS AND DISCUSSION

A. Closed-form solutions

For the closed-form solutions, both Ordinary Least Squares (OLS) and Ridge regression were tested using datasets of 500 and 1000 data points. Each method was evaluated under three different noise levels (standard deviations): 0.5, 1.0, and 2.0. Polynomial degrees from 1 to 15 were used to examine how model complexity affects performance.

1. *OLS*

In Figure 2, we observe that the MSE between the training and test datasets does not deviate significantly. The test MSE steadily decreases before stabilizing around 0.24, indicating that the model continues to improve until it reaches a point of diminishing returns. Looking at the R^2 values, we see that the relative correlation increases similarly, although the training correlation eventually exceeds that of the test set at higher degrees. Both metrics appear to stabilize at approximately a polynomial degree of 10, suggesting that additional model complexity does not provide meaningful improvements in generalization performance.

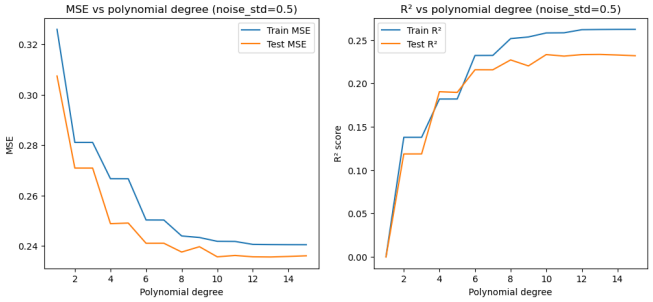

Figure 2: Closed-form OLS results with noise level std = 0.5. MSE and R^2 as functions of polynomial degree, using the scaled dataset with 1000 data points.

For the noisier dataset shown in Figure 3, the overall error has increased noticeably. The training MSE continues to decrease with increasing degree, but the test MSE now stabilizes earlier and even begins to rise slightly beyond a certain complexity. The R^2 score follows a similar trend, reaching its peak at around degree 8. These results illustrate that, beyond a certain point, the model begins to fit the noise in the data rather than the underlying trend — a clear sign of overfitting.

At an even higher noise level (std = 2.0), OLS performs only marginally better than simply predicting the mean of the dataset. The training error decreases slightly with increasing complexity, but the test MSE remains high and starts to rise already beyond degree 4 (see Figure 4). In this regime, the model fails to capture the signal and instead overfits to random fluctuations in the data.

Feature scaling did not affect the results beyond the sixth decimal place, which is expected for OLS since scaling does not change the underlying least-squares solution. When the dataset size was reduced to 500 samples, the test results became more sensitive to increasing complexity. In absolute terms, the overall performance also de-

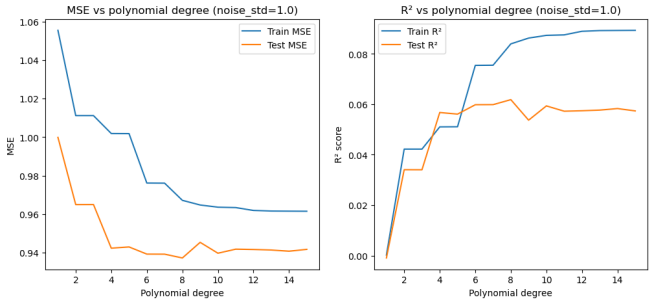

Figure 3: Closed-form OLS results with noise level std = 1.0. MSE and R^2 as functions of polynomial degree, using the scaled dataset with 1000 data points.

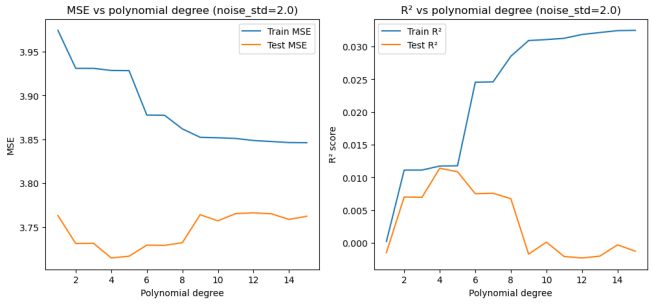

Figure 4: Closed-form OLS results with noise level std = 2.0. MSE and R^2 as functions of polynomial degree, using the scaled dataset with 1000 data points.

teriorated — most notably in the R^2 values for the test data.

2. *Ridge*

In Figure 5, we clearly observe that the models with lower penalty values (λ) perform better, and that the overall shape of the curves closely resembles the OLS results for the same noise level. The inclusion of a small regularization term helps reduce overfitting at higher polynomial degrees, while maintaining a low test error.

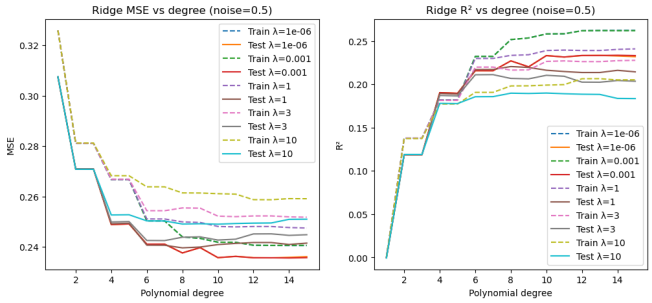

Figure 5: Closed-form Ridge results with noise level std = 0.5. MSE and R^2 as functions of polynomial degree, using the scaled dataset with 1000 data points.

When examining the noisier dataset in Figure 6, we can see that a moderate penalty term ($\lambda = 1$) yields slightly better results, particularly around a polynomial degree of 8. This demonstrates that the regularization term becomes more beneficial as the noise level increases, by constraining the model complexity and reducing variance.

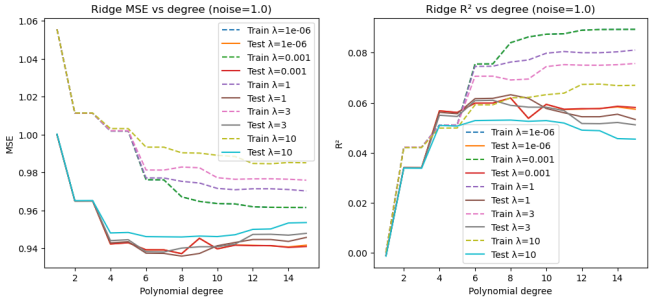

Figure 6: Closed-form Ridge results with noise level std = 1.0. MSE and R^2 as functions of polynomial degree, using the scaled dataset with 1000 data points.

The heatmap in Figure 7 illustrates more clearly how the penalty term λ affects the model's performance. The lowest MSE values are obtained for small to moderate regularization strengths, while both very small and very large λ values lead to higher errors.

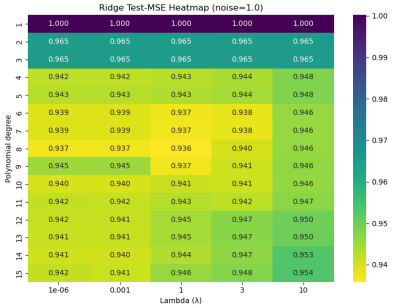

Figure 7: Closed-form Ridge results with noise level std = 1.0, showing the relationship between the penalty term λ, MSE, and polynomial degree. Results are from the scaled dataset with 1000 data points.

Overall, Ridge regression performs slightly better than standard OLS on the dataset with noise level std = 1.0, primarily because of its ability to limit overfitting through regularization. At higher noise levels (std = 2.0), however, Ridge does not provide a noticeable improvement over OLS, as the signal-to-noise ratio is too low for the regularization to recover meaningful structure.

When the dataset size is reduced to 500 samples, Ridge regression shows more stable results than OLS across all noise levels, although the improvement is smaller for the highest noise case (std = 2.0). Feature scaling appears to have a negative effect on the smaller dataset, producing

results that are nearly identical to OLS for very small λ values. With 1000 data points, however, scaling has little influence on the overall performance.

B. With Gradient Descent

1. *OLS and Ridge*

When analyzing the results from OLS and Ridge regression with gradient descent, we observe that the overall best performance is similar for both models across noise levels ($\sigma = 0.5$ and $\sigma = 0.1$). From Figure 8, it is clear that increasing the regularization strength (λ) leads to higher test MSE. This indicates that overly strong penalization reduces model flexibility, resulting in underfitting.

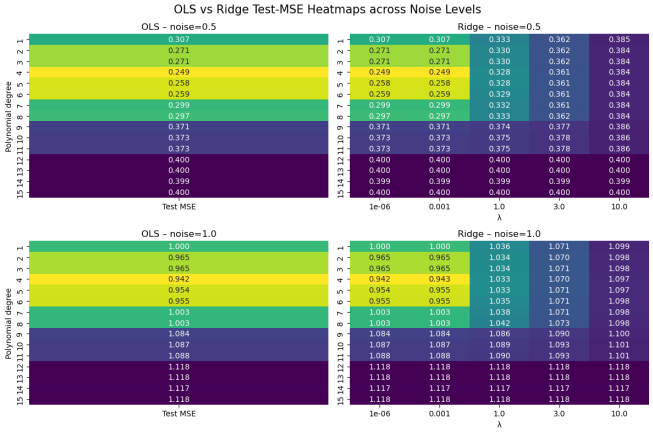

Figure 8: Comparison of OLS and Ridge regression test MSE across polynomial degrees and regularization strengths (λ) for different noise levels. Results are from the scaled dataset with 1000 data points.

When exploring the effect of the learning rate, we see that OLS and Ridge behave similarly when using the best-performing regularization term ($\lambda = 1 \times 10^{-6}$). This small λ value has negligible effect compared to standard OLS. As shown in Figure 9, higher learning rates can lead to divergence or unstable updates, which is reflected in the missing values (cases where the coefficients exploded or the algorithm failed to converge). Moderate learning rates achieve comparable results to the closed-form solution, while very low learning rates lead to slow convergence and higher error even for complex models.

Introducing optimizers on top of standard gradient descent further improves convergence and model performance, as illustrated in Figure 10. Apart from Momentum, all optimizers perform stably for most learning rates. However, RMSProp becomes unstable when both the polynomial degree and learning rate are high. Overall, AdaGrad and RMSProp do not outperform plain gradient descent, while Adam consistently provides stable and accurate convergence.

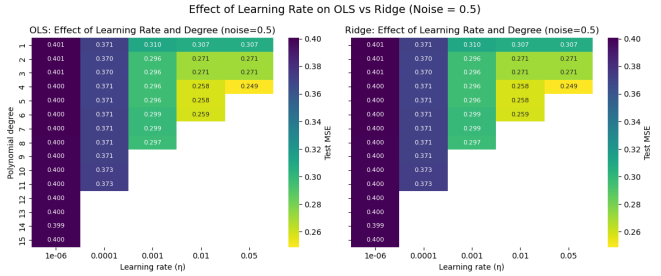

Figure 9: Comparison of OLS and Ridge regression test MSE across polynomial degrees and learning rates. For Ridge, $\lambda = 1 \times 10^{-6}$. Results are from the scaled dataset with 1000 data points.

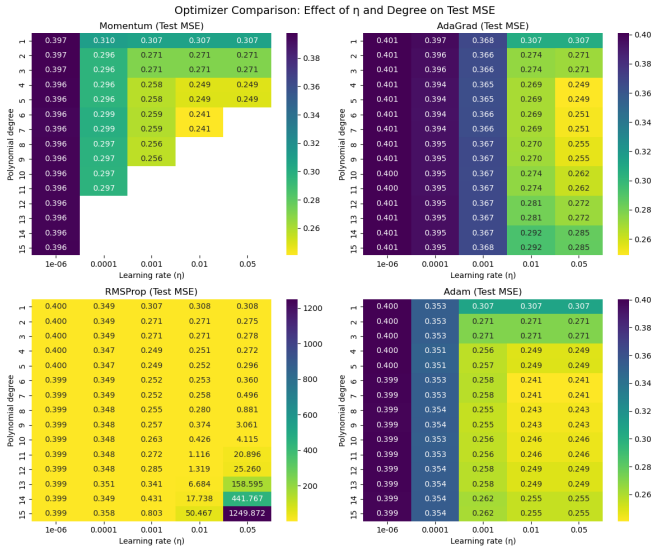

Figure 10: OLS gradient descent with different optimizers. Results are from the scaled dataset with 1000 data points and noise standard deviation $\sigma = 0.5$.

The Ridge results in Figure 11 show similar behavior. As before, large λ values lead to underfitting, while Adam demonstrates strong stability and performance across learning rates.

From the summary table in Figure 12, Adam achieves the lowest test MSE for Ridge as well. Overall, OLS and Ridge produce nearly identical performance patterns, with Adam emerging as the most robust optimizer.

2. *LASSO*

The LASSO regression results show similar patterns to OLS and Ridge (Figure 13). While RMSProp again struggles with high learning rates and polynomial degrees, the other optimizers outperform plain gradient descent. Adam once more stands out as the most consistent and stable optimizer, achieving the lowest test MSE across λ values.

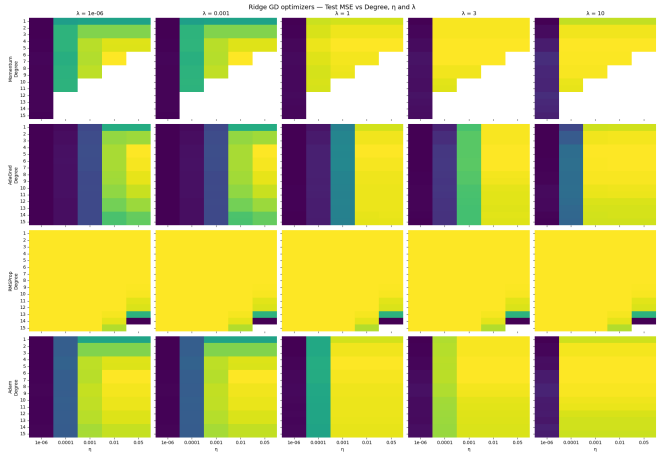

Figure 11: Heatmaps showing Test MSE across combinations of learning rate and polynomial degree for Ridge regression using different gradient descent optimizers (rows) and regularization strengths (columns). Results are based on the scaled dataset with noise standard deviation $\sigma = 0.5$.

	Method	Degree	Lambda	Eta	Test MSE
317999	AdaGrad	4	0.000001	0.050	0.248835
639999	Adam	7	0.001000	0.050	0.240525
612999	Momentum	7	0.000001	0.010	0.240962
310999	RMSProp	4	0.000001	0.001	0.249290

Figure 12: Lowest test MSE for each optimization method across combinations of regularization strength and learning rate.

As seen in Figure 14, Adam again achieves the lowest MSE, and LASSO slightly outperforms Ridge when both use Adam optimization. This demonstrates the effect of the L1 penalty in promoting sparsity and reducing overfitting in the higher-degree terms.

C. With Stochastic Gradient Descent

Using stochastic gradient descent (SGD), with all other settings kept fixed, the overall test error tends to increase slightly compared to full-batch gradient descent. As expected, performance generally improves with larger batch sizes because the gradient estimate becomes less noisy. Nevertheless, even a small batch size (e.g., $B = 16$) produces competitive results across all model types.

For OLS (Figure 15), the Momentum optimizer yields the best result at the largest batch size ($B = 256$), while Adam is superior for smaller batches. In general, the estimates improve as the batch size increases, whereas overly large learning rates lead to instability.

For Ridge (Figure 16), we observe the same qualitative trends: larger batches improve stability, and strong regularization (large λ) leads to underfitting. Small to moderate regularization (e.g., $\lambda \approx 10^{-3}$) combined with a moderate learning rate typically performs best. In our grid, Adam achieves the lowest error (approximately Test MSE $= 0.241$) at $d = 7$, $\lambda = 10^{-3}$, $\eta = 0.05$, and $B = 256$.

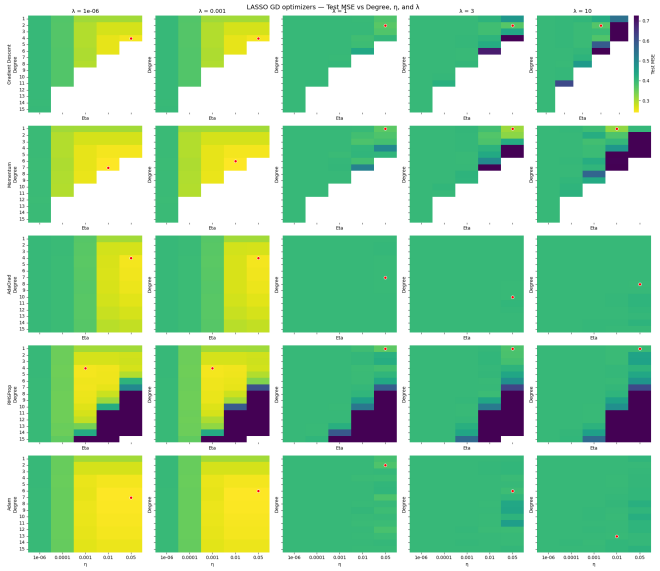

Figure 13: Heatmaps showing Test MSE across combinations of learning rate and polynomial degree for LASSO regression using different gradient descent optimizers (rows) and regularization strengths (columns). The best result in each heatmap is marked in red. Values above 0.95 were clipped to preserve a consistent color scale. Results are based on the scaled dataset with noise standard deviation $\sigma = 0.5$.

	Method	Lambda	Eta	Degree	Test MSE
	AdaGrad	0.000001	0.050	4	0.248835
	Adam	0.001000	0.050	6	0.240267
	Gradient Descent	0.000001	0.050	4	0.248882
	Momentum	0.000001	0.010	7	0.240962
	RMSProp	0.000001	0.001	4	0.249290

Figure 14: Best-performing combinations of λ and η for each optimizer applied to LASSO regression. Results are from the scaled dataset with noise standard deviation $\sigma = 0.5$.

For LASSO (Figure 17), the errors are slightly higher overall than for Ridge, consistent with the stronger bias induced by the ℓ_1 penalty. We also observe that some optimizers achieve their best results with smaller batches, while Adam remains the most robust across learning rates and degrees. Further analysis (e.g., coefficient paths or sparsity patterns) would help clarify when smaller batches are preferable for LASSO.

Method	Degree	Eta	Batch	Test MSE
AdaGrad	4	0.05	256	0.249719
Adam	7	0.05	256	0.240865
Momentum	6	0.01	256	0.240803
RMSProp	4	0.01	128	0.248264
SGD	4	0.05	128	0.249491
Method	Degree	Eta	Batch	Test MSE
Adam	5	0.05	16	0.252313
Adam	8	0.01	32	0.249882
Adam	7	0.05	64	0.246461
Adam	6	0.05	128	0.243957
Momentum	6	0.01	256	0.240803

Figure 15: OLS with SGD: Test MSE across polynomial degrees and learning rates η for different batch sizes B (columns) and optimizers (rows). The best cell in each heatmap corresponds to the lowest Test MSE for that (optimizer, B) combination. Results are computed on the scaled dataset ($n = 1000$, noise std $\sigma = 0.5$).

Method	Degree	Lambda	Eta	Batch	Test MSE
AdaGrad	4	0.001000	0.05	256	0.249659
Adam	7	0.001000	0.05	256	0.241024
Momentum	6	0.001000	0.01	256	0.243722
RMSProp	7	0.000001	0.01	256	0.247415
SGD	4	0.000001	0.05	64	0.248120
Method	Lambda	Degree	Eta	Batch	Test MSE
RMSProp	0.000001	4	0.01	16	0.249094
Adam	0.001000	8	0.01	32	0.249656
Adam	0.000001	6	0.05	64	0.245532
Adam	0.001000	9	0.01	128	0.244624
Adam	0.001000	7	0.05	256	0.241024

Figure 16: Ridge with SGD: Test MSE across polynomial degrees and learning rates η for different regularization strengths λ (columns), batch sizes B, and optimizers (rows). Results are based on the scaled dataset ($n = 1000$, noise std $\sigma = 0.5$). Strong regularization underfits, while small to moderate λ combined with Adam or Momentum yields the best results.

D. Resampling

1. *Bootstrap*

Figure 18 shows the bias- variance tradeoff in practice. We can see the MSE decreases steadily before stabilizing between 10-15 polynomial degree, before starting to rise again, before becomming extremely high at 25 polynomial degree. When linking this with bias and variance, we can see the model is very well generalized at the start, but results are bad due to high bias, then the model gets more complex and the bias sinks with variance still relatively low, leading to a good overall MSE.

Method	Degree	Lambda	Eta	Batch	Test MSE
AdaGrad	5	0.000001	0.05	256	0.249926
Adam	6	0.001000	0.05	64	0.241175
Momentum	6	0.000001	0.01	256	0.241956
RMSProp	9	0.000001	0.01	64	0.248370
SGD	4	0.001000	0.05	128	0.249704
Method	Lambda	Degree	Eta	Batch	Test MSE
Adam	0.000001	4	0.05	16	0.249629
Adam	0.001000	6	0.05	32	0.246560
Adam	0.001000	6	0.05	64	0.241175
Adam	0.001000	8	0.01	128	0.245855
Momentum	0.000001	6	0.01	256	0.241956

Figure 17: LASSO with SGD: Test MSE across polynomial degrees and learning rates η for different regularization strengths λ (columns), batch sizes B, and optimizers (rows). Results are computed on the scaled dataset ($n = 1000$, noise std $\sigma = 0.5$). LASSO shows slightly higher error on average than Ridge, while Adam is again the most stable optimizer.

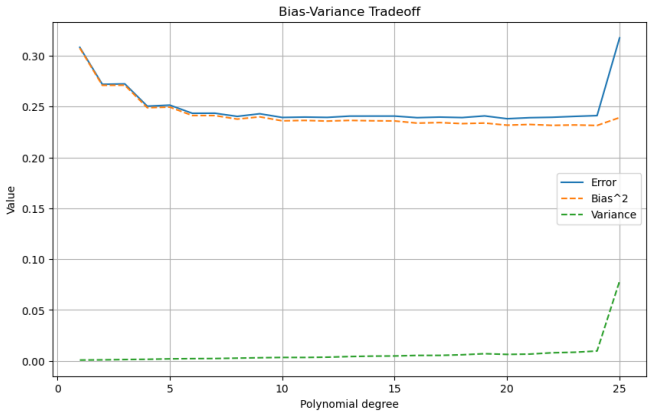

Figure 18: Here we can see the MSE of OLS with bootstrap resampling over 25 polynomial degrees, as well as $bias^2$ and variance

2. *Cross-Validation*

During k-fold cross-validation the GD-based LASSO routine occasionally diverged. This behavior is consistent with the sensitivity of subgradient descent on ℓ_1 objectives: large polynomial degrees yield ill-conditioned feature matrices, and fixed step sizes can overshoot near the non-differentiable kink at zero. In practice this is an optimizer issue rather than a flaw in the CV procedure, and can be mitigated by smaller learning rates, more iterations, adaptive/decaying step sizes, proximal updates (ISTA/FISTA) instead of plain subgradients, or

by switching to a coordinate-descent LASSO solver.

Figure 19 shows that cross-validation works robustly for OLS and Ridge, though CV errors are slightly higher than those from the bootstrap analysis (as expected: CV is typically more conservative). Importantly, both assessments lead to the same qualitative conclusions about model complexity and regularization strength.

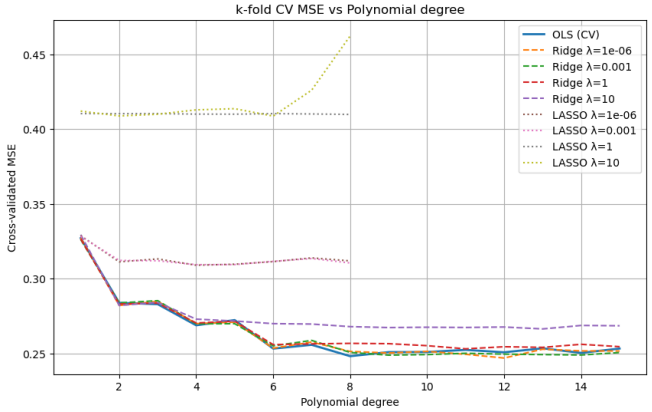

Figure 19: Five-fold cross-validated MSE versus polynomial degree for closed-form OLS and Ridge, and LASSO trained with gradient descent (fixed step size). Inputs are standardized within each fold to avoid leakage.

IV. CONCLUSION

This study compared closed-form estimators (OLS and Ridge) with gradient–based training (full-batch GD, optimizers, and SGD) across a controlled synthetic setting with varying polynomial degrees, noise levels, dataset sizes, and regularization strengths. The main findings are consistent with the bias–variance trade-off and standard regularization theory, and they translate into clear practical guidance.

Some key findings where:

- **Model complexity.** Test error decreases with degree at first and then increases, revealing a clear sweet spot. With n=1000 and moderate noise ($\sigma \in [0.5, 1.0]$), the best degrees typically lie in the mid range (roughly $d \approx$ 4–10), while higher noise shifts the optimum toward lower degrees and increases variance.

- **Closed-form OLS vs. Ridge.** For low noise, OLS and lightly regularized Ridge perform similarly. As noise increases, small to moderate Ridge penalties (λ) reduce variance and modestly improve test MSE; overly large λ underfits. At very high noise (σ=2.0), neither OLS nor Ridge recovers much signal.

- **Optimization with full-batch GD.** With a well-tuned learning rate, GD matches closed-form performance. Too large η causes divergence; too small η converges slowly and yields higher test error at fixed compute. Among optimizers, **Adam** was consistently the most robust across degrees and noise, while RMSProp and AdaGrad were more sensitive to η and model complexity.

- **Stochastic gradient descent (SGD).** Holding other settings fixed, SGD tends to yield slightly higher test error than full-batch GD, but remains competitive—especially with larger batches, which stabilize the stochastic gradient. Even small batches (e.g., B=16) can perform well with Adam or Momentum and a moderate η.

- **LASSO.** Patterns mirror Ridge/OLS: performance hinges on moderate complexity and careful step-size control. With adaptive optimizers (notably Adam), LASSO occasionally *slightly* outperformed Ridge by promoting sparsity in higher-order coefficients. Plain subgradient LASSO can be unstable at larger degrees unless η is reduced or proximal steps are used.

- **Resampling.** Bootstrap and k-fold CV led to the same qualitative conclusions. CV produced slightly higher (more conservative) errors, while bootstrap curves illustrated the bias–variance decomposition and the emergence of overfitting at high degrees.

A. Future work

The experiments focus on univariate polynomial regression and fixed noise distributions. Future work could explore multivariate settings, alternative feature maps (e.g., splines, kernels), adaptive or decaying learning-rate schedules, early stopping, and proximal/coordinate-descent solvers for LASSO. Extending the analysis to real datasets and reporting uncertainty (e.g., via repeated CV or nested CV) would further validate the conclusions.

B. Overall

Across estimators and training regimes, the results reinforce that (i) moderate complexity with modest regularization yields the best generalization; (ii) Adam is a robust default for gradient-based fitting; and (iii) systematic model selection via cross-validation is essential—particularly as noise increases or dataset size decreases.

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016), `http://www.deeplearningbook.org`.
[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] M. Mahesh, *Runge's phenomenon*, `https://deepnote.com/app/ecnm1/ALEMOHW-b35f068a-9fae-449a-92c7-1271830670dd` (2024), besøkt dd. mm. åååå.

Appendix A: Bias Variance expression

Let the data be generated by

$$y = f(x) + \varepsilon, \qquad \mathbb{E}[\varepsilon] = 0, \quad \mathrm{Var}(\varepsilon) = \sigma^2,$$

and let $\hat{y} = \hat{y}(x; \mathcal{D})$ be the prediction of a learning procedure trained on a random dataset $\mathcal{D}$. Expectations below are with respect to all sources of randomness (the sampling of $\mathcal{D}$ and the noise ε).

The mean–squared error (the cost used by OLS) can be written as

$$C(X, \beta) \;=\; \mathbb{E}\big[(y - \hat{y})^2\big] .$$

Insert $y = f(x) + \varepsilon$ and expand:

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

Expanding:

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

Because ε is independent of $\hat{y}$ and $\mathbb{E}[\varepsilon] = 0$, the cross term vanishes and $\mathbb{E}[\varepsilon^2] = \sigma^2$. Hence

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

Write $\mathbb{E}[\hat{y}] = \mu_{\hat{y}}(x)$ and add/subtract it inside the square:

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

This expands to

$$= \underbrace{\big(f(x) - \mu_{\hat{y}}(x)\big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}\big[(\hat{y} - \mu_{\hat{y}}(x))^2\big]}_{\text{variance}} .$$

Therefore the pointwise bias–variance decomposition is

$$\mathbb{E}\big[(y - \hat{y})^2\big] = \underbrace{\big(\mathbb{E}[\hat{y}] - f(x)\big)^2}_{\mathrm{Bias}^2[\hat{y}]} + \underbrace{\mathrm{Var}[\hat{y}]}_{\mathbb{E}[(\hat{y} - \mathbb{E}[\hat{y}])^2]} + \underbrace{\sigma^2}_{\text{noise}} .$$

For a dataset $\{x_i\}_{i=1}^n$ the average MSE decomposes as

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\big[(y_i - \hat{y}_i)^2\big] = \frac{1}{n}\sum_{i=1}^{n} \big(\mathbb{E}[\hat{y}_i] - f_i\big)^2 + \frac{1}{n}\sum_{i=1}^{n} \mathrm{Var}[\hat{y}_i] + \sigma^2,$$

with $f_i = f(x_i)$.

$$\mathrm{Bias}^2[\hat{y}_i] = \big(\mathbb{E}[\hat{y}_i] - f_i\big)^2, \qquad \mathrm{Var}[\hat{y}_i] = \mathbb{E}\big[(\hat{y}_i - \mathbb{E}[\hat{y}_i])^2\big] .$$

Since f_i is unknown, a common plug-in approximation (e.g. in bootstrap or repeated train/validation splits) is to replace f_i by the observed target y_i:

$$\widehat{\mathrm{Bias}^2} \;\approx\; \frac{1}{n}\sum_{i=1}^{n} \big(\bar{\hat{y}}_i - y_i\big)^2,$$

$$\widehat{\mathrm{Var}} \;\approx\; \frac{1}{n}\sum_{i=1}^{n} \frac{1}{B}\sum_{b=1}^{B} \big(\hat{y}_i^{(b)} - \bar{\hat{y}}_i\big)^2,$$

where $\hat{y}_i^{(b)}$ is the prediction from the b-th resample/split and $\bar{\hat{y}}_i = \frac{1}{B}\sum_{b=1}^{B} \hat{y}_i^{(b)}$. Finally, σ^2 is the irreducible noise variance of the data-generating process.