AERSP 497 HW1 - Multivariate Linear Regression

Vivek Mathew

1 Introduction

This report outlines the approach and findings from implementing a multivariate linear regression method to extract the underlying dynamics of a system based on observed data. The goal was to build a model that could learn the relationship between a system's state and its time derivative using regression-based techniques.

The dataset used in this assignment came from the **Lorenz system**, a well-known system of nonlinear differential equations that exhibits chaotic behavior. The data was provided in a MATLAB file named `data.mat`, which contained two matrices: `a` and `da`, each of size 5677×3. Specifically:

- `a` corresponds to the state vector (x, y, z) over 5677 time steps.
- `da` corresponds to the time derivatives $(\dot{x}, \dot{y}, \dot{z})$ at those same steps.

The objective was to model the mapping $f : (x, y, z) \to (\dot{x}, \dot{y}, \dot{z})$ using two methods:

- Manual implementation of multivariate linear regression using **gradient descent**.
- Using MATLAB's built-in `mvregress` function for comparison.

2 Methods and Implementation

2.1 Data Preparation

The dataset was randomly partitioned into:

- 70% training data
- 30% test data

Before training the model, the input features were scaled using a technique called min-max normalization. This method transforms each feature so that its values fall within a standard range. This helps ensure that all features contribute equally during training and prevents features with larger numerical ranges from dominating the learning process. In particular, it significantly improves the stability and speed of gradient descent by keeping the updates to the parameters more consistent across different features.

2.2 Manual Gradient Descent

Gradient descent is an iterative optimization algorithm used to minimize a cost function by updating parameters in the direction of the negative gradient. In this case, it was used to minimize the mean squared error between predicted and true time derivatives.

The parameters (weights) of the model were updated iteratively using the rule: The parameters (weights) of the model were updated iteratively using the rule:

$$\Theta_{\text{new}} = \Theta_{\text{old}} - \alpha \frac{dJ}{d\Theta}$$

where the gradient of the cost function with respect to the parameters is given by:

$$\frac{dJ}{d\Theta} = \frac{1}{m}\mathbf{X}^T(\mathbf{X}\Theta - \mathbf{Y})$$

where:

- Θ is the parameter matrix.
- α is the learning rate.
- $J(\Theta)$ is the cost function.

During training, the parameter matrix Θ was regularly saved to a file (`tht_data.mat`). This allowed adjustment of the learning rate α mid-training, improving convergence speed without restarting from scratch.

2.3 mvregress

MATLAB's `mvregress` function performs multivariate linear regression by solving for the regression coefficients that minimize the residual error. This method is computationally efficient and internally optimized.

To avoid issues encountered with the multivariate response structure, each output dimension was regressed independently by looping over each column of the target matrix. This allowed successful convergence without errors.

3 Feature Engineering Experiments

To explore how different input features affect model performance, several experiments were conducted. The goal was to assess how increasing feature complexity impacts prediction accuracy, convergence speed, and computational cost.

3.1 Experiment 1: Second-Order Polynomial Expansion (Final Model)

The feature set used in the final version of the model was based on a second-order Taylor expansion, which was also the feature set discussed during class. This made it a natural choice as a baseline for comparison against other, more complex feature configurations.

This expansion captures linear relationships, basic nonlinearities, and pairwise interactions between the state variables.

The following features were included:

- Linear terms: x, y, z
- Second-order terms: x^2, y^2, z^2, xy, xz, yz
- Bias term: a_0

This results in a total of **10 features**:

$$a_0,\ x,\ y,\ z,\ x^2,\ y^2,\ z^2,\ xy,\ xz,\ yz$$

Observations:

- Provided a good balance between model accuracy and simplicity.
- Low test error was achieved with relatively fast and stable convergence.
- Served as the baseline feature set for evaluating the impact of higher-order or alternative features.

3.2 Experiment 2: Third-Order Polynomial Expansion

To enhance the expressiveness of the model, a full third-order Taylor expansion was tested. This includes higher-order nonlinearities and more complex interactions between state variables.

The expanded feature set included:

- Linear terms: x, y, z
- Second-order terms: x^2, y^2, z^2, xy, xz, yz
- Third-order terms:
  - Pure cubics: x^3, y^3, z^3
  - Quadratic-linear combinations: x^2y, x^2z, y^2x, y^2z, z^2x, z^2y
  - Mixed term: xyz
- Bias term: a_0

This resulted in a total of **20 features**:

$$a_0,\ x,\ y,\ z,\ x^2,\ y^2,\ z^2,\ xy,\ xz,\ yz,\ x^3,\ y^3,\ z^3,\ x^2y,\ x^2z,\ y^2x,\ y^2z,\ z^2x,\ z^2y,\ xyz$$

Observations:

- A slight improvement in test prediction error was observed.
- However, training time increased significantly due to added parameters.
- The improvement was not substantial enough to justify the extra complexity.

3.3 Experiment 3: Trigonometric Features

Given the structured and circular nature of the Lorenz system (see Figure 1), trigonometric functions were also tested as potential features.

- Added: $\sin(x)$, $\sin(y)$, $\sin(z)$, $\cos(x)$, $\cos(y)$, $\cos(z)$

Observations:

- Motivated by the visual structure of the data, which appeared periodic or rotational.
- Despite the intuition, these features did not significantly improve model performance.
- Convergence using gradient descent became slower and less stable.
- Ultimately, these features were not included in the final model.

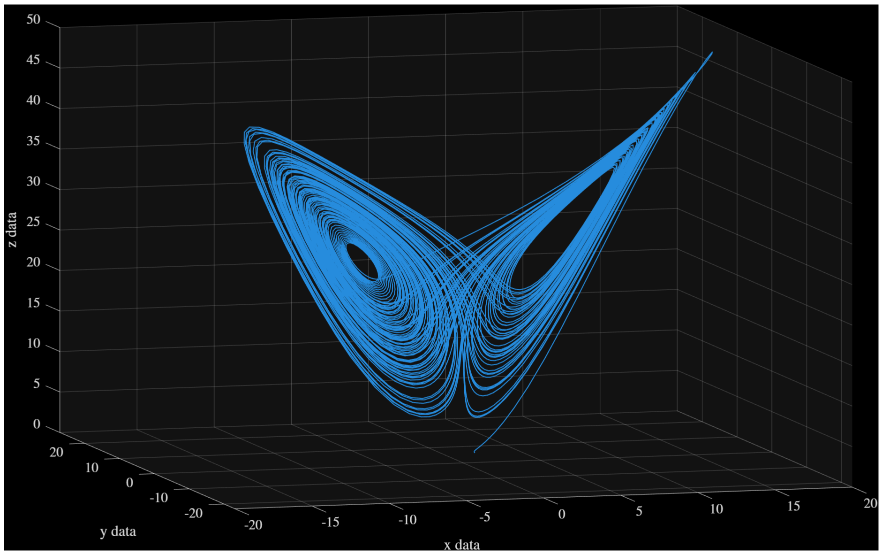

Figure 1: Lorenz system plotted from (x, y, z) data

4 Comparison of Methods

The results from both methods were evaluated using the normalized error on the test set:

$$\text{Error} = \frac{||\hat{Y} - Y||}{||Y||} \times 100\%$$

Manual Gradient Descent

- Final test error: **0.034%**
- Required multiple training iterations and manual tuning of the learning rate.

mvregress

- Final test error: **0.000%**
- Much faster training and more stable convergence.

While gradient descent provided a solid understanding of the optimization process, `mvregress` proved more efficient and accurate for this task. Given the negligible test error and lower computation time, it was deemed the preferred approach.

5 Conclusion

This assignment demonstrated how multivariate linear regression can be used to learn the dynamics of a nonlinear system like the Lorenz system. By exploring feature engineering and comparing optimization methods, valuable insights were gained into the trade-offs between model complexity, accuracy, and computational efficiency.

Ultimately, `mvregress` provided the most reliable solution, while gradient descent offered more flexibility and educational value in understanding how parameters are iteratively optimized.