Validation of Lift Modeling Accuracy in Python Through Paper Airplane Flight Behavior Analysis

[1]Eun June Lee, [2]Hyeon Seong Kim
[1,2]Department of Mechanical Engineering, Gaon Highschool, Gyeonggi-do, Republic of Korea

[1]Corresponding Author: eunjune1208@gmail.com

Keywords: Numerical Analysis, Simulation, Paper Airplane, Lift Modeling Accuracy

Abstract

Python is a widely utilized programming language for numerical analysis and simulation. This study aims to evaluate the efficiency and accuracy of lift modeling using Python by comparing numerically simulated flight behavior of a paper airplane with its actual observed performance. The simulation incorporates the effects of lift and gravity, while the real-world flight trajectory is approximated through curve fitting. The degree of similarity between the two results is assessed to validate the numerical modeling approach.

1. Introduction

Analyzing the behavior of an object subjected to multiple physical forces, including lift, through equations of motion is a highly complex process. As a result, numerical analysis is commonly employed to address such problems. Among various numerical techniques, simulation-based analysis using programming languages such as Python has become a widely adopted approach. This method offers the advantage of performing numerical computations without requiring manual calculations.

In this study, numerical analysis is conducted using Python as a representative programming language. Specifically, the Euler Method—one of the most fundamental numerical techniques—is implemented in Python to simulate the behavior of a paper airplane. The accuracy and efficiency of this Python-based numerical approach are assessed by comparing the simulated trajectory with actual flight observations. To induce more complex and nonlinear dynamics, the experiment is designed around a paper airplane, with interdependent calculations of lift and velocity incorporated into the model.

2. Theoretical Background

Herein, this chapter explains two similarity comparison methods used to compare the results obtained from simulations with the actual behavior analysis within the study. The methods employed are the RMSE (Root Mean Square Error) approach and the Cosine Similarity approach, which were used to verify numerical accuracy and the similarity of patterns or shapes, respectively.

2.1. Root Mean Squared Error (RMSE)

Root Mean Squared Error (hereinafter RMSE) is a regression model also known as the root mean square error. By applying the square root to the Mean Squared Error (MSE), which is the average of squared errors, RMSE reduces the distortion caused by squaring the errors. The formula for calculating RMSE is as follows.

$$RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2} \quad (1)$$

The advantage of RMSE is its intuitiveness. However, it is difficult to determine whether the prediction is lower or higher than the actual value. Additionally, RMSE is scale-dependent, and even if the magnitude of the error is the same, the error rate may vary depending on the model. An RMSE value closer to zero is interpreted as indicating a smaller error.

2.2. Cosine Similarity

Cosine Similarity refers to the similarity between two vectors measured using the cosine of the angle between them in an inner product space. The magnitude of the vectors does not affect the similarity; only the similarity of their directions is considered. Cosine Similarity is defined as follows.

$$SIM=\frac{\sum_{i=1}^{n}A_i\times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\times\sqrt{\sum_{i=1}^{n}(B_i)^2}} \quad (2)$$

Cosine similarity is interpreted such that the closer the value is to 1, the smaller the error.

3. Analysis of Paper Airplane Behavior through Simulation

Since the paper airplane was made from a single sheet of A4 paper, its weight was set to 0.005 kg, which is the weight of one A4 sheet. Additionally, the mass distribution of the paper airplane was assumed to be uniform. The total length of the paper airplane is 30.7 cm, and its structure is as follows.

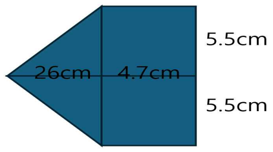

Figure.1 Feature of paper airplane.

When taking the left tip of the paper airplane as the origin, the coordinates of the airplane's center of mass calculated using the area-weighted average method are as follows.

$$(\overline{x},\overline{y})=(20.26cm,5.5cm) \quad (3)$$

Since the mass distribution of the paper airplane is uniform, the centroid coincides with the center of gravity, which is the point of application of gravity. The point of application of lift was set to a position 10% behind the centroid.

$$(x_{lift},y_{lift})=(22.286cm,5.5cm) \quad (4)$$

The gravitational force and lift acting on the paper airplane are as follows.

$$F_{gravity}=0.049N \quad (5)$$

$$F_{Lift}=0.00158\cdot v^2N \quad (6)$$

Based on these two forces, the paper airplane's vertical and horizontal rotational motions were simulated. To prevent excessive rotation, air resistance was implemented as a resistive force against rotation in the program.

The complete code is provided in Appendix A. The simulation results are as follows.

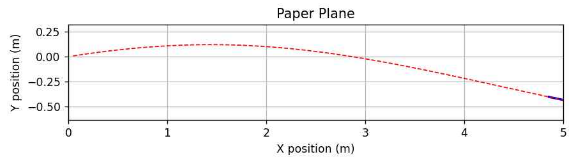

Figure.2 Paper airplane trajectory simulation.

In Fig. 2, the x-axis represents the horizontal position of the airplane relative to the ground, while the y-axis represents its height. In the range where x is approximately between 0 and 1.5, the height increases, and then decreases afterward, forming a rising and falling trajectory.

The initial velocity, initial angle, and initial position for the simulation were programmed using experimentally obtained data. Figure 3 shows the graphs of the lift acting on the paper airplane, as well as the velocity and rotational angle of the paper airplane, calculated based on the simulation.

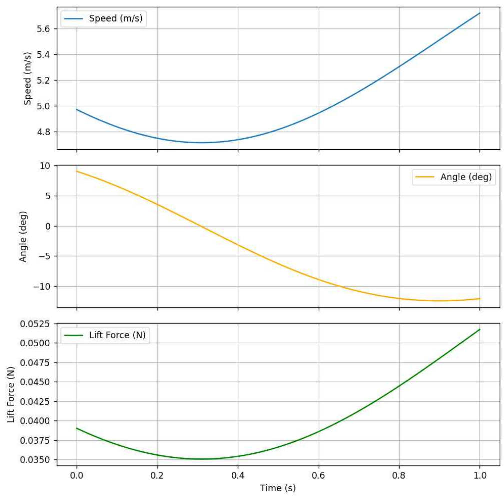

Figure.3 Simulation of paper airplane velocity, angle, and lift.

The graphs in Fig. 3 represent, from top to bottom, the velocity, angle, and lift of the paper airplane. The velocity decreases from the initial point and then begins to increase after the airplane reaches its maximum altitude. The lift shows a similar trend to the velocity, which appears to result from their mutual interaction through cross-referencing. The angle continuously decreases without any increasing phase.

4. Analysis of Actual Paper Airplane Behavior

The experimentally obtained data of the actual paper airplane's behavior are as follows.

Table.1 Paper airplane time/position table.

Time(s)	x	y
9.267	51.616	187.887
9.3	68.026	187.893
9.333	82.164	188.988
9.4	96.292	188.56
9.433	122.715	185.824
9.5	139.119	184.719
9.533	151.871	182.502
9.567	165.077	179.715
9.6	191.026	172.42
9.667	206.511	169.011
9.7	221.992	164.455
9.733	235.192	159.316
9.767	262.961	147.802
9.833	276.608	139.149
9.867	289.362	134.448
9.9	302.556	125.137
9.933	332.609	108.584
10	347.177	97.921
10.033	359.933	90.706
10.067	378.61	78.652

Based on the data, the two functions obtained using Python are as follows. The complete Python code is provided in Appendix B.

$$y = 6.46\times10^{-9}x^4 - 5.27\times10^{-6}x^3 + 0.0001x^2 + 0.04x + 185.76 \tag{7}$$

$$y = 1.01e^{0.1x} - 0.001x^2 + 0.21x + 179.69 \tag{8}$$

Within the measured range, the accuracy of Eq. (8) was higher; however, considering the subsequent trends, Eq. (7) was judged to have better overall accuracy and was therefore adopted. The graph of Eq. (7) is shown in Fig. 4.

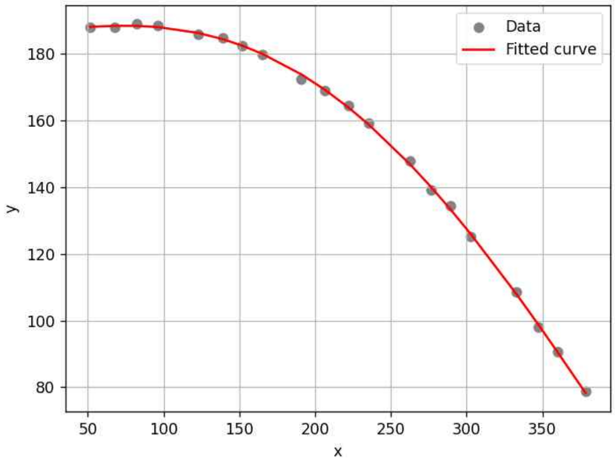

Figure.4 Paper airplane trajectory function.

5. Similarity Analysis

The data extracted from the simulation results are as follows.

Table.2 Table of extracted data from paper airplane simulation.

x	y
0	0
1	0.15
2	0.20
3	0.10
4	-0.10
5	-0.45

The data extracted from the experimental result graphs are as follows.

Table.3 Table of extracted data from paper airplane experiment.

x	y
50	188
100	188
150	185
200	180
250	165
300	140
350	100
380	80

This can be normalized using the following equation.

$$y' = \frac{y - y_{\min}}{y_{\max} - y_{\min}} \tag{9}$$

For the RMSE analysis, values corresponding to six x-interval ratios from the two graphs were sampled. The sampled values are shown below. represents the values from the simulation, and represents the values derived from the actual experimental graph.

Table.4 Normalized data table.

x구간	y_1	y_2
0	0.692	1.0
1	1.0	1.0
2	1.154	0.972
3	0.846	0.926
4	0.538	0.789
5	0	0

The RMSE value is calculated as follows.

$$RMSE = \sqrt{\frac{1}{6}\sum_{i=1}^{6}(y_{1,i} - y_{2,i})^2} \approx 0.181 \tag{10}$$

Cosine similarity is calculated as follows.

$$SIM = \frac{\sum_{i=1}^{6} y_{1,i} \times y_{2,i}}{\sqrt{\sum_{i=1}^{6} (y_{1,i})^2} \times \sqrt{\sum_{i=1}^{6} (y_{2,i})^2}} \approx 0.977 \quad (11)$$

6. Conclusion and Recommendations

The experimental results showed that the RMSE value was 0.181, which is very close to zero. Additionally, the Cosine Similarity value was 0.977, also very close to one. The analysis of the reasons behind such high similarity values is as follows.

In the actual flight of an airplane, nonlinear forces and flows such as turbulence and vortices occur, making it nearly impossible for an uncontrolled object like a paper airplane－whose motion cannot be corrected after launch－to behave exactly as predicted. Nevertheless, the close match between the Python simulation and actual behavior can be inferred for the following reasons.

First, the structure of the paper airplane's wings. Unlike real airplanes, the paper airplane's wings do not have an airfoil shape. Being flat like a simple plate, there is almost no difference in airflow velocity above and below the wings, significantly reducing the possibility of vortex formation compared to conventional airplane wings. Furthermore, due to the small velocity difference, the shear forces that cause turbulence are also minimized.

Second, the angle of attack of the paper airplane used in this study is very small, which further reduces shear forces and the likelihood of turbulence. The generation of lift is also influenced by the angle of attack, and a small angle reduces the magnitude of lift. Since speed and lift interact in airplane behavior, analyzing the motion is generally complex. However, in the case of the paper airplane, the relatively small lift generated by the small angle of attack lessens this complexity. As a result, errors arising during lift calculation have less impact, leading to results that closely approximate actual behavior.

These results confirm that the numerical simulation performed using Python can produce outcomes highly similar to actual behavior. Despite considering only lift, gravity, and drag among the many complex physical forces occurring in real environments, the close match with experimental results demonstrates that Python-based numerical simulations can partially substitute actual experiments for analyzing simple object movements.

It is hoped that future experiments and research will improve the fidelity of numerical methods like Python simulations in reproducing real-world conditions. Moreover, since these approaches can efficiently and rationally handle complex calculations such as cross-references, they are expected not only to replicate real environments but also to enable simulations of conditions unattainable in laboratories and expand the horizons of numerical analysis methods.

■ Reference

[1] Timothy O. Hodson, “Root-mean-square-error (RMSE) or mean absolute error (MAE): when to use them or not,” *Geoscientific Model Development*, vol. 15, issue. 14, pp.5481-5487, 2022.

[2] Osman Gunay, Cem Emre Akbas, A. Enis Cetin, “Cosine Similarity Measure According to a Convex Cost Function,” *arXiv*, arXiv:1410.6093, 2014

[3] Jörg U Schlüter, “Aerodynamic study of the dart paper airplane for micro air vehicle application,” *Sage Journals*, vol. 228, Issue. 4, pp.1-10, 2013

[4] Rabii El Maani, Soufiane Elouardi, Bouchaib Radi, Abdelkhalek El Hami, “Study of the turbulence models over an aircraft wing,” *ISTE OpenScience*, 2018

[Appendix A]

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

g = 9.81
Fg = 0.049
m = Fg / g
L = 0.307
w = 0.11
r = -0.01013 #point of application of Lift

dt = 0.01
max_steps = 10000

theta = np.radians(9)
vx = 4.97 * np.cos(theta)
vy = 4.97 * np.sin(theta)
x, y = 0.0, 0.0

x_data, y_data, theta_data = [], [], []
v_data = []
Fl_data = []

for _ in range(max_steps):
    v = np.sqrt(vx**2 + vy**2)
    if v == 0:
        break

    theta = np.arctan2(vy, vx)
```

```
    Fl = 0.00158 * v**2

    fx, fy = np.cos(theta), np.sin(theta)
    nx, ny = -fy, fx

    Flx = Fl * nx
    Fly = Fl * ny

    Fx = Flx
    Fy = Fly - Fg

    ax = Fx / m
    ay = Fy / m
    vx += ax * dt
    vy += ay * dt
    x += vx * dt
    y += vy * dt

    x_data.append(x)
    y_data.append(y)
    theta_data.append(theta)
    v_data.append(v)
    Fl_data.append(Fl)

    if x >= 5.0:
        break

fig1, ax1 = plt.subplots(figsize=(8,5))
ax1.set_xlim(0, 5.0)
ax1.set_ylim(min(y_data)-0.2, max(y_data)+0.2)
ax1.set_aspect('equal')
plane_line, = ax1.plot([], [], 'b-', lw=2)
trace_line, = ax1.plot([], [], 'r--', lw=1)

trace_x, trace_y = [], []

def init():
    plane_line.set_data([], [])
    trace_line.set_data([], [])
    return plane_line, trace_line

def animate(i):
    cx, cy = x_data[i], y_data[i]
    angle = theta_data[i]
```

```
    dx = (L / 2) * np.cos(angle)
    dy = (L / 2) * np.sin(angle)

    x1, x2 = cx - dx, cx + dx
    y1, y2 = cy - dy, cy + dy
    plane_line.set_data([x1, x2], [y1, y2])

    trace_x.append(cx)
    trace_y.append(cy)
    trace_line.set_data(trace_x, trace_y)

    return plane_line, trace_line

ani = animation.FuncAnimation(
    fig1, animate, frames=len(x_data),
    init_func=init, blit=True, interval=10,
    repeat=False
)

plt.title("Paper Plane")
plt.xlabel("X position (m)")
plt.ylabel("Y position (m)")
plt.grid(True)

fig2, (axv, axtheta, axFl) = plt.subplots(3, 1, figsize=(8, 8), sharex=True)
time = np.arange(len(v_data)) * dt

axv.plot(time, v_data, label="Speed (m/s)")
axv.set_ylabel("Speed (m/s)")
axv.grid(True)
axv.legend()

axtheta.plot(time, np.degrees(theta_data), label="Angle (deg)", color='orange')
axtheta.set_ylabel("Angle (deg)")
axtheta.grid(True)
axtheta.legend()

axFl.plot(time, Fl_data, label="Lift Force (N)", color='green')
axFl.set_xlabel("Time (s)")
axFl.set_ylabel("Lift Force (N)")
axFl.grid(True)
axFl.legend()

plt.tight_layout()

plt.show()
```

[Appendix B]

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def model_func(x, A, B, C, D, E):
    return A*np.exp(B*x) + C * x**2 + D * x + E

x_data = [51.616, 68.026, 82.164, 96.292, 122.715, 139.119, 151.871, 165.077, 191.026, 206.511,
    221.992, 235.192, 262.961, 276.608, 289.362, 302.556, 332.609, 347.177, 359.933, 378.61]
x_data = np.array(x_data)
y_data = [187.887, 187.893, 188.988, 188.56, 185.824, 184.719, 182.502, 179.715, 172.42, 169.011,
    164.455, 159.316, 147.802, 139.149, 134.448, 125.137, 108.584, 97.921, 90.706, 78.652]

initial_guess = [1, 0.1, 1, 1, 1]

popt, pcov = curve_fit(model_func, x_data, y_data, p0=initial_guess)

A_fit, B_fit, C_fit, D_fit, E_fit = popt
print("Fitted parameters:")
print(f"A = {A_fit}, B = {B_fit}, C = {C_fit}, D = {D_fit}, E = {E_fit}")

y_fit = model_func(x_data, *popt)

plt.scatter(x_data, y_data, label='Data', color='gray')
plt.plot(x_data, y_fit, label='Fitted curve', color='red')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Nonlinear Fit: $y = Ae^{Bx} + Cx^2 + Dx + E$')
plt.grid(True)
plt.show()
```