Data Analysis and Machine Learning
FYS-STK3155 - Project 1

See Gek Cheryl Ong, Boon Kiat Khaw, Naden Jarel Anthony Koh
University of Oslo
(Dated: October 6, 2025)

Regression is pivotal and forms the foundation for many statistics and machine learning methods. Although many regression techniques today can approximate accurately, high-degree polynomial regression still proves to be a challenge, as it often suffers from overfitting and instability. Therefore, our project aims to investigate the performance of several regression techniques - Ordinary Least Squares (OLS), Ridge regression, and LASSO regression in the polynomial fitting of Runge's function. Through our results and analysis, we aim to extract valuable insights into both statistical analysis and practical machine learning for analyzing high-degree polynomials.

I. INTRODUCTION

We begin with OLS, implemented both directly and via gradient descent, before extending the analysis to Ridge and LASSO. In order to evaluate the role of regularization in improving stability, we apply bootstrap resampling and cross-validation to study the bias-variance tradeoff. Finally, we conclude with an analysis of optimization algorithms such as Momentum and AdaGrad, comparing their stability under different conditions.

II. METHODS

A. Method 1/X

To investigate model complexity and regularisation effects, we approximated the one-dimensional Runge function using polynomial regression. The input variable x was uniformly sampled from $[-1, 1]$ and used to construct polynomial features up to degree 15. The corresponding target values y were generated by evaluating the Runge function with added Gaussian noise to simulate measurement error.

We divided the dataset into training and test subsets (70% and 30% respectively) and applied `StandardScaler` from scikit-learn [1] to standardise x before constructing polynomial features. Scaling is essential to ensure that all polynomial terms remain numerically comparable, preventing dominance by higher-order terms and ensuring stable convergence during optimisation.

Model performance was assessed using the mean squared error (MSE) and coefficient of determination (R^2) for both training and test sets. These metrics were chosen to capture both absolute error magnitude and relative explanatory power.

In Part (c), we aimed to replicate the ordinary least squares (OLS) regression results using an iterative optimisation approach rather than the closed-form analytic solution. We employed gradient descent (GD) with a fixed learning rate to minimise the mean squared error (MSE) cost function:

$$C(\boldsymbol{\theta}) = \frac{1}{n}\|y - X\boldsymbol{\theta}\|^2.$$

The gradient of this cost function with respect to the coefficients is given by

$$\nabla_{\boldsymbol{\theta}} C = \frac{2}{n} X^T (X\boldsymbol{\theta} - y),$$

and the update rule is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \gamma \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}^{(k)}),$$

where γ is the fixed learning rate. The algorithm terminates when the gradient norm falls below a defined tolerance or when the maximum number of iterations is reached.gradient norm falls below a tolerance threshold or when the maximum iteration count is reached.

Part (d) compares four full-batch optimisers for least-squares regression: Momentum, AdaGrad, RMSProp, and Adam. We minimise the regularised quadratic objective

$$C(\boldsymbol{\theta}) = \frac{1}{n}\|y - X\boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}_{1:}\|_2^2,$$

where the intercept θ_0 is not regularised. All methods use the full-batch gradient

$$\nabla C(\boldsymbol{\theta}) = \frac{2}{n} X^\top (X\boldsymbol{\theta} - y) + 2\lambda\, [0; \boldsymbol{\theta}_{1:}],$$

and differ only in their update rules (fixed-step Momentum; per-parameter scaling for AdaGrad and RMSProp; bias-corrected first/second moments for Adam).

To study convergence, we compute the closed-form training optimum

$$\boldsymbol{\theta}^\star = \left(X^\top X + n\lambda\, I_{(\text{bias off})}\right)^{-1} X^\top y,$$

and track (i) the parameter error $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}^\star\|_2$ and (ii) the *test* MSE at every iteration.

In Part (e) we extend the gradient–based approach to LASSO regression. The optimisation problem is

$$C(\boldsymbol{\theta}) = \frac{1}{n}\|y - X\boldsymbol{\theta}\|^2 + \lambda\|\boldsymbol{\theta}_{1:}\|_1,$$

where the intercept θ_0 is not penalised. Because the ℓ_1 term is non-differentiable, we employ the proximal gradient method (ISTA): a gradient step on the smooth squared-error term followed by soft-thresholding on the coefficients $\theta_{1:}$,

$$\boldsymbol{\theta}_0^{k+1} = \boldsymbol{\theta}_0^k, \qquad \boldsymbol{\theta}_{1:}^{k+1} = \mathcal{S}_{\lambda t}\Big(\boldsymbol{\theta}_{1:}^k - t\nabla \tfrac{1}{n}\|y - X\boldsymbol{\theta}^k\|^2\Big),$$

with step size $t = 1/L$, where $L = \frac{2}{n}\lambda_{\max}(X^\top X)$. To ensure comparability with scikit-learn, we standardise polynomial features (columns $1 : p{-}1$) using training statistics and leave the intercept unscaled; we also match regularisation by using $\alpha = \lambda/2$ since scikit-learn [1] minimises $(1/(2n))\|y{-}X\theta\|^2{+}\alpha\|\theta\|_1$, and we fit scikit-learn's `Lasso` [1] with `fit_intercept=False`.

In Part (f) we extend the optimisation in Parts (c)–(d) from *full-batch* gradient methods to *stochastic/mini-batch* updates. Unless otherwise stated we solve ordinary least squares (OLS) on the training split,

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} C(\boldsymbol{\theta}) = \frac{1}{n}\|y - X\boldsymbol{\theta}\|_2^2,$$

with the intercept included in X (column of ones) and not regularised. We keep the same data pipeline as in Parts (a)–(e).

At iteration t a batch $\mathcal{B}_t \subset \{1,\ldots,n\}$ of size b is drawn and we use the unbiased gradient estimate

$$\mathbf{g}_t = \frac{2}{|\mathcal{B}_t|} X_{\mathcal{B}_t}^\top\big(X_{\mathcal{B}_t}\boldsymbol{\theta}_t - y_{\mathcal{B}_t}\big), \qquad \mathbb{E}[\mathbf{g}_t] = \nabla C(\boldsymbol{\theta}_t).$$

We study two sampling schemes that match our code:

- **v1 (with replacement):** each step draws indices i.i.d. with replacement.
- **v2 (without replacement per epoch):** we shuffle the n indices once per epoch and traverse them in contiguous mini-batches.

These schemes have identical expectation but different gradient *variance*: v2 tends to yield smoother, more monotone progress within an epoch.

In part (g), we replicate the qualitative shape of Fig. 2.11 in Hastie–Tibshirani–Friedman by plotting *training MSE* and *test MSE* as functions of model complexity (polynomial degree). The test curve is then used to reason qualitatively about high-bias (left) and high-variance (right) regions.

We use the one–dimensional Runge function on $[-1, 1]$ as the ground truth,

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

and generate noisy observations $y = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. We fit Ordinary Least Squares (OLS) separately for each degree d:

$$\hat{\boldsymbol{\theta}}_d \;=\; \arg\min_{\boldsymbol{\theta}} \; \frac{1}{n_{\text{train}}}\|y_{\text{train}} - X_{d,\text{train}}\boldsymbol{\theta}\|_2^2,$$

and compute

$$\text{MSE}_{\text{train}}(d) = \frac{1}{n_{\text{train}}}\|y_{\text{train}} {-} X_{d,\text{train}}\hat{\boldsymbol{\theta}}_d\|_2^2, \quad \text{MSE}_{\text{test}}(d) = \frac{1}{n_{\text{test}}}$$

We sweep degrees up to a relatively large $d_{\max}$ (e.g. $d_{\max} \in [15, 30]$) to expose the high-variance regime. Because single splits can be noisy, we optionally *repeat* the whole train/test split and fit process R times (e.g. $R = 30$) with different random seeds and report the mean MSE across repeats.

For Bias-Variance Trade-off analysis, to approximate expectations over the data–generating process without collecting new datasets, we use the nonparametric bootstrap on the *training* split and evaluate predictions on a fixed *hold-out* test split. Let $(X_{\text{tr}}, y_{\text{tr}})$ and $(X_{\text{te}}, y_{\text{te}})$ denote the training and test sets, respectively, with $n_{\text{test}} = |y_{\text{te}}|$. For each polynomial degree d and for $b = 1, \ldots, B$ bootstrap replicates:

1. Draw indices with replacement:

$$I^{(b)} \sim \text{Unif}\{1,\ldots,n_{\text{train}}\}^{n_{\text{train}}}, \qquad X_{\text{tr}}^{(b)} = X_{\text{tr}}[I^{(b)}], \;\; y_{\text{tr}}^{(b)} = y_{\text{tr}}[I$$

2. Fit the degree-d model on $(X_{\text{tr}}^{(b)}, y_{\text{tr}}^{(b)})$ to obtain parameters $\hat{\boldsymbol{\theta}}^{(d,b)}$ (OLS in our experiments).
3. Predict on the *same* fixed test inputs:

$$\hat{\mathbf{y}}^{(d,b)} = X_{\text{te}}^{(d)}\,\hat{\boldsymbol{\theta}}^{(d,b)} \in \mathbb{R}^{n_{\text{test}}}.$$

Stacking the B prediction vectors column-wise yields the matrix $\hat{Y}^{(d)} = \big[\hat{\mathbf{y}}^{(d,1)} \,\cdots\, \hat{\mathbf{y}}^{(d,B)}\big] \in \mathbb{R}^{n_{\text{test}}\times B}$. Writing $\hat{y}_i^{(d,b)}$ for the prediction at test point i, we define per-point bootstrap moments

$$\mu_i^{(d)} \;=\; \frac{1}{B}\sum_{b=1}^{B}\hat{y}_i^{(d,b)}, \qquad \text{Var}_i^{(d)} \;=\; \frac{1}{B-1}\sum_{b=1}^{B}\big(\hat{y}_i^{(d,b)} {-} \mu_i^{(d)}\big)^2,$$

and aggregate them across test points to obtain the displayed curves:

$$\text{Variance}(d) \;=\; \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\text{Var}_i^{(d)},$$

$$\text{Bias}^2_{\text{noisy}}(d) \;=\; \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\big(\mu_i^{(d)} - y_i^{\text{test}}\big)^2,$$

$$\text{MSE}_{\text{test}}(d) \;=\; \frac{1}{B}\sum_{b=1}^{B}\left[\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\big(y_i^{\text{test}} - \hat{y}_i^{(d,b)}\big)^2\right].$$

The bias term matches our implementation choice of comparing the bootstrap mean to the *noisy* test targets. (Optionally, one may also report a "noise-free" bias by replacing y_i^{test} with $f(x_i^{\text{test}})$ when the ground-truth f is known.)

We fix the random seed for data generation, the train/test split, and the bootstrap indices to ensure reproducibility. Increasing B reduces the Monte Carlo noise of these estimates at rate $\mathcal{O}(B^{-1/2})$, yielding smoother curves at higher computational cost.

For part (h),for each degree d, we repeatedly refit a model on resampled versions of the *training* data and predict on the *fixed* test set. Let $\hat{y}_i^{(d,b)}$ be the prediction at test point i from refit b ($b = 1, \ldots, B$ where $B = k$ for K-fold and B is the number of bootstrap replicates). Define the per-point mean and (sample) variance

$$\mu_i^{(d)} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}_i^{(d,b)}, \qquad \mathrm{Var}_i^{(d)} = \frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{y}_i^{(d,b)} - \mu_i^{(d)}\right)^2,$$

and report the degree-wise curves

$$\mathrm{Variance}(d) = \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}} \mathrm{Var}_i^{(d)}, \quad \mathrm{Bias}^2_{\text{noisy}}(d) = \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\left(\mu_i^{(d)} - y_i^{\text{test}}\right)^2.$$

$$\mathrm{MSE}_{\text{test}}(d) = \frac{1}{B}\sum_{b=1}^{B}\left[\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\left(y_i^{\text{test}} - \hat{y}_i^{(d,b)}\right)^2\right].$$

The bias term uses the *noisy* test targets y_i^{test} to match the code.

a. K-fold refitting (OLS). For OLS we sweep degrees $d = 1{:}23$ with $k = 10$ folds. We use `KFold` with shuffling and a fixed seed.

b. K-fold refitting (Ridge). For Ridge we sweep $d = 1{:}20$ with $k = 5$ folds and $\lambda = 0.5$. Within each fold we repeat the same scaling protocol as OLS, then fit

$$\hat{\theta}_{\text{ridge}} = \left(X^\top X + \lambda\, I_{(\text{bias off})}\right)^{-1} X^\top y,$$

leaving the intercept unpenalised, and predict on the fixed test set.

c. Bootstrap (Ridge). For Ridge with bootstrap we use $B = 100$ replicates and degrees $d = 1{:}18$ with $\lambda = 0.5$. Here the inputs are scaled *once globally* on the training set (yielding $x_{\text{train,scaled}}$ and $x_{\text{test,scaled}}$), then we: (i) build $X_{\text{train}}^{(d)}$ and $X_{\text{test}}^{(d)}$ once; (ii) for each replicate, sample training row indices with replacement, fit Ridge on the resampled rows of $X_{\text{train}}^{(d)}$, and predict on $X_{\text{test}}^{(d)}$; (iii) aggregate $\mu_i^{(d)}$, $\mathrm{Var}_i^{(d)}$, $\mathrm{MSE}_{\text{test}}(d)$.

d. K-fold refitting (LASSO via ISTA). For LASSO we sweep $d = 1{:}50$ with $k = 5$ and $\lambda = 10^{-4}$. Within each fold: (i) fit a `StandardScaler` on the fold's training inputs x and transform both the fold-train inputs and the global test inputs; (ii) build $X^{(d)}$; (iii) standardise *columns* $1{:}p{-}1$ *of* X (mean/SD from the fold-train design; intercept left unscaled) to stabilise ℓ_1 updates; (iv) solve

$$\min_{\theta}\ \frac{1}{n}\|y - X\theta\|_2^2 + \lambda\|\theta_{1:}\|_1$$

with ISTA (no additional internal standardisation), keeping the intercept unpenalised; (v) predict on the fixed test set and aggregate as above.

e. Bootstrap (LASSO via ISTA). For LASSO with bootstrap we use $B = 200$ replicates and degrees $d = 1{:}20$ with $\lambda = 10^{-2}$. For each replicate: (i) sample training rows with replacement; (ii) fit a `StandardScaler` on the replicate's inputs and transform the replicate-train and the global test inputs; (iii) build $X^{(d)}$ and standardise columns $1{:}p{-}1$ using replicate-train design statistics (intercept unscaled); (iv) run ISTA as above and predict on the fixed test set; (v) aggregate $\mu_i^{(d)}$, $\mathrm{Var}_i^{(d)}$, $\mathrm{MSE}_{\text{test}}(d)$.

f. Implementation details and reproducibility. All K-fold splits use shuffling with a fixed `random_state`; bootstrap replicates use a fixed PRNG seed. For K-fold, scaling is fit *inside each fold* and applied to the test inputs; for Ridge bootstrap, we intentionally use a *single global* scaler (matching the code). For LASSO, the intercept is never penalised and design columns $1{:}p{-}1$ are standardised per-fold/per-replicate prior to ISTA. Sample variances use ddof = 1.

B. Implementation

For Part (a), we implemented OLS regression. The model coefficients $\boldsymbol{\beta}$ were estimated using the closed-form normal equation:

$$\boldsymbol{\beta}_{OLS} = (X^T X)^{-1} X^T y,$$

where X denotes the polynomial design matrix including the intercept term. Predictions were then obtained via $\hat{y} = X\boldsymbol{\beta}_{OLS}$.

For Part (b), we extended this approach to Ridge regression by introducing an L_2 regularisation term. The coefficients were estimated as:

$$\boldsymbol{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y,$$

where λ is the regularisation parameter controlling the strength of the penalty. Setting $\lambda = 0$ recovers the OLS solution. Both models were implemented in Python using NumPy, with polynomial feature generation and scaling performed manually to maintain consistency across all subsequent analyses.

For part (c), the gradient descent algorithm was implemented in Python using NumPy for efficient matrix computations. We adopted a fixed learning rate of $\gamma = 10^{-3}$ and a gradient tolerance of 10^{-8}, with a maximum of 200,000 iterations. The same polynomial design matrices used in Parts (a) and (b) were employed, spanning polynomial degrees 1 to 15. For each degree, the model was trained on the training set, predictions were generated for both training and test data, and performance was measured using the mean squared error (MSE) and the coefficient of determination (R^2). The results were then plotted as MSE and R^2 versus polynomial degree.

For part (d), we implemented four *pure batch* optimisers in Python/NumPy: `gd_momentum_batch`, `adagrad_batch`, `rmsprop_batch`, and `adam_batch`. Each iteration computes one full gradient on the training design matrix and updates parameters accordingly. A helper routine executes each optimiser for T iterations with `max_iters=1` per call to record a parameter path, per-iteration loss, and gradient norms.

For degrees $d \in \{3, 6, 9\}$ we: (1) build $X_{\text{train}}, X_{\text{test}}$ with a degree-d polynomial design (intercept included), (2) compute $\boldsymbol{\theta}^\star$ on the training split, (3) run each optimiser for $T = 1000$ full-batch steps, (4) record the parameter error $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}^\star\|_2$ and test MSE per iteration, and (5) plot/snapshot both trajectories.

Unless stated otherwise, we used: Momentum ($\eta = 2 \cdot 10^{-2}$, $\mu = 0.9$), AdaGrad ($\eta = 5 \cdot 10^{-2}$, $\delta = 10^{-8}$), RMSProp ($\eta = 10^{-2}$, $\rho = 0.9$, $\delta = 10^{-8}$), Adam ($\eta = 5 \cdot 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\delta = 10^{-8}$). We evaluate two regimes: **OLS** ($\lambda = 0$) and **Ridge** ($\lambda = 0.01$), with the intercept excluded from regularisation.

For part (e), we implemented ISTA with element-wise soft-thresholding and an automatically chosen step size $t = 1/L$. After building degree-d polynomial design matrices, we standardised columns $1 : p-1$ with training means and standard deviations and reused the same transform for the test set. For each pair (degree d, regularisation λ) on a grid $d \in \{1, \ldots, 15\}$ and $\lambda \in [10^{-4}, 10^{1}]$, we:

1. solved LASSO with ISTA to obtain $\hat{\boldsymbol{\theta}}_{\text{ISTA}}$,
2. fit scikit-learn's `Lasso` with `fit_intercept=False` and $\alpha = \lambda/2$ on the same scaled design,
3. evaluated Test MSE and Test R^2 for both solutions,
4. populated two heatmaps (MSE and R^2) with degree on the y-axis and $\log_{10}(\lambda)$ on the x-axis.

For part (f), fixing a representative mini-batch configuration (e.g. Adam with $b = 32$, a fixed number of epochs E and steps-per-epoch S), we sweep polynomial degrees $d \in \{1, \ldots, 15\}$. All Adam hyperparameters are kept *fixed across d* to isolate the effect of model complexity; the total update budget is held constant at $U = E \times S$ updates for every degree (for v2, one epoch $\approx \lceil n/b \rceil$ updates). For each (d, sampler) pair (v1: with replacement, v2: without replacement).

Fixing a representative degree (e.g. $d = 9$), we sweep batch sizes $b \in \{8, 16, 32, 64, 128\}$ for both samplers (v1/v2). We keep the Adam hyperparameters *fixed across b* to isolate the effect of batch-size–induced gradient noise; number of updates is reported directly (for v2, one epoch $\approx \lceil n/b \rceil$ updates). For each (b, sampler) pair we log the per-update test MSE (and, when comparing to full-batch, the distance to $\boldsymbol{\theta}^\star$).

For degrees $d \in \{3, 6, 9\}$ we build the polynomial design matrices $X_{\text{train}}, X_{\text{test}}$ (intercept included) using the same scaling pipeline as in Parts (a)–(e), and perform the same comparison. We report all curves against the number of *updates* (mini-batch steps). One epoch corresponds to

$$\text{updates per epoch} = \left\lceil \frac{n_{\text{train}}}{b} \right\rceil .$$

For full-batch baselines (Momentum, AdaGrad, RMSProp, Adam in batch mode) we call each optimiser with `max_iters=1` repeatedly, so that one recorded point equals one full-gradient update.

Unless stated otherwise we use:

- **Batch size:** $b = 32$; **seed:** 0 (NumPy `default_rng`).
- **Budget:** 1000 updates for full-batch methods; for mini-batch Adam we run enough epochs to exceed 1000 updates and then plot the first 1000.
- **Learning rates:** Momentum ($\eta = 2 \times 10^{-2}$, $\mu = 0.9$); AdaGrad ($\eta = 5 \times 10^{-2}$, $\delta = 10^{-8}$); RMSProp ($\eta = 10^{-2}$, $\rho = 0.9$, $\delta = 10^{-8}$); Adam ($\alpha = 5 \times 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$).

All runs start from $\boldsymbol{\theta}_0 = \mathbf{0}$.

Quantities recorded each update. (i) *Parameter error* $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}^\star\|_2$; and (ii) *Test loss* $\text{MSE}_{\text{test}}(t) = \frac{1}{n_{\text{test}}}\|y_{\text{test}} - X_{\text{test}}\boldsymbol{\theta}_t\|_2^2$. Early stopping is applied only if the *full-batch* training gradient norm falls below a tolerance (10^{-8}), otherwise we run the full update budget to make methods comparable.

Outputs. For each degree d we save two figures: *Convergence to $\boldsymbol{\theta}^\star$ vs updates* and *Test MSE vs updates* (e.g., `Figures/part f OLS convergence deg {d}.png` and `Figures/part f OLS MSE deg {d}.png`), and reference them in Sec. III.

For part (g) We sample $x \sim \text{Unif}[-1, 1]$ and generate noisy targets

$$y \;=\; f(x) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2),$$

with f the Runge function and $\sigma = 0.1$ unless otherwise stated. For each degree d we fit Ordinary Least Squares on the training split. For each d we compute

$$\text{MSE}_{\text{train}}(d) = \frac{1}{n_{\text{train}}}\|y_{\text{train}} - X_{d,\text{train}}\hat{\boldsymbol{\theta}}_d\|_2^2, \quad \text{MSE}_{\text{test}}(d) = \frac{1}{n_{\text{test}}}\|$$

We store these in arrays indexed by degree. We sweep degrees up to $d_{\max} \in [15, 30]$ to expose the variance–dominated regime. Because a single split can be noisy, We plot $\text{MSE}_{\text{train}}(d)$ and $\text{MSE}_{\text{test}}(d)$ versus d on the same axes (test as a solid line, train as a dashed line). We mark the degree minimising test MSE with a dot/annotation.

For Bias-Variance Trade-off analysis, the key steps mirror the definitions in the Methods section and reproduce Figure 2.11 of Hastie et al. [2] in spirit:

1. **Synthetic data.** Generate $n_{\text{points}} = 100$ inputs x uniformly on $[-1, 1]$ and targets from the Runge function with additive Gaussian noise:

$$y_i^{\text{true}} = f(x_i), \qquad y_i = f(x_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2), \; \sigma = 0.1.$$

2. **Train/test split and scaling.** Split (x, y) into training and test sets (70/30) using a fixed `random_state`. Standardise only the input using `StandardScaler` fitted on the training inputs; apply the same transform to the test inputs.

3. **Bootstrap loop.** For each degree d and for $b = 1, \dots, B$ with $B = 100$:

   (a) Sample training indices with replacement $I^{(b)} \in \{1, \dots, n_{\text{train}}\}^{n_{\text{train}}}$.

   (b) Fit *OLS* on the bootstrap resample via the normal equations (pseudoinverse):

$$\hat{\boldsymbol{\theta}}^{(d,b)} = \big((X_{\text{tr}}^{(d)}[I^{(b)}])^\top X_{\text{tr}}^{(d)}[I^{(b)}]\big)^\dagger (X_{\text{tr}}^{(d)}[I^{(b)}])^\top y_{\text{tr}}[I^{(b)}].$$

   (c) Predict on the *fixed* test design $X_{\text{te}}^{(d)}$ and store $\hat{\mathbf{y}}^{(d,b)} = X_{\text{te}}^{(d)} \hat{\boldsymbol{\theta}}^{(d,b)}$.

4. **Bias/variance/MSE aggregation on the test set.** Stack predictions column-wise $\hat{Y}^{(d)} = \big[\hat{\mathbf{y}}^{(d,1)} \ \cdots \ \hat{\mathbf{y}}^{(d,B)}\big] \in \mathbb{R}^{n_{\text{test}} \times B}$, then compute per-point moments

$$\mu_i^{(d)} = \frac{1}{B} \sum_{b=1}^{B} \hat{y}_i^{(d,b)}, \qquad \text{Var}_i^{(d)} = \frac{1}{B-1} \sum_{b=1}^{B} \big(\hat{y}_i^{(d,b)} - \mu_i^{(d)}\big)^2,$$

   and aggregate to the curves reported in the figures:

$$\text{Variance}(d) = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \text{Var}_i^{(d)}, \quad \text{Bias}^2_{\text{noisy}}(d) = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \big(\mu_i^{(d)} - y_i^{\text{test}}\big)^2$$

$$\text{MSE}_{\text{test}}(d) = \frac{1}{B} \sum_{b=1}^{B} \left[\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \big(y_i^{\text{test}} - \hat{y}_i^{(d,b)}\big)^2 \right].$$

   (Optionally, a "noise-free" bias can be obtained by replacing y_i^{test} with $f(x_i^{\text{test}})$.)

5. **Visualisation and reproducibility.** Plot MSE_{test}, $\text{Bias}^2_{\text{noisy}}$, and Variance versus degree d. We fix all RNG seeds (data generation, split, bootstrap indices) to ensure exact reproducibility. Increasing B smooths the curves at cost $\mathcal{O}(B)$ additional fits.

For part (h). we implement the bias–variance study with NumPy/`matplotlib` and `scikit-learn` utilities (`StandardScaler`, `KFold`). For each polynomial degree d, we build a design matrix $X = [\mathbf{1}, x, x^2, \dots, x^d]$ (intercept included), fit a model on refitted training subsets, *always* evaluate on the same held-out test set, and then aggregate across refits.
K-fold refitting (OLS). For each $d \in \{1, \dots, 23\}$ we run `KFold` with $K = 10$, `shuffle=True`, `random_state=0`. For each fold j: (i) fit a `StandardScaler` on fold-train inputs T_j; transform T_j and the global test inputs; (ii) build degree-d design matrices on the scaled inputs; (iii) fit OLS via a stable pseudo-inverse; (iv) predict on the fixed test set and store the column j of $Y_{\text{pred}}^{(d)}$.

K-fold refitting (Ridge). For each $d \in \{1, \dots, 20\}$ we use $K = 5$ and the same per-fold scaling as above, then fit Ridge with $\lambda = 0.5$ (intercept not penalised) on T_j, predict on the fixed test set, and fill $Y_{\text{pred}}^{(d)}$.

Bootstrap refitting (Ridge). For each $d \in \{1, \dots, 18\}$ we first construct X_{train} and X_{test} from pre-computed globally scaled inputs (`x_train_scaled`, `x_test_scaled`). Then over $B = 100$ bootstrap replicates we sample n_{train} rows of X_{train} with replacement, fit Ridge with $\lambda = 0.5$ (intercept not penalised), predict on X_{test}, and store the column in $Y_{\text{pred}}^{(d)}$.

K-fold refitting (LASSO via ISTA). For each $d \in \{1, \dots, 50\}$ we use $K = 5$. For each fold j we: (i) fit a `StandardScaler` on T_j inputs; transform T_j and test inputs; (ii) build raw degree-d designs $X_{\text{tr}}^{\text{raw}}, X_{\text{te}}^{\text{raw}}$; (iii) standardise columns $1{:}p{-}1$ of $X_{\text{tr}}^{\text{raw}}$ (mean/STD from T_j) and apply the same transform to $X_{\text{te}}^{\text{raw}}$ (leave the intercept unscaled), yielding $X_{\text{tr}}, X_{\text{te}}$; (iv) solve LASSO with ISTA (`max_iters`=10000, `tol`=10^{-8}, intercept excluded from ℓ_1 penalty) at $\lambda = 10^{-4}$, predict on the fixed test set, and populate $Y_{\text{pred}}^{(d)}$.

Bootstrap refitting (LASSO via ISTA). For each $d \in \{1, \dots, 20\}$ and $B = 200$ we: (i) draw a bootstrap index set of size n_{train}; (ii) fit a `StandardScaler` on the resampled inputs only; transform the resampled inputs and the fixed test inputs; (iii) build degree-d raw designs, then standardise columns $1{:}p{-}1$ using statistics from the resampled design (intercept left unscaled) for both train/test designs; (iv) run ISTA at $\lambda = 10^{-2}$ with the same stopping rules and no internal standardisation; (v) predict on the fixed test set and store the column in $Y_{\text{pred}}^{(d)}$.

Visualisation. For every degree d we plot $\text{MSE}_{\text{test}}(d)$, $\text{Bias}^2_{\text{noisy}}(d)$, and Variance$(d)$ versus d (one curve per quantity) using `matplotlib`. Titles include the model and, where relevant, the chosen λ.

C. Use of AI tools

ChatGPT was used to support code refinement, documentation, drafting and clarification of results.

III. RESULTS AND DISCUSSION

For Part (a), we applied OLS regression using polynomial features of x up to degree 15. We then evaluated the performance of the model using the score R^2 and the MSE, plotting MSE and R^2 against the polynomial degree, as shown in Figure 1.

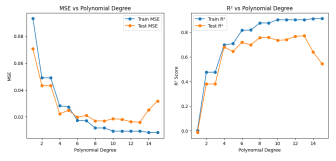

Figure 1: MSE vs Polynomial Degree, R^2 vs Polynomial Degree

Training MSE decreases steadily with increasing polynomial degree, since higher-degree polynomials fit the training data more closely. In contrast, test MSE decreases until around degrees 7–9, but rises again beyond degree 10. This suggests that the OLS model generalises well for degrees 6–9, underfits below degree 6, and overfits beyond degree 9.

The R^2 scores show a similar pattern: the training R^2 increases with degree and approaches 0.9, indicating an excellent fit in-sample. However, test R^2 plateaus after degree 9. Taken together, these results show that OLS regression performs best between degrees 7–9, where it strikes the right balance between bias and variance. Outside this range, the model either underfits or overfits.

In Part (b), we performed the same analysis for Ridge regression. From Figures 2 and 3, when $\lambda = 0$, Ridge regression produces results equivalent to OLS. For very small values of λ (e.g., 10^{-6}, 10^{-3}), the results remain similar to OLS, but with slightly greater stability at higher polynomial degrees. With moderate values of λ (e.g., 0.1, 1), Ridge regression reduces variance, leading to less overfitting and lower error compared to OLS. However, for large values of λ (e.g., 10, 100), underfitting occurs: R^2 decreases and test MSE increases. This behaviour illustrates the bias–variance trade-off: while OLS tends to overfit at high polynomial degrees, Ridge regression stabilises solutions by penalising large coefficients.

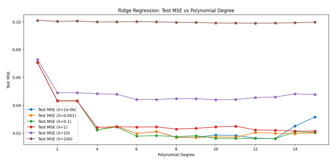

Figure 2: MSE vs Polynomial Degree for Ridge Regression

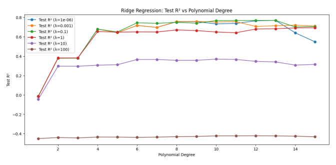

Figure 3: R^2 vs Polynomial Degree for Ridge Regression

In Part (c), we replaced the analytic OLS solution with our own gradient descent implementation using a fixed learning rate. The plots of MSE and R^2 against polynomial degree in Figure 4 show that the GD-based OLS reproduces the same overall trend observed in Part (a). As the polynomial degree increases, training MSE decreases while training R^2 increases, reflecting improved in-sample fit. However, test performance begins to deteriorate beyond degree 8, indicating overfitting. Convergence was achieved reliably up to approximately degree 8, after which gradient descent with $\gamma = 10^{-3}$ failed to converge within the iteration budget. This highlights the sensitivity of gradient descent to both learning rate and polynomial degree, as higher-degree design matrices amplify numerical instability and slow convergence.

Figure 5 illustrates the convergence behaviour for different learning rates at degree 10. For small γ values (e.g., 10^{-4}, 5×10^{-3}, 10^{-3}), the algorithm converges steadily, though at varying speeds. With $\gamma = 10^{-3}$, gradient descent converges efficiently to a solution comparable to the closed-form OLS, confirming that GD can approximate the analytic optimum when properly tuned. However, for large γ (e.g., 0.02), the updates overshoot and cause the cost function to diverge immediately to extremely large values, dominating the plot.

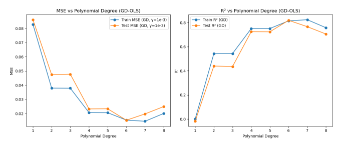

Figure 4: MSE and R$^2 ofGD - basedOLS$

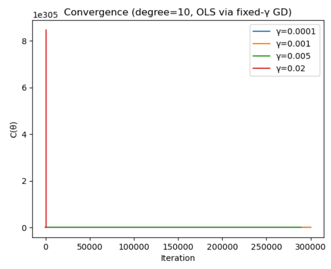

Figure 5: Convergence plot of GD-based OLS

These results demonstrate that while gradient descent provides a flexible, iterative alternative to the analytic OLS solution, its success depends critically on the choice of learning rate. Too small a γ leads to slow convergence, while too large a γ causes divergence. The instability at higher polynomial degrees further emphasises this sensitivity, motivating the use of more advanced optimisers with momentum or adaptive learning rates, which are explored in Part (d).

In part (d), at lower degree in Figures 6 and 7, Momentum decreases the parameter error more slowly but ultimately gets closest to the analytic optimum $\boldsymbol{\theta}^{\star}$; Adam reaches a similarly small parameter error but in fewer iterations. AdaGrad and RMSProp make the fastest early progress yet plateau farther from $\boldsymbol{\theta}^\star$. In terms of prediction, Adam and Momentum achieve the lowest test MSE.

insertols

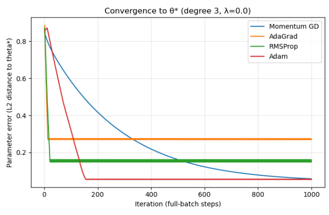

Figure 6: Convergence plot of OLS degree 3

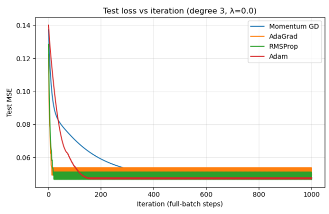

Figure 7: MSE plot of OLS degree 3

At higher degrees ($d = 6, 9$) as seen in Figures 8, 9, 10 and 11, this qualitative picture persists with one nuance: Adam can end farther from $\boldsymbol{\theta}^\star$ in parameter space while still delivering the best test MSE. This shows that distance to the training optimum is not a perfect proxy for test risk—adaptive steps act like an implicit preconditioner that emphasises the directions most relevant for prediction.

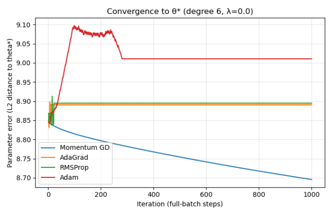

Figure 8: Convergence plot of OLS degree 6

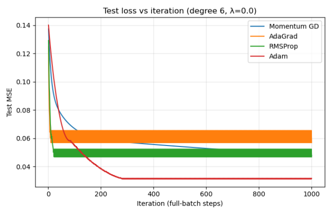

Figure 9: MSE plot of OLS degree 6

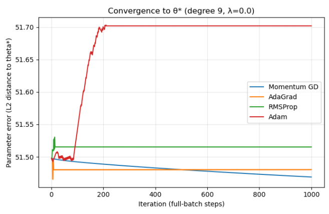

Figure 10: Convergence plot of OLS degree 9

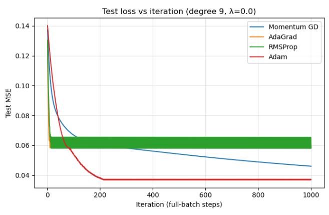

Figure 11: MSE plot of OLS degree 9

With a small L_2 penalty, all methods benefit from improved conditioning: parameter trajectories are smoother and the spread between methods narrows. From Figures 12, 13, 14, 15, 16 and 17, we observe the same overall ranking, but now Adam consistently converges to $\boldsymbol{\theta}^\star$ even at higher degrees, while also maintaining the best (lowest) test MSE. Momentum remains competitive and reaches a near-optimal parameter error, whereas AdaGrad and RMSProp still descend rapidly at the start but stabilise farther from $\boldsymbol{\theta}^\star$ and yield slightly higher test error. Overall, Ridge stabilisation reduces optimisation sensitivity and sharpens the alignment between parameter convergence and generalisation, with Adam offering the best balance of speed, robustness, and test performance.

insertridge

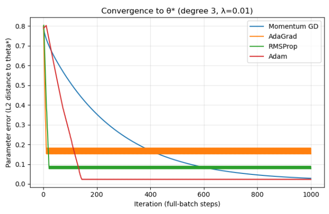

Figure 12: Convergence plot of Ridge degree 3

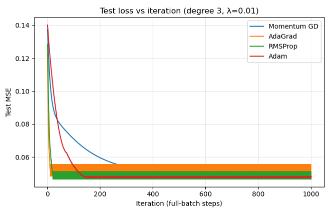

Figure 13: MSE plot of Ridge degree 3

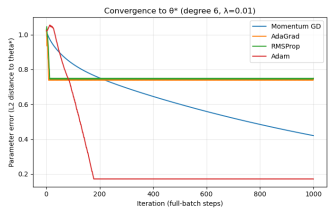

Figure 14: Convergence plot of Ridge degree 6

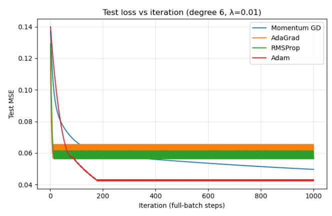

Figure 15: MSE plot of Ridge degree 6

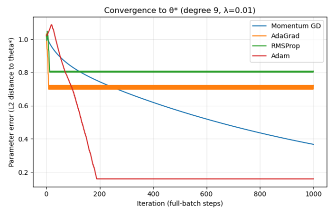

Figure 16: Convergence plot of Ridge degree 9

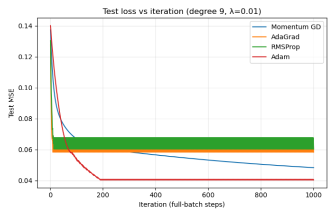

Figure 17: MSE plot of Ridge degree 9

For part (e), the heatmaps (Figures 18–19) show that our ISTA implementation closely matches scikit-learn across degrees and regularisation strengths. For small λ, LASSO behaves near OLS with low Test MSE and high R^2. As λ increases, coefficients are progressively shrunk and many are set exactly to zero, yielding sparser models; beyond a critical λ the models underfit, which appears as increased Test MSE and sharply reduced (often negative) R^2.

Across degrees, performance typically improves from very low degrees to a moderate range, then degrades at high degrees due to variance and ill-conditioning; stronger λ partly counteracts this but eventually over-regularises. Minor discrepancies between ISTA and scikit-learn at large λ or high degree are attributable to solver tolerances and step-size selection, not modelling differences. Overall, these results validate the proximal gradient approach and highlight LASSO's key distinction from Ridge (Part (b)): beyond shrinkage, LASSO performs *feature selection* by driving coefficients exactly to zero.

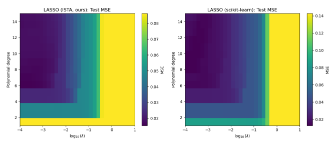

Figure 18: Heatmaps of **Test MSE** for LASSO across polynomial degree (y-axis) and $\log_{10}(\lambda)$ (x-axis). Left: ISTA; Right: scikit-learn. Both methods achieve low error for small λ and increasingly underfit as λ grows.

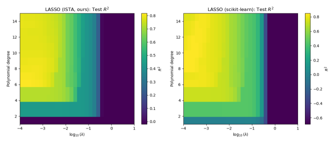

Figure 19: Heatmaps of **Test** R^2 for LASSO across polynomial degree (y-axis) and $\log_{10}(\lambda)$ (x-axis). Left: ISTA; Right: scikit-learn. High R^2 appears at small λ and collapses at large λ due to over-regularisation; ISTA closely tracks scikit-learn across the grid.

For part (f), Figure 20 shows the result for Stochastic Gradient Descent with ADAM and number of epochs = 20, iterating over the number of Polynomial Degrees. The results are rather similar to plain gradient descent, where at high degrees the test error increases significantly. Furthermore, SGD V2 seems to perform better than SGD V1. This is because in SGD V1 we sample batches with replacement and the same data can be seen multiple times in an epoch while others are skipped. As a result, the mini-batch gradients of SGD V2 have lower variance, and the coverage is balanced, leading to better convergence.

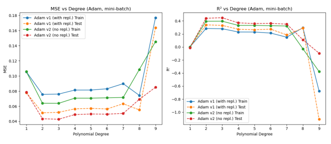

Figure 20: SGD with ADAM set at 20 epochs

Figure 21 shows the same gradient descent but with a larger number of epochs. With a higher number of epochs, there are more parameter updates and we are closer to the OLS minimizer, and we can see that the difference in test MSE is much lesser between SGD V1 and V2

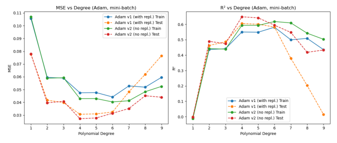

Figure 21: SGD with ADAM set larger than 20 epochs

Figure 22 shows the results for Stochastic Gradient Descent with a fixed learning rate, iterating over Batch Size with a fixed polynomial degree. We do not fix the number of epochs since for larger batch sizes, the parameters is updated less for a fixed number of epochs. So we increase the number of epochs as the batch size increases. The test MSE improves as the number of batch size increases initially, before it worsens. This is because at small batch sizes, there is too much noise under training. While at larger batch sizes, we lose the stochastic regularization.

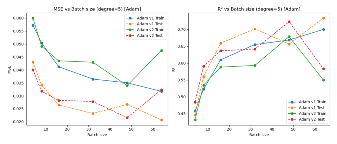

Figure 22: MSE plot of Ridge degree 9SGD with ADAM at fixed learning rate

From the figures, For both OLS (Figures 23, 24, 25, 26, 27 and 28) and Ridge (Figures 29, 30, 31, 32, 33 and 34), Stochastic Gradient Descent converges the best to the optimal Parameters for the Runge function for all degrees, at the same time having the lowest Test MSE. Minibatch V1 outperforms mini-batch V2 and converges faster.

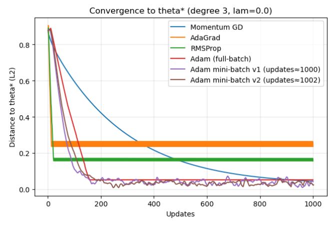

Figure 23: Convergence plot of OLS degree 3

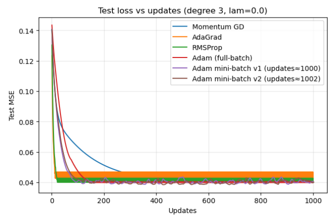

Figure 24: MSE plot of OLS degree 3

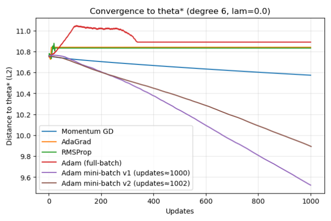

Figure 25: Convergence plot of OLS degree 6

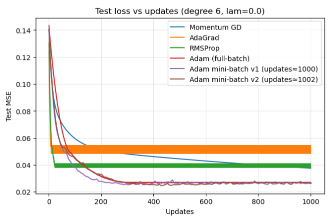

Figure 26: MSE plot of OLS degree 6

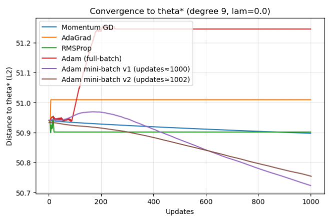

Figure 27: Convergence plot of OLS degree 9

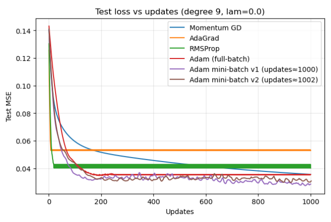

Figure 28: MSE plot of OLS degree 9

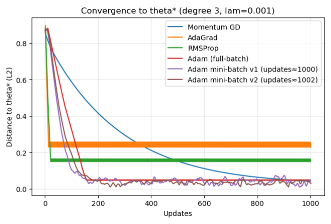

Figure 29: Convergence plot of Ridge degree 3

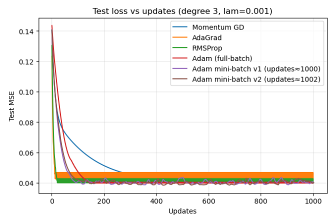

Figure 30: MSE plot of Ridge degree 3

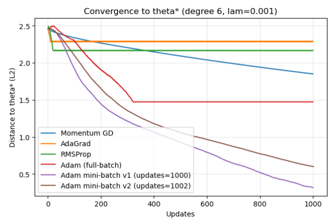

Figure 31: Convergence plot of Ridge degree 6

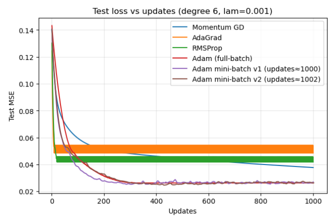

Figure 32: MSE plot of Ridge degree 6

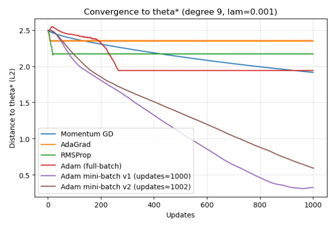

Figure 33: Convergence plot of Ridge degree 9

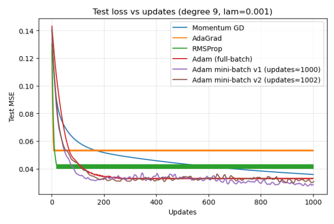

Figure 34: MSE plot of Ridge degree 9

In part (g), Figure 35 below imitates Fig. 2.11 of Hastie, Tibshirani, and Friedman. It is a U shaped curve representing high test error at low numbers of polynomial degree, due to high bias. The low variance at low polynomial degrees is seen by the Train and Test MSE being close. At high polynomial degrees, There is high variance resulting in higher Test MSE, although Train MSE continues decreasing. The Test MSE is lowest at some intermediate polynomial degree where there is low variance and low bias.

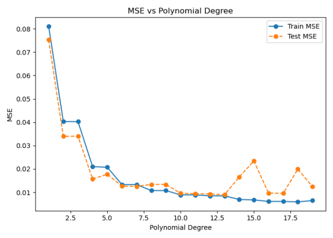

Figure 35: MSE plot against polynomial degree. Imitation of Fig. 2.11 of Hastie, Tibshirani, and Friedman

Figure 36 shows the Bias Variance Trade-off for the Runge function using Bootstrap as a resampling method. We lower the maximum degree to 12 for easier analysis. At low degrees, the model is too rigid to learn Runge's Curvature and underfits, with a large bias that dominates the Test MSE. At the same time, variance is small since different bootstrap fits look alike. As the model complexity increases, the model captures Runge's curvature better and bias decreases. At the same time, variance increases slightly and the Test error decreases. This intermediate complexity is where Test MSE is the lowest. At higher degrees, The model starts overfitting, and different bootstrap fits differ largely. As a result, while bias is low, variance increases and domaintes the test MSE, which increases.

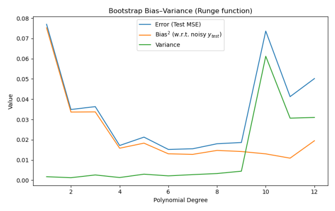

Figure 36: Bootstrap Bias Variance Trade-off for Runge function

In part (g), Figure 37 above shows the Bias Variance Tradeoff using KfoldCV as a resampling method. The results are rather similar, but The Variance in K-foldCV tends to be more stable and only blows up at much higher polynomial degrees. The reason for this is because K-fold trains each model on more distinct data points as compared to bootstrap, and with more unique data for each fit, the model fluctuates less leading to lower variance.

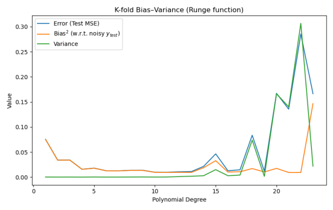

Figure 37: Kfold Bias Variance Trade-off for OLS

Performing Bias Variane Trade-off analysis using Ridge, we get a rather similar result (Figures 38 and 39). However, because Ridge shrinks coefficient towards 0, predictions fluctuate much less and the variance rises more slowly. With Lasso Regression, the same can be observed (Figures 40 and 41), except that the Variance blows up at much higher degrees compared to Ridge. This is largely because Lasso zeroes out many higher order terms. As a result, we also see that the bias increases for Lasso as well,

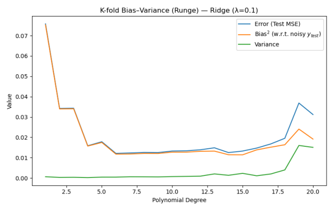

Figure 38: Kfold Bias Variance for OLS

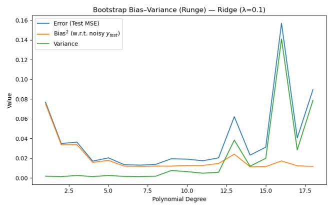

Figure 39: Bootstrap Bias Variance for OLS

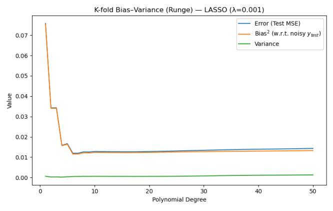

Figure 40: Kfold Bias Variance for Lasso

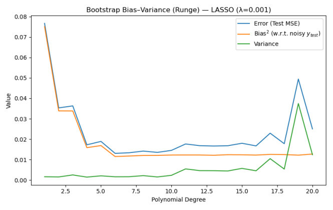

Figure 41: Bootstrap Bias Variance for Lasso

Part (g) The following is a derivation of the bias-variance decomposition for the expected squared error between the true value y and its estimator $\tilde{y}$.

$$
\begin{aligned}
E[(y-\tilde{y})^2] &= E[(y-f(x)+f(x)-\tilde{y})^2] \\
&= E[(y-f(x))^2+(f(x)-\tilde{y})^2 \\
&\quad +2(y-f(x))(f(x)-\tilde{y})] \\
&= E[(y-f(x))^2]+E[(f(x)-\tilde{y})^2] \\
&\quad +2E[(y-f(x))(f(x)-\tilde{y})] \\
&= E[(y-f(x))^2]+E[(f(x)-\tilde{y})^2]+0
\end{aligned}
$$

Assuming the error term $\epsilon = y - f(x)$ has zero mean and is independent of the model's prediction error, the last term vanishes. Now, focusing on the second term:

$$
\begin{aligned}
E[(f(x)-\tilde{y})^2] &= E[(\,f(x)-E(\tilde{y})+E(\tilde{y})-\tilde{y}\,)^2] \\
&= E[(f(x)-E(\tilde{y}))^2+(E(\tilde{y})-\tilde{y})^2 \\
&\quad +2(f(x)-E(\tilde{y}))(E(\tilde{y})-\tilde{y})] \\
&= [f(x)-E(\tilde{y})]^2+E[(E(\tilde{y})-\tilde{y})^2] \\
&\quad +2(f(x)-E(\tilde{y}))E[E(\tilde{y})-\tilde{y}] \\
&= [f(x)-E(\tilde{y})]^2+E[(\tilde{y}-E(\tilde{y}))^2]+0
\end{aligned}
$$

The cross-term is zero because $E[E(\tilde{y})-\tilde{y}] = E(\tilde{y}) - E(\tilde{y}) = 0$. Recognizing the definitions of bias and variance, we substitute back:

$$
\begin{aligned}
\therefore\ E[(y-\tilde{y})^2] &= E[(y-f(x))^2]+[f(x)-E(\tilde{y})]^2 \\
&\quad +E[(\tilde{y}-E(\tilde{y}))^2] \\
&= \sigma^2+\mathrm{Bias}(\tilde{y})^2+\mathrm{Var}(\tilde{y})
\end{aligned}
$$

Where $\sigma^2 = E[(y-f(x))^2]$ is the irreducible error.

IV. CONCLUSION

This project examined polynomial regression on the Runge function using *OLS*, *Ridge*, and *LASSO*, trained with both full-batch and stochastic optimisers, and analysed generalisation through Fig. 2.11–style curves and bias–variance decompositions (via bootstrap and K-fold refitting).

Main findings.

- **Model complexity.** Low degrees underfit with high bias and low variance, while high degrees overfit with low bias and high variance. The best generalisation occurs at some intermediate degree.
- **Effect of L_2 regularisation (Ridge).** Small to moderate λ improves conditioning and reduces variance with the shrinking of coefficients, stabilising high-degree fits and lowering test error relative to OLS; overly large λ induces underfitting (bias dominates).
- **Effect of L_1 regularisation (LASSO).** LASSO adds sparsity. At small λ it behaves similarly to OLS/Ridge; increasing λ trades variance for bias and can underfit. On this smooth, dense target, LASSO mainly offers interpretability (feature selection) rather than accuracy gains.
- **Optimisers (full-batch).** Momentum and Adam achieve the strongest test performance; AdaGrad/RMSProp often make the fastest early progress but plateau farther from the optimum. Notably, Adam can sit *farther* from the closed-form $\boldsymbol{\theta}^\star$ while still yielding the *lowest* test MSE, showing that distance to the training optimum is not a reliable surrogate for generalisation.
- **Stochastic/mini-batch training.** Mini-batch Adam with no-replacement sampling per epoch (v2) produces smoother progress within an epoch than with-replacement sampling (v1) and it also achieves the strongest test performance and convergence speed. A batch-size sweep reveals a "sweet spot" (e.g. $b \approx 32$–64 in our runs): very small batches inject too much gradient noise (higher test MSE), while very large batches lose the beneficial stochastic regularisation. With more epochs, the performance gap between v1 and v2 narrows.
- **Bias–variance via resampling.** Bootstrap and K-fold refitting reproduce the canonical trade-off: bias decreases and variance increases with degree. K-fold aggregates tend to be smoother/more stable than bootstrap at the same compute budget, while regularisation (Ridge/LASSO) delays the variance growth and flattens the test-error curve.

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Journal of Machine Learning Research **12**, 2825 (2011), URL `http://jmlr.org/papers/v12/pedregosa11a.html`.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics (Springer, New York, NY, 2009), 2nd ed., URL `https://link.springer.com/book/10.1007/978-0-387-84858-7`.