GEO-5017 Assignment 1: Linear Regression

Group 9: Ming-Chieh Hu, Neelabh Singh & Frederick Auer

March 5, 2025

1 Basic model

Suppose we have observed N data points $\{x_i, t_i\}_{i=1...N}$:

1. **What do we need to do with the data before we can formulate a linear regression problem?** To formulate a linear regression problem, you need to assume a model that mathematically relates input(independent) and output(dependent) variables, to ensure the data meets the assumptions of linear regression i.e. linearity. The relationship between independent and dependent variables should be linear.

2. **What would be the optimal solution given your hypothesized model?** Let's say we have a simple model $x = at + b$. The objective function is $E = \sum_{i=0}^{n}(x_i - (at_i + b))^2$, and the optimal solution will occur when one find the proper a and b (parameters) that minimizes the difference between the observed values and the values predicted by the model. For a simple linear regression model the solution to a and b is $a = \bar{y} - b\bar{t}$, $b = \frac{Cov(x,y)}{Var(x)}$ with $\bar{x} = mean(x)$ and $\bar{y} = mean(y)$.

2 Programming: drone

2.1 Drone Trajectory

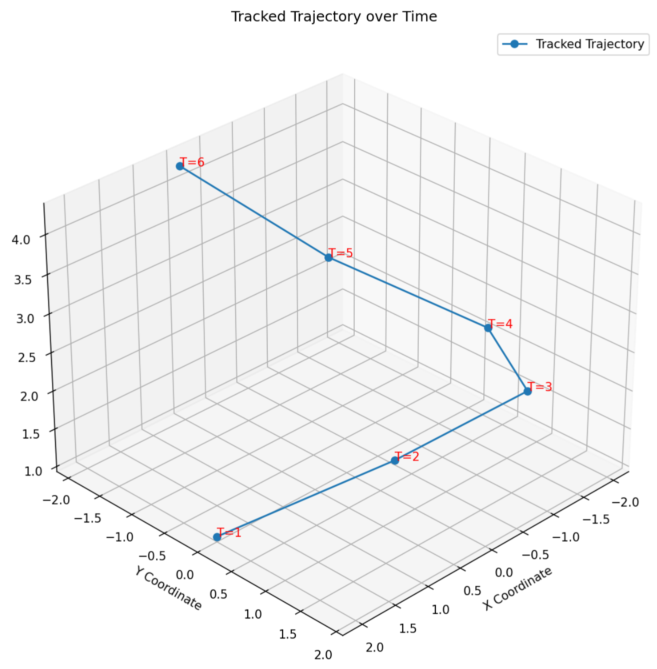


Figure 1: Tracked drone trajectory over time

2.2 Polynomial Regression

(a) Constant speed model

The model is a mapping from time t to point position $\mathbf{p} = (x, y, z)$. We know that the position $p = p_0 + v_0 t$. With $\alpha_0, \beta_0, \gamma_0$ equals to initial position x_0, y_0, z_0 and $\alpha_1, \beta_1, \gamma_1$ equals to constant speed v_x, v_y, v_z, we can derive the following equations 1 and 2:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha_0 + \alpha_1 t \\ \beta_0 + \beta_1 t \\ \gamma_0 + \gamma_1 t \end{pmatrix} \tag{1}$$

$$\mathbf{P} = \mathbf{DA}, \mathbf{P} = \begin{pmatrix} \mathbf{p_1^T} \\ \vdots \\ \mathbf{p_n^T} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} x_1 & y_1 & z_1 \end{pmatrix} \\ \begin{pmatrix} x_2 & y_2 & z_2 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} x_n & y_n & z_n \end{pmatrix} \end{pmatrix}, \mathbf{D} = \begin{pmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \\ 1 & t_n \end{pmatrix}, \mathbf{A} = \begin{pmatrix} \alpha_0 & \beta_0 & \gamma_0 \\ \alpha_1 & \beta_1 & \gamma_1 \end{pmatrix} \tag{2}$$

1. **What speed does it have?**
   If the drone flies with a constant speed in each of the three dimensions, it's speed can be calculated using the velocity components v_x, v_y, and v_z, we use the following formula : Speed $= \sqrt{v_x^2 + v_y^2 + v_z^2}$.

   Given the values: $v_x = -0.422, \quad v_y = -0.554, \quad v_z = 0.481$. Speed $= \sqrt{0.716361} \approx 0.846$. So, the speed is approximately **0.846 units per time step**. Let's assume each unit is a meter and each time step a second, then its speed would therefore be **0.846 m/s**.

2. **What is the residual error of the estimated positions?**
   The residual error is 16.0169 according to our implementation (Figure 2) after 100 iterations.

Check out the `Degree1` class in the source code for the implementation. The result we obtained here was generated using these parameters: `python main.py -d 1 -it 100 -lr 0.01`.

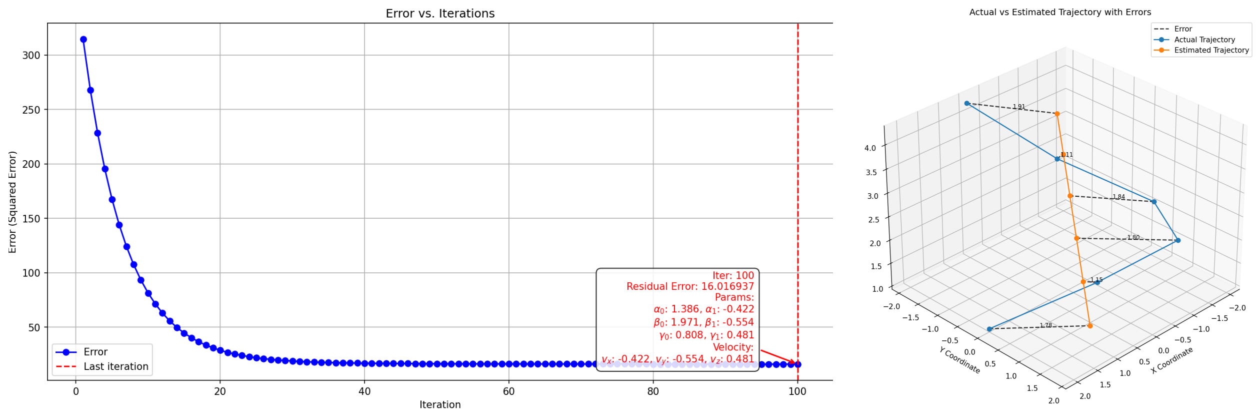


Figure 2: Fitting process and results for constant speed model

(b) Constant acceleration model

We know that the position $p = p_0 + v_0 t + \dfrac{at^2}{2}$. With $\alpha_2', \beta_2', \gamma_2'$ equals to acceleration a_x, a_y, a_z and equation 1, we can derive the following equations 3 and 4:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha_0 + \alpha_1 t + \dfrac{\alpha_2' t^2}{2} \\ \beta_0 + \beta_1 t + \dfrac{\beta_2' t^2}{2} \\ \gamma_0 + \gamma_1 t + \dfrac{\gamma_2' t^2}{2} \end{pmatrix} = \begin{pmatrix} \alpha_0 + \alpha_1 t + \alpha_2 t^2 \\ \beta_0 + \beta_1 t + \beta_2 t^2 \\ \gamma_0 + \gamma_1 t + \gamma_2 t^2 \end{pmatrix} \tag{3}$$

$$\mathbf{P} = \mathbf{DA}, \mathbf{P} = \begin{pmatrix} \mathbf{p_1^T} \\ \vdots \\ \mathbf{p_n^T} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} x_1 & y_1 & z_1 \end{pmatrix} \\ \begin{pmatrix} x_2 & y_2 & z_2 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} x_n & y_n & z_n \end{pmatrix} \end{pmatrix}, \mathbf{D} = \begin{pmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ & \vdots & \\ 1 & t_n & t_n^2 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} \alpha_0 & \beta_0 & \gamma_0 \\ \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \end{pmatrix} \tag{4}$$

1. **What is its acceleration?**
   If the drone has acceleration components a_x, a_y, and a_z, the magnitude of the acceleration can be calculated using the following formula: acceleration $= \sqrt{a_x^2 + a_y^2 + a_z^2}$. Given the values: $a_x = 0.857, \quad a_y = -0.644, \quad a_z = 0.109$, the acceleration value is $\sqrt{1.161066} \approx 1.078$.

   So, the acceleration is approximately **1.078 units per time step squared**. Assuming each unit is a meter and each time step is a second, the acceleration would therefore be **1.078 m/s²**.

2. **What is the residual error compared to constant speed model, is the error higher or lower? Why?**
   The residual error is 4.9432 according to our implementation (Figure 3) after 100000 iterations, which is lower than the linear constant speed model.

3. **Can you still improve your model?**
   We can always incorporate a more suitable or a higher-degree polynomial model to fit with the observed dataset and predict values with less error. However, we should be cautious while defining models, especially in such a small dataset, as it can lead to overfitting with inaccurate predictions (Figure 4).

Check out the `Degree2` class in the source code for the implementation. The result we obtained here was generated using these parameters: `python main.py -d 2 -it 100000 -lr 0.0001` (we have tuned down the learning rate to make it converge).

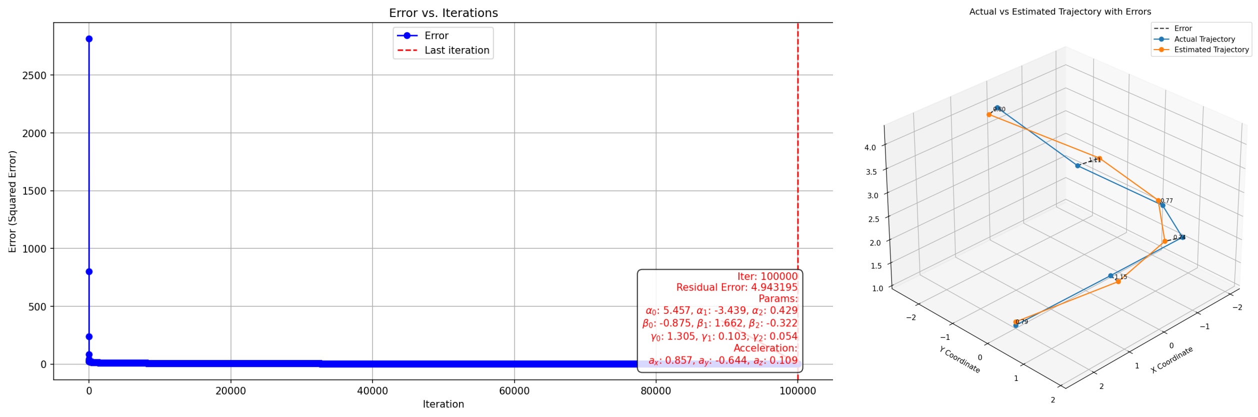


Figure 3: Fitting process and results for constant acceleration model

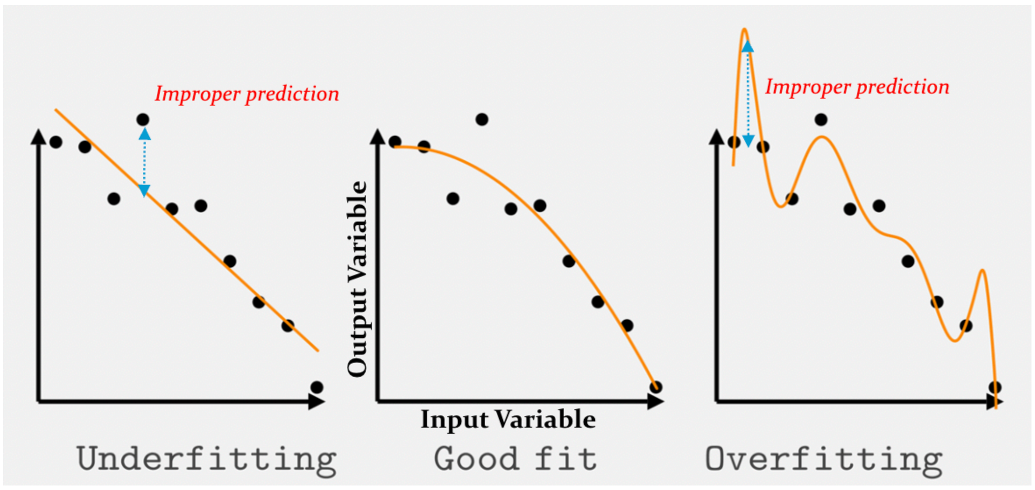


Figure 4: Different degrees of model

(c) Prediction

When $t = 7$ using constant acceleration model and its parameters, we construct a D matrix $D_{t=7} = (1 \quad 7 \quad 49)$ and use it in the equation 3.

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 & t & t^2 \end{pmatrix} \begin{pmatrix} \alpha_0 & \beta_0 & \gamma_0 \\ \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \end{pmatrix} = \begin{pmatrix} 5.457 + (-0.875)t + 1.305t^2 \\ (-3.439) + 1.662t + 0.103t^2 \\ 0.429 + (-0.322)t + 0.054t^2 \end{pmatrix} \tag{5}$$

As showed in Figure 5, according to equation 5, the predicted position $p_7 = \begin{pmatrix} 63.277 \\ 13.242 \\ 0.821 \end{pmatrix}$.

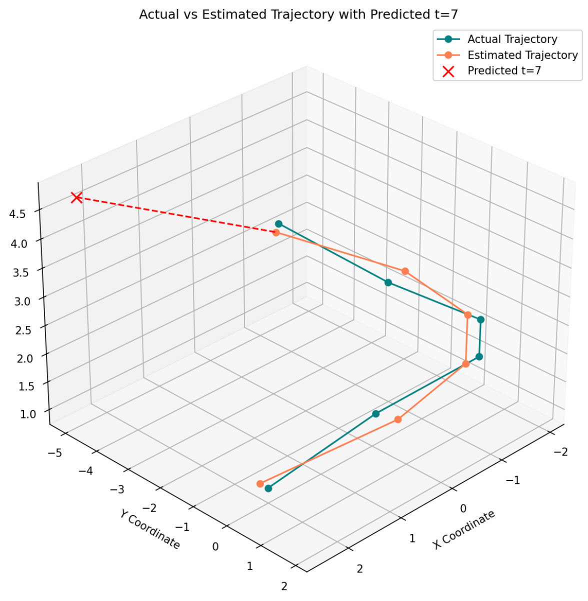


Figure 5: Plotting position t=7 with previous estimated positions

2.3 Experiments

For the constant speed model it's easy to converge no matter what parameters we chose to use, so we didn't make experiments exhaustively (Table 1). For the constant acceleration model, we observed that the trend of error reduction continued even after a large number of iterations (Table 2). We fixed the learning rate to 0.0001 because this model is prone to gradient explosions if the learning rate is not carefully chosen.

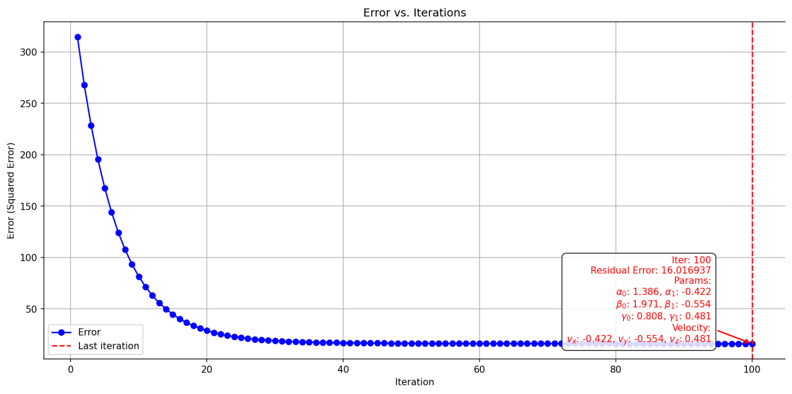
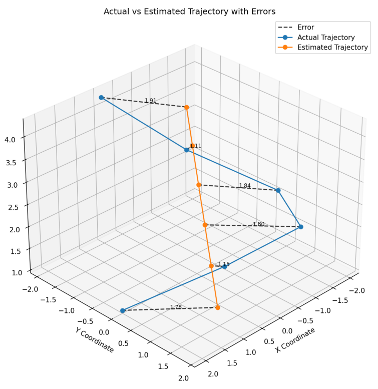
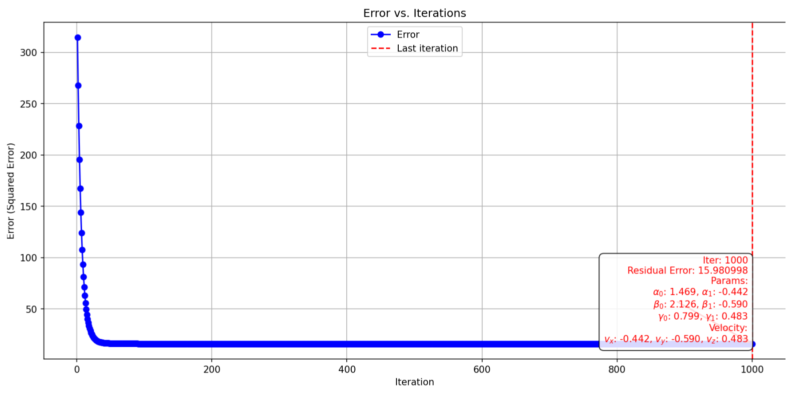
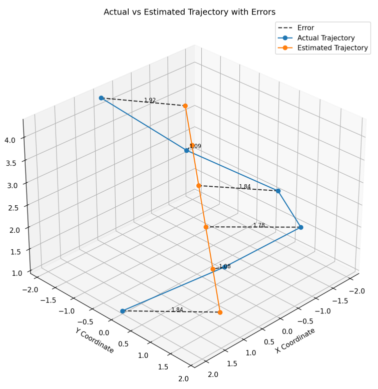
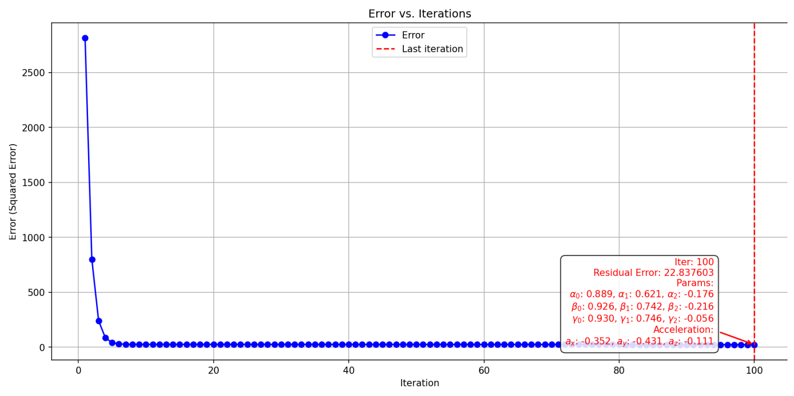
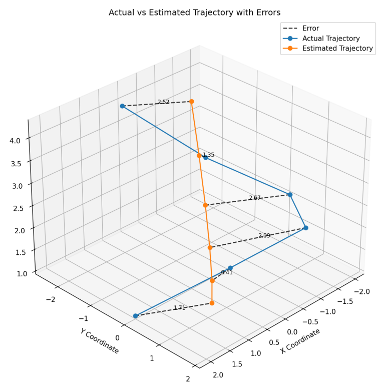
Fitting Process: Error-Iterations	**Trajectory Plot**
	
↑ learning rate: 0.01, iteration: 100	
	
↑ learning rate: 0.01, iteration: 1000	

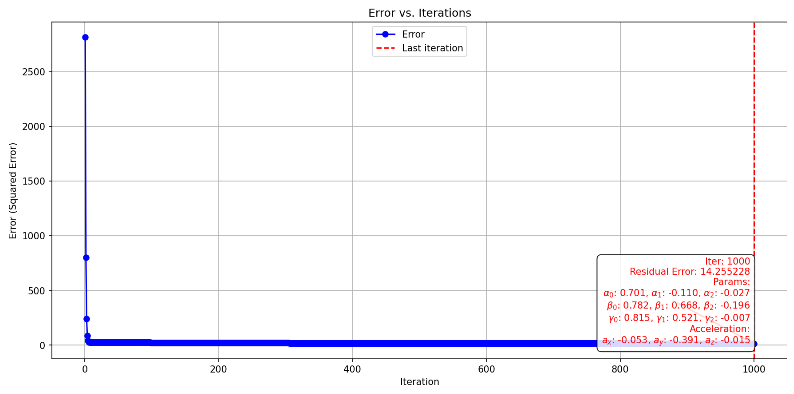
Table 1: Experiments with the constant speed model

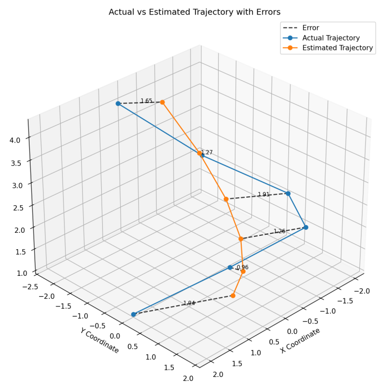
Fitting Process: Error-Iterations	Trajectory Plot



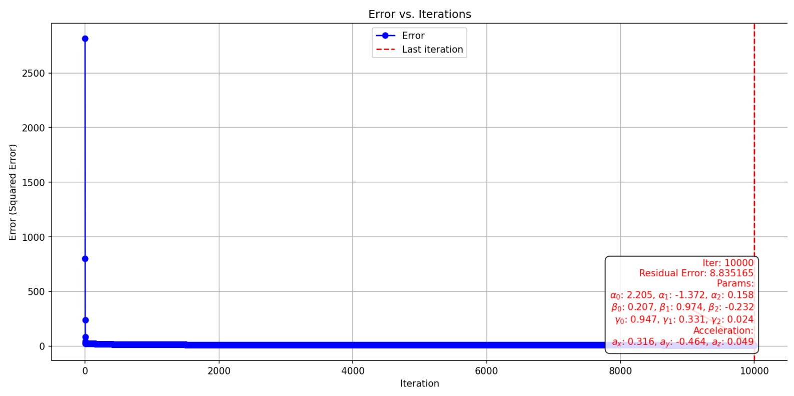


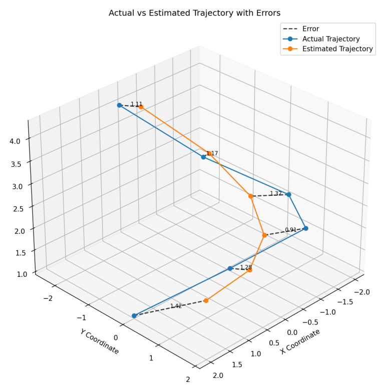
↑ learning rate: 0.0001, iteration: 100



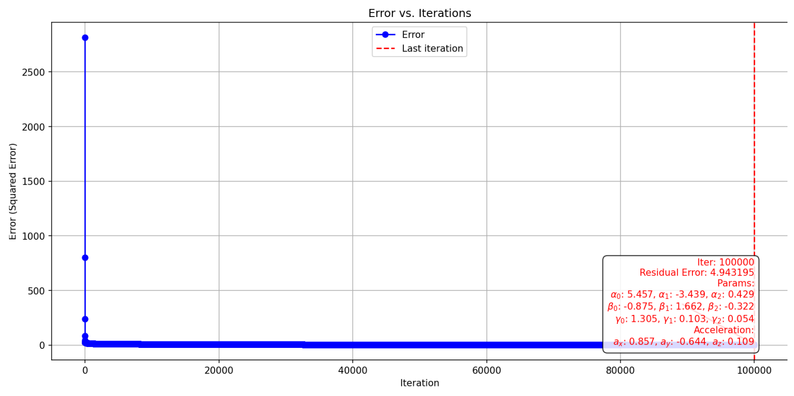


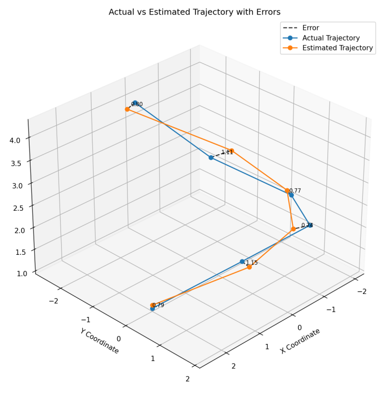
↑ learning rate: 0.0001, iteration: 1000





↑ learning rate: 0.0001, iteration: 10000





↑ learning rate: 0.0001, iteration: 100000

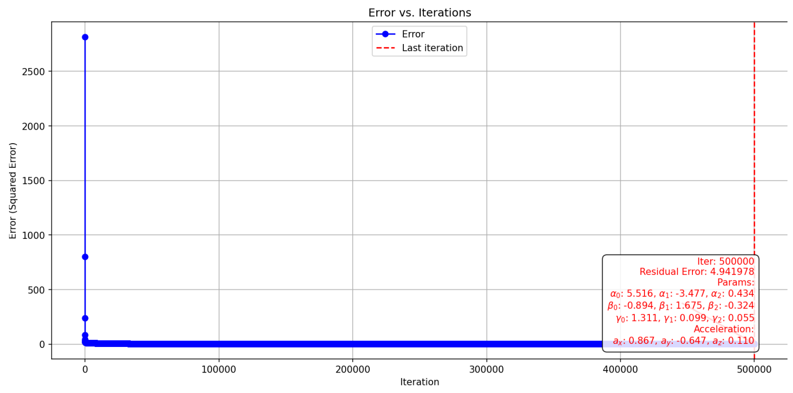
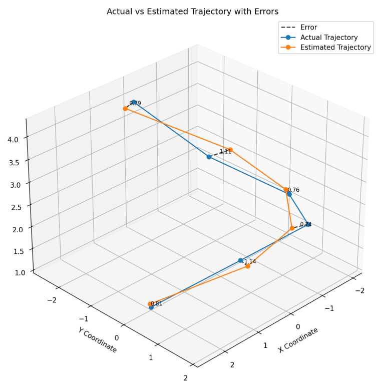
Fitting Process: Error-Iterations	**Trajectory Plot**
	
↑ learning rate: 0.0001, iteration: 500000 (no improvement since iteration=100000)	

Table 2: Experiments with the constant acceleration model

Link to github repository: https://github.com/MCHU-1999/geo-5017-assignments.

Credit

- **Ming-Chieh Hu**: derived formulas, class definition, fitting function, gradient calculation and report drafting.
- **Neelabh Singh**: derived formulas, plotting gradient decent and errors, theoretical questions, and overfitting.
- **Frederick Auer**: plotting functions for both the ground truth and the predicted value, edits on report.