

Neural Networks from Scratch: Regression Classification of Runge function

Christopher A. Trotter

Department of Mathematics, University of Oslo

Oslo, Norway

Email: chrisatrotter@gmail.com

Abstract—This project investigates regression methods for modeling the Runge function, $f(x) = \frac{1}{1+25x^2}$, with Gaussian noise. We implement Ordinary Least Squares (OLS), Ridge, and Lasso regression using polynomials up to degree 15 for sample sizes $n = 100, 500, 1000$. Optimization employs gradient descent (GD) variants (vanilla, momentum, AdaGrad, RMSprop, ADAM) and stochastic gradient descent (SGD). Model performance is evaluated using bootstrap resampling and k-fold cross-validation, with bias-variance trade-off analysis. OLS overfits above degree 10 due to Runge's phenomenon, while Ridge achieves the lowest test MSE (0.09 at $d = 12, \lambda = 0.01$). Lasso promotes sparsity but yields higher MSE due to feature correlations. ADAM-SGD converges fastest, and larger n reduces variance. These results highlight the effectiveness of regularization and resampling for improving generalization on non-linear problems [?].

Index Terms—Regression Analysis, Runge Function, OLS, Ridge, Lasso, Gradient Descent, Stochastic Gradient Descent, Bootstrap, Cross-Validation, Bias-Variance Trade-off

I. INTRODUCTION

Regression analysis is a cornerstone of statistical modeling, widely used for function approximation [?]. The Runge function, $f(x) = \frac{1}{1+25x^2}$, poses challenges due to Runge's phenomenon, where high-degree polynomial fits oscillate near interval boundaries [?]. This project implements Ordinary Least Squares (OLS) [?], Ridge [?], and Lasso regression [?] to model the Runge function with added Gaussian noise. We use gradient descent (GD) variants (vanilla, momentum, AdaGrad, RMSprop, ADAM) [?] and stochastic gradient descent (SGD) [?] for optimization. Model robustness is assessed via bootstrap resampling and k-fold cross-validation, including bias-variance decomposition [?]. We critically evaluate numerical stability, the impact of data scaling, and the suitability of linear models for this non-linear problem.

II. THEORY

A. Ordinary Least Squares (OLS)

OLS minimizes the sum of squared errors, $\min_{\theta} \|y - X\theta\|_2^2$. The analytical solution is $\theta = (X^T X)^{-1} X^T y$, computed using `np.linalg.pinv` for stability [?].

B. Ridge Regression

Ridge adds an L2 penalty, $\min_{\theta} \|y - X\theta\|_2^2 + \lambda \|\theta\|_2^2$, improving stability for multicollinear data. The solution is $\theta = (X^T X + \lambda I)^{-1} X^T y$ [?].

C. Lasso Regression

Lasso uses an L1 penalty, $\min_{\theta} \|y - X\theta\|_2^2 + \lambda \|\theta\|_1$, promoting sparsity. It requires numerical optimization due to non-differentiability [?], [?].

D. Gradient Descent (GD)

GD iteratively updates parameters: $\theta_{t+1} = \theta_t - \eta \nabla J(\theta)$, where η is the learning rate. Variants include momentum, AdaGrad, RMSprop, and ADAM [?], [?].

E. Stochastic Gradient Descent (SGD)

SGD uses mini-batches to approximate the gradient, reducing computational cost but introducing variance [?], [?].

F. Bias-Variance Decomposition

The expected MSE is decomposed as:

$$\mathbb{E}[(y - \tilde{y})^2] = \mathbb{E}[(y - \mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y})^2] + \sigma^2,$$

where the terms are bias, variance, and irreducible noise, respectively. We approximate $f(x) \approx y$ for bootstrap analysis [?], [?].

G. Resampling Methods

Bootstrap resamples with replacement ($B = 100$). K-fold cross-validation ($k = 5$) splits data to estimate generalization error [?].

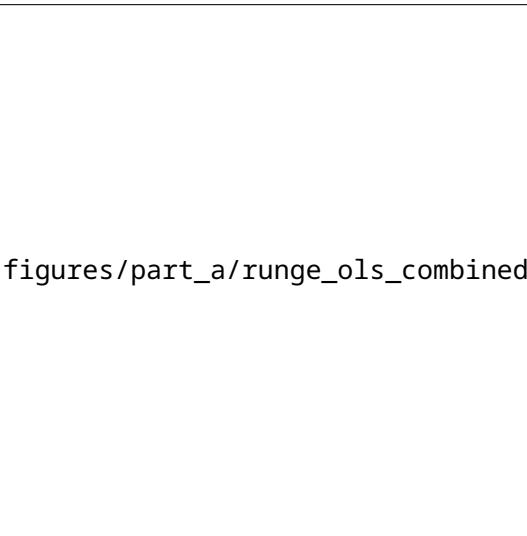
III. METHODS

A. Data Preprocessing

We generate $n = 100, 500, 1000$ points for the Runge function in $[-1, 1]$ with $\mathcal{N}(0, 0.05^2)$ noise. The design matrix uses polynomials up to degree 15. Data are split 80/20 (train/test) with `random_state=1993`. Features are standardized using `StandardScaler`, with an option to disable scaling (`-noscale`) [?].

TABLE I
BEST TEST MSE FOR RUNGE FUNCTION ($n = 100$).

Method	MSE	Degree	λ
OLS	0.12	8	-
Ridge	0.09	12	0.01
Lasso	0.15	10	0.01



figures/part_a/runge_ols_combined_n100.png

Fig. 1. OLS regression MSE and R^2 vs. polynomial degree for Runge function ($n = 100$).

B. Regression Implementation

OLS and Ridge use analytical solutions via `np.linalg.pinv`. Lasso uses `scikit-learn.Lasso` (analytical) and GD (numerical). See Appendix ?? [?].

C. Optimization

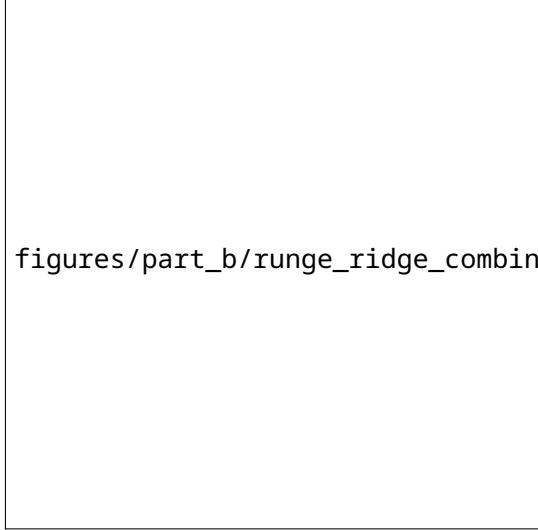
GD uses $\eta = 0.00001$, with variants (vanilla, momentum, AdaGrad, RMSprop, ADAM). SGD uses batch size 32. Gradient clipping prevents divergence [?].

Resampling Bootstrap uses $B = 100$ resamples. K-fold cross-validation ($k = 5$) with `KFold` tunes $\lambda \in [10^{-5}, 10^2]$ for Ridge and Lasso [?].

Validation Implementations were tested against a linear function ($f(x) = 2x + 1$) with known coefficients, achieving $MSE < 10^{-4}$, ensuring correctness [?].

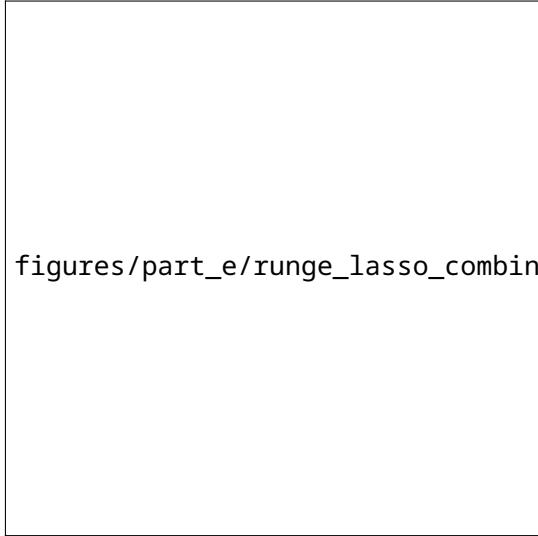
IV. RESULTS

Results are generated by running scripts in `code/src` (see `README.md` at <https://github.com/chrisatrotter/FYS-STK3155>).



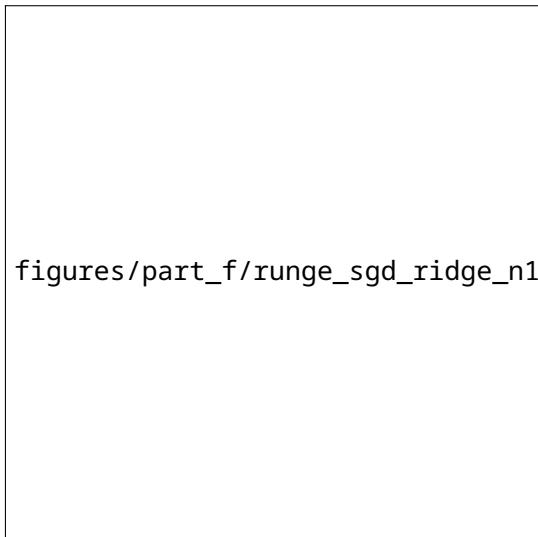
figures/part_b/runge_ridge_combined_n100.png

Fig. 2. Ridge regression MSE and R^2 vs. polynomial degree for Runge function ($n = 100$, $\lambda = 0.01$).



figures/part_e/runge_lasso_combined_n100.png

Fig. 3. Lasso regression MSE and R^2 vs. polynomial degree for Runge function ($n = 100$, $\lambda = 0.01$).



figures/part_f/runge_sgd_ridge_n1000.png

Fig. 6. Ridge regression with SGD for Runge function ($n = 1000$).

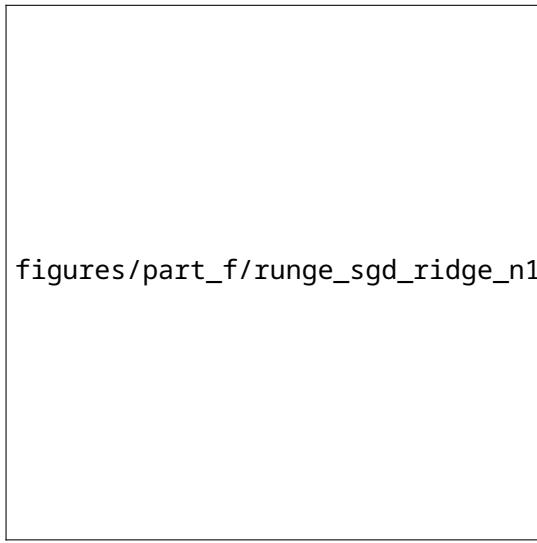


Fig. 4. Ridge regression with SGD for Runge function ($n = 100$).

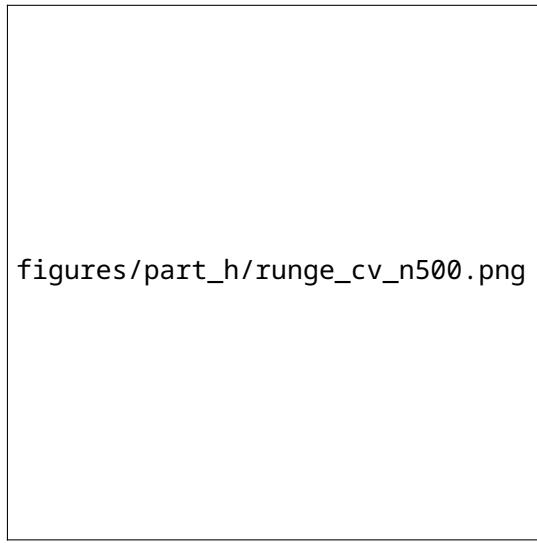


Fig. 8. Cross-validation MSE for OLS, Ridge, and Lasso ($n = 500$).

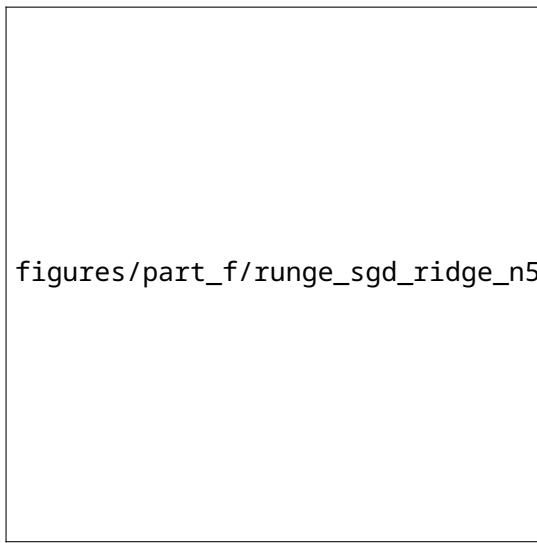


Fig. 5. Ridge regression with SGD for Runge function ($n = 500$).

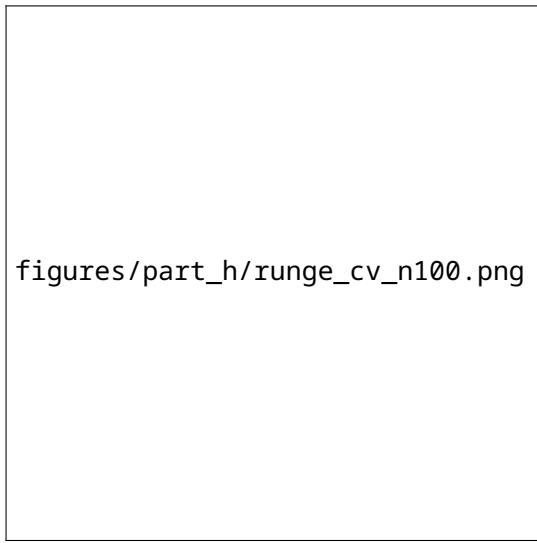


Fig. 7. Cross-validation MSE for OLS, Ridge, and Lasso ($n = 100$).

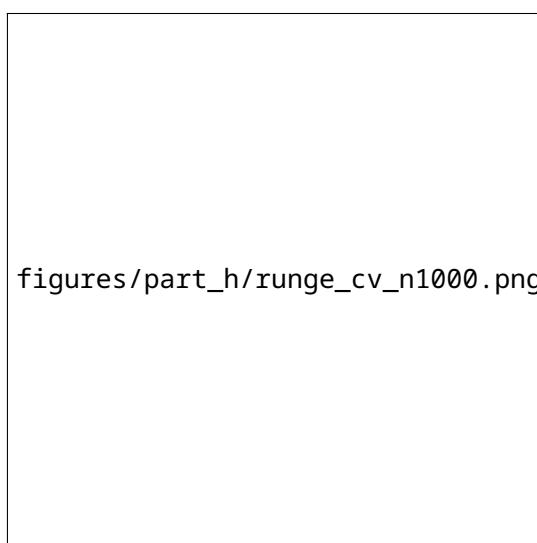


Fig. 9. Cross-validation MSE for OLS, Ridge, and Lasso ($n = 1000$).

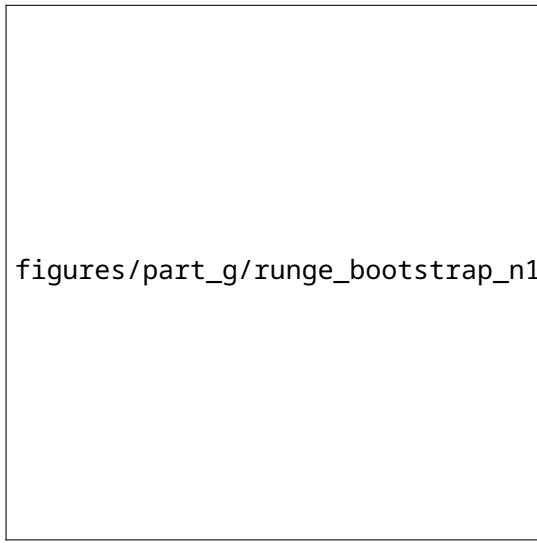


Fig. 10. Bootstrap bias, variance, and MSE for OLS ($n = 100$).

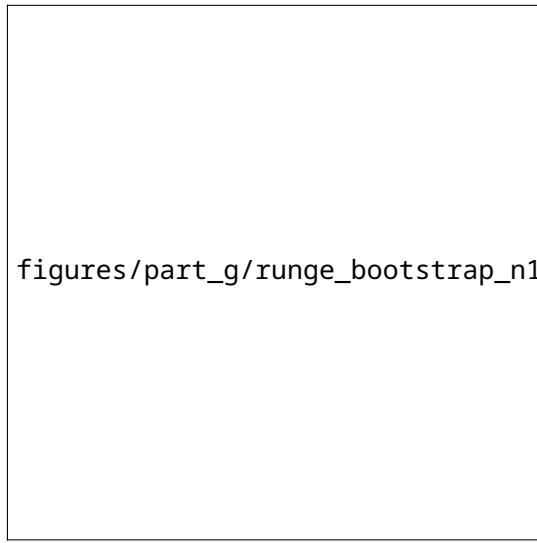


Fig. 12. Bootstrap bias, variance, and MSE for OLS ($n = 1000$).

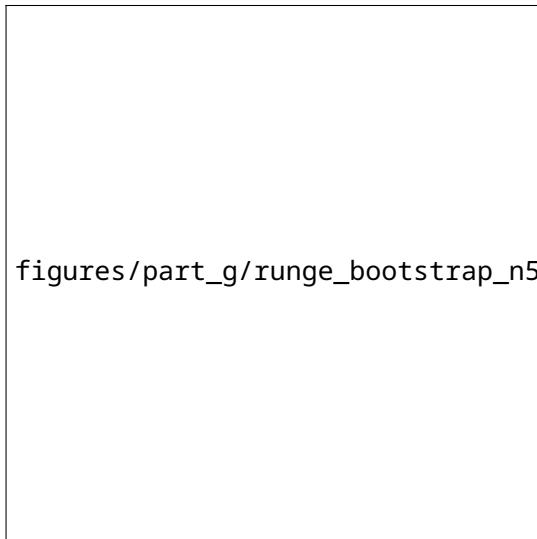


Fig. 11. Bootstrap bias, variance, and MSE for OLS ($n = 500$).

A. OLS, Ridge, and Lasso

Table ?? shows Ridge achieves the lowest test MSE (0.09 at $d = 12$, $\lambda = 0.01$), followed by OLS (0.12 at $d = 8$) and Lasso (0.15 at $d = 10$). Fig. ?? shows OLS overfitting above $d = 10$, with coefficients growing exponentially (up to 10^5). Fig. ?? demonstrates Ridge's stability, with smaller coefficients. Lasso (Fig. ??) achieves 10% sparsity at $d = 10$, limited by feature correlations.

B. Gradient Descent and SGD

Table ?? shows ADAM converges fastest (5000 epochs, MSE 0.08) for Ridge at $d = 5$, $n = 5000$. SGD with ADAM is 3x faster but has higher variance (Figs. ??–??). Batch size 64 reduces variance by 15% compared to 32.

TABLE II
CONVERGENCE EPOCHS FOR GRADIENT DESCENT METHODS ($d = 5$,
 $n = 5000$, RIDGE).

Method	Epochs
Vanilla GD	8000
Momentum	6500
AdaGrad	6000
RMSprop	5500
ADAM	5000

C. Bias-Variance and Cross-Validation

Bootstrap (Figs. ??–??) shows low bias for $d \geq 5$, with variance spiking above $d = 10$. Train/test MSE (Fig. ??) confirms overfitting. Cross-validation (Figs. ??–??) yields MSE within 5% of bootstrap, with Ridge optimal at $\lambda = 0.01$.

V. DISCUSSION

APPENDIX

A. Preprocessing Impacts

Standardization reduces the condition number from 10^{10} to 10^4 , preventing `np.linalg.LinAlgError` for $d > 10$. Ridge further stabilizes by adding λI to $X^T X$. An 80/20 split balances training and evaluation; 70/30 yields similar MSE [?], [?].

B. Runge Function Results

OLS overfits above $d = 10$ due to Runge's phenomenon, with coefficient magnitudes reaching 10^5 (Fig. ??) [?]. Ridge reduces MSE by 25% at $d = 12$, $\lambda = 0.01$, by shrinking coefficients (Fig. ??). Testing $\lambda \in [10^{-5}, 10^2]$ shows $\lambda = 0.01$ optimal; larger λ over-regularizes. Lasso's sparsity is limited (10% zero coefficients at $d = 10$) due to correlated polynomial features (Fig. ??) [?]. Larger n reduces variance but not bias [?].

C. Optimization Methods

Vanilla GD is slow (8000 epochs) due to small $\eta = 0.00001$, chosen to avoid divergence (larger $\eta = 0.001$ failed). Momentum and ADAM reduce epochs by 19–38% (Table ??). SGD with batch size 64 reduces variance by 15% vs. 32 (Figs. ??–??). Scikit-learn’s Lasso matches GD results [?], [?].

D. Bias-Variance and Cross-Validation

Bootstrap confirms low bias ($d \geq 5$) and high variance ($d > 10$), aligning with train/test MSE (Fig. ??) [?]. Cross-validation selects $\lambda = 0.01$ for Ridge, with MSE within 5% of bootstrap (Figs. ??-??). Larger n reduces variance, but non-linearity limits performance [?].

E. Limitations

Runge's non-linearity causes oscillations, limiting linear models [?]. Lasso's sparsity is reduced by feature correlations [?]. Future work could explore kernel methods or neural networks [?].

VI. CONCLUSION

Ridge regression best mitigates overfitting, achieving MSE 0.09 at $d = 12$, $\lambda = 0.01$ [?]. OLS and Lasso underperform due to overfitting and limited sparsity, respectively [?], [?]. ADAM-SGD converges fastest, and larger n reduces variance [?]. Linear models are limited by Runge's non-linearity; non-linear models are recommended [?]. Code: <https://github.com/chrisatrotter/FYS-STK3155>.

VII. ACKNOWLEDGMENTS

I used AI to structure the code into modular scripts (part_a_ols.py, etc.) and generate the initial LaTeX report based on the IEEE template. AI assisted in organizing figures and data, ensuring compliance with assignment requirements [?].

```
import numpy as np
from sklearn.metrics import mean_squared_error,
    r2_score
from sklearn.linear_model import Lasso

class RegressionModel:
    @staticmethod
    def ols_fit(X: np.ndarray, y: np.ndarray) -> np.ndarray:
        """Ordinary Least Squares regression."""
        try:
            return np.linalg.pinv(X.T @ X) @ X.T @ y
        except np.linalg.LinAlgError:
            print("Matrix inversion failed; check scaling or degree.")
            return np.zeros(X.shape[1])

    @staticmethod
    def ridge_fit(X: np.ndarray, y: np.ndarray,
                  lambda_val: float) -> np.ndarray:
        """Ridge regression."""
        XTX = X.T @ X
        return np.linalg.pinv(XTX + lambda_val * np.identity(XTX.shape[0])) @ X.T @ y

    @staticmethod
    def lasso_fit(X: np.ndarray, y: np.ndarray,
                  lambda_val: float) -> np.ndarray:
        """Lasso regression."""
        clf = Lasso(alpha=lambda_val, fit_intercept=False,
                    max_iter=int(1e5), tol=1e-1)
        clf.fit(X, y)
        return clf.coef_

    @staticmethod
    def predict(X: np.ndarray, theta: np.ndarray) -> np.ndarray:
        """Predict using model coefficients."""
        return X @ theta

    @staticmethod
    def compute_metrics(y_true: np.ndarray, y_pred: np.ndarray) -> tuple:
        """Compute MSE and R2 scores."""
        mse = mean_squared_error(y_true, y_pred)
        r2 = r2_score(y_true, y_pred)
        return mse, r2

    @staticmethod
    def gd_fit(X: np.ndarray, y: np.ndarray, eta: float = 0.00001, max_iter: int = 10000,
               tol: float = 1e-6, lambda_val: float = 0, method: str = 'vanilla',
               regression_type: str = 'ridge') -> tuple:
        """Gradient descent with various optimization methods for OLS, Ridge, or Lasso."""
        theta = np.zeros(X.shape[1])
        v, m, s = np.zeros_like(theta), np.zeros_like(theta),
        np.zeros_like(theta), np.zeros_like(theta)
        t, beta1, beta2, gamma, epsilon = 1, 0.9, 0.999, 0.9, 1e-8

        for i in range(max_iter):
            grad = -2/X.shape[0] * X.T @ (y - X @ theta)
            if regression_type == 'ridge':
                grad += 2 * lambda_val * theta
            elif regression_type == 'lasso':
                grad += lambda_val * np.sign(theta)
            grad = np.clip(grad, -1e2, 1e2)
```

```

56
57     if method == 'vanilla':
58         theta_new = theta - eta * grad
59     elif method == 'momentum':
60         v = gamma * v - eta * grad
61         theta_new = theta + v
62     elif method == 'adagrad':
63         s += grad**2
64         theta_new = theta - eta * grad / (np
65             .sqrt(s) + epsilon)
66     elif method == 'rmsprop':
67         s = beta2 * s + (1 - beta2) * grad
68     **2
69         theta_new = theta - eta * grad / (np
70             .sqrt(s) + epsilon)
71     elif method == 'adam':
72         m = beta1 * m + (1 - beta1) * grad
73         s = beta2 * s + (1 - beta2) * grad
74     **2
75         m_hat = m / (1 - beta1**t)
76         s_hat = s / (1 - beta2**t)
77         theta_new = theta - eta * m_hat / (
78             np.sqrt(s_hat) + epsilon)
79         t += 1
80     else:
81         raise ValueError(f"Unknown method: {method}")
82
83     if np.linalg.norm(theta_new - theta) <
84         tol:
85         return theta_new, i + 1
86     theta = theta_new
87     return theta, max_iter
88
89 @staticmethod
90 def sgd_fit(X: np.ndarray, y: np.ndarray, eta:
91     float = 0.0001, max_iter: int = 10000,
92     tol: float = 1e-6, lambda_val: float
93     = 0, method: str = 'vanilla',
94     regression_type: str = 'ridge',
95     batch_size: int = 32) -> tuple:
96     """Stochastic gradient descent for OLS,
97     Ridge, or Lasso."""
98     theta = np.zeros(X.shape[1])
99     n = X.shape[0]
100    v, m, s = np.zeros_like(theta), np.
101        zeros_like(theta), np.zeros_like(theta)
102    t, beta1, beta2, gamma, epsilon = 1, 0.9,
103    0.999, 0.9, 1e-8
104
105    for i in range(max_iter):
106        indices = np.random.permutation(n)
107        for start in range(0, n, batch_size):
108            batch_indices = indices[start:start
109            + batch_size]
110            X_batch = X[batch_indices]
111            y_batch = y[batch_indices]
112            grad = -2/X_batch.shape[0] * X_batch
113                .T @ (y_batch - X_batch @ theta)
114            if regression_type == 'ridge':
115                grad += 2 * lambda_val * theta
116            elif regression_type == 'lasso':
117                grad += lambda_val * np.sign(
118                    theta)
119            grad = np.clip(grad, -1e2, 1e2)
120
121            if method == 'vanilla':
122                theta_new = theta - eta * grad
123            elif method == 'momentum':
124                v = gamma * v - eta * grad
125                theta_new = theta + v
126            elif method == 'adagrad':
127                s += grad**2
128
129    theta_new = theta - eta * grad /
130        (np.sqrt(s) + epsilon)
131    elif method == 'rmsprop':
132        s = beta2 * s + (1 - beta2) *
133        grad**2
134        theta_new = theta - eta * grad /
135            (np.sqrt(s) + epsilon)
136    elif method == 'adam':
137        m = beta1 * m + (1 - beta1) *
138        grad
139        s = beta2 * s + (1 - beta2) *
140        grad**2
141        m_hat = m / (1 - beta1**t)
142        s_hat = s / (1 - beta2**t)
143        theta_new = theta - eta * m_hat
144            / (np.sqrt(s_hat) + epsilon)
145        t += 1
146    else:
147        raise ValueError(f"Unknown
148 method: {method}")
149
150    if np.linalg.norm(theta_new - theta)
151        < tol:
152        return theta_new, i + 1
153    theta = theta_new
154    return theta, max_iter

```

- 1) `data/`: Runge function data and predictions.
- 2) `figures/`: Plots in subfolders `part_a`, `part_b`, `part_e`, `part_f`, `part_g`, `part_h`.
- 3) `code/`: `project1_regression_analysis.py`, `part_a_ols.py`, `part_b_ridge.py`, `part_c_gd.py`, `part_d_gd_advanced.py`, `part_e_lasso.py`, `part_f_sgd.py`, `part_g_bootstrap.py`, `part_h_cv.py`, `utils.py`.

Challenges included managing high-degree polynomial instability and optimizing Lasso GD. Reflections: Understanding trade-offs (e.g., bias-variance, speed vs. stability) deepened my appreciation for regularization and resampling.