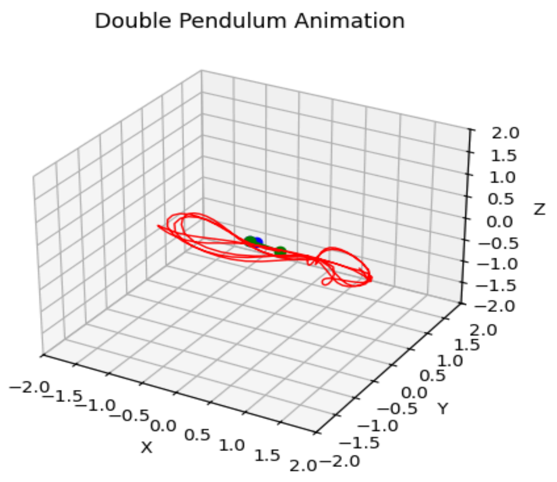


Submitted by:

Team name : <u>Tech coders</u>

Team members:

Mahendra Tard , Roll no- 22EJCCS123

Kunal Kumar , Roll no- 22EJCCS117

Under the gidance of:

Faculty Name: Uma maheshwari

Department of Computer Science Engineering

JECRC FOUNDATION

2024-25

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

INTRODUCTION

Double Pendulum Simulation-:

Double pendulum is a simple yet fascinating physical system with unpredictable and chaotic behavior. This simulation aims to provide insights into the complex dynamics of a double pendulum and its sensitivity to initial conditions. The study of chaotic systems has applications in various fields, including physics, engineering, and even meteorology.

Objective

To design and develop a realistic 3D animation of a double pendulum system that visually demonstrates the principles of chaotic motion and nonlinear dynamics. The animation aims to accurately simulate the physical behavior of the double pendulum using appropriate mathematical modeling, physics-based rendering, and real-time visualization techniques for educational and research purposes.

Scope

- The scope of a project involving 3D animation and 3D plot development in Python programming is vast and diverse, spanning across scientific research, education, entertainment, and various technical domains. In scientific research, Python's powerful libraries such as Matplotlib, Plotly, and Mayavi facilitate the creation of intricate 3D visualizations, enabling scientists to explore and communicate complex ata and simulations effectively. In education, Python's accessibility makes it an excellent choice for developing interactive 3D animations that enhance the learning experience, particularly in fields like physics and computer science. Additionally, the application of 3D animation in entertainment, game development, and virtual reality experiences is extensive, with Python libraries like Pygame and Panda3D providing tools for creative expression. Furthermore, Python's capabilities in robotics simulation, augmented reality, and data visualization contribute to the project's potential applications in diverse sectors. Overall, a project involving 3D animation and 3D plot development in Python offers a broad scope, combining scientific exploration, educational innovation, and creative expression across various industries and disciplines.

Expected Outcome

- **Accurate Simulation**: A physically accurate simulation of a double pendulum using appropriate differential equations and numerical integration methods.

- **Realistic 3D Animation**: A visually appealing 3D animation that clearly represents the chaotic motion of the double pendulum in real time.

- **User Interaction (Optional)**: Ability for users to manipulate initial conditions (such as angle and length) and observe resulting changes in the pendulum's motion.

- **Educational Value**: An intuitive and engaging tool to help students and researchers understand complex dynamics and chaotic systems.

- **Performance Optimization**: A smooth and responsive animation optimized for real-time performance on standard computing hardware.

- **Visualization of Key Data**: Inclusion of graphs, trails, or overlays showing angular velocity, energy, or phase space to enhance understanding of the system’s dynamics.

Technologies Used

1. Programming Language

Python 3.x – Simple, powerful, and ideal for scientific computing and visualization.

2. Mathematical Modeling & Simulation

NumPy – For numerical operations and handling arrays.

SciPy – For solving the system of differential equations (e.g., using scipy.integrate.odeint or solve_ivp).

3. 3D Visualization & Animation

VPython (Visual Python) – Easiest way to create 3D simulations with real-time interactivity.

Matplotlib (optional) – For plotting graphs like angular velocity, energy, or phase diagrams.

PyOpenGL + Pygame – For more custom 3D rendering if you want full control (advanced option).

Manim – For making mathematically precise animations, though it's more for pre-rendered videos than real-time interactivity.

4. Development Environment

Jupyter Notebook or **Google Colab** – Great for testing and showcasing simulations with visual output.

VS Code / PyCharm – Full-featured code editors for Python development.

5. Optional Enhancements

Tkinter / PyQt – If you want to build a GUI to adjust parameters like pendulum length or angle.

Pandas – For handling and analyzing simulation data if you're doing more complex output.

Methodology & Implementation

Methodology & Implementation
1. Problem Definition

The objective is to simulate and visualize the chaotic motion of a double pendulum in a 3D environment. This requires solving the pendulum's equations of motion and rendering the motion in real-time.

2. Mathematical Modeling

The double pendulum consists of two masses connected by rigid rods.

The system is governed by a set of **nonlinear differential equations**, which describe angular positions and velocities.

The equations are derived using **Lagrangian mechanics**, involving kinetic and potential energy.

State Variables:

θ_1: Angle of first pendulum

θ_2: Angle of second pendulum

ω_1: Angular velocity of first pendulum

ω_2: Angular velocity of second pendulum

Numerical Solver:

Use scipy.integrate.solve_ivp() or odeint() to solve the system of equations.

3. Tools and Libraries Use

Python 3

NumPy for math operations

SciPy for solving ODEs

VPython (via vpython module) for 3D visualization and real-time animation

Matplotlib (optional) for plotting graphs of energy, angles, or trajectories

4. Implementation Steps

Step 1: Initialize System Parameters

-

Masses, lengths, gravitational constant, initial angles, and angular velocities.

-

Step 2: Define the Equations of Motion

-

Use Lagrange's equations to derive a system of 4 first-order differential equations.

-
-

Implement a function that returns the time derivatives of the state variables.

Step 3: Solve the ODE System

Use solve_ivp to numerically solve the equations over a specified time period.

Step 4: Set Up 3D Visualization (VPython)

Create 3D objects: rods (as cylinders) and masses (as spheres).

Update their positions at each time step using the results from the solver.

Step 5: Animate the Motion

Use a time loop to update the 3D scene in real time, synchronizing it with the solver output.

Optional Step: Add Graphs or UI Controls

Add energy plots or GUI sliders (via Tkinter) to explore system dynamics interactively.

---

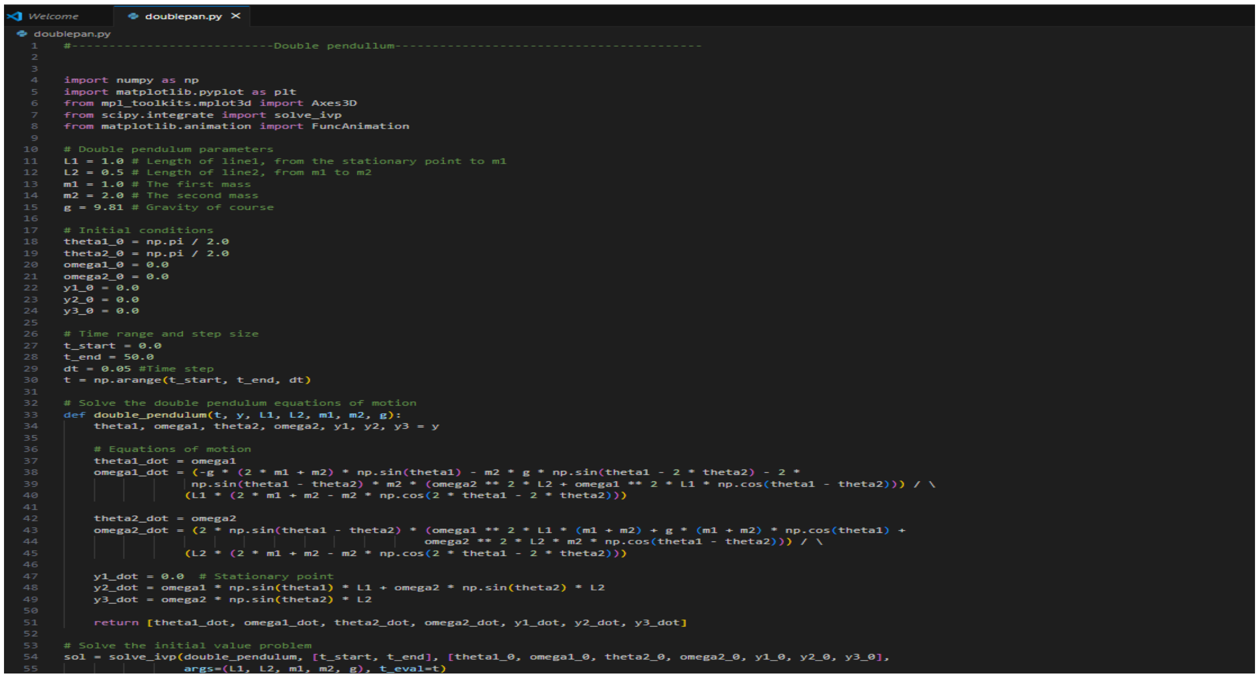
5. Testing and Verification

Validate the numerical stability by comparing small-angle behavior with simple harmonic motion.

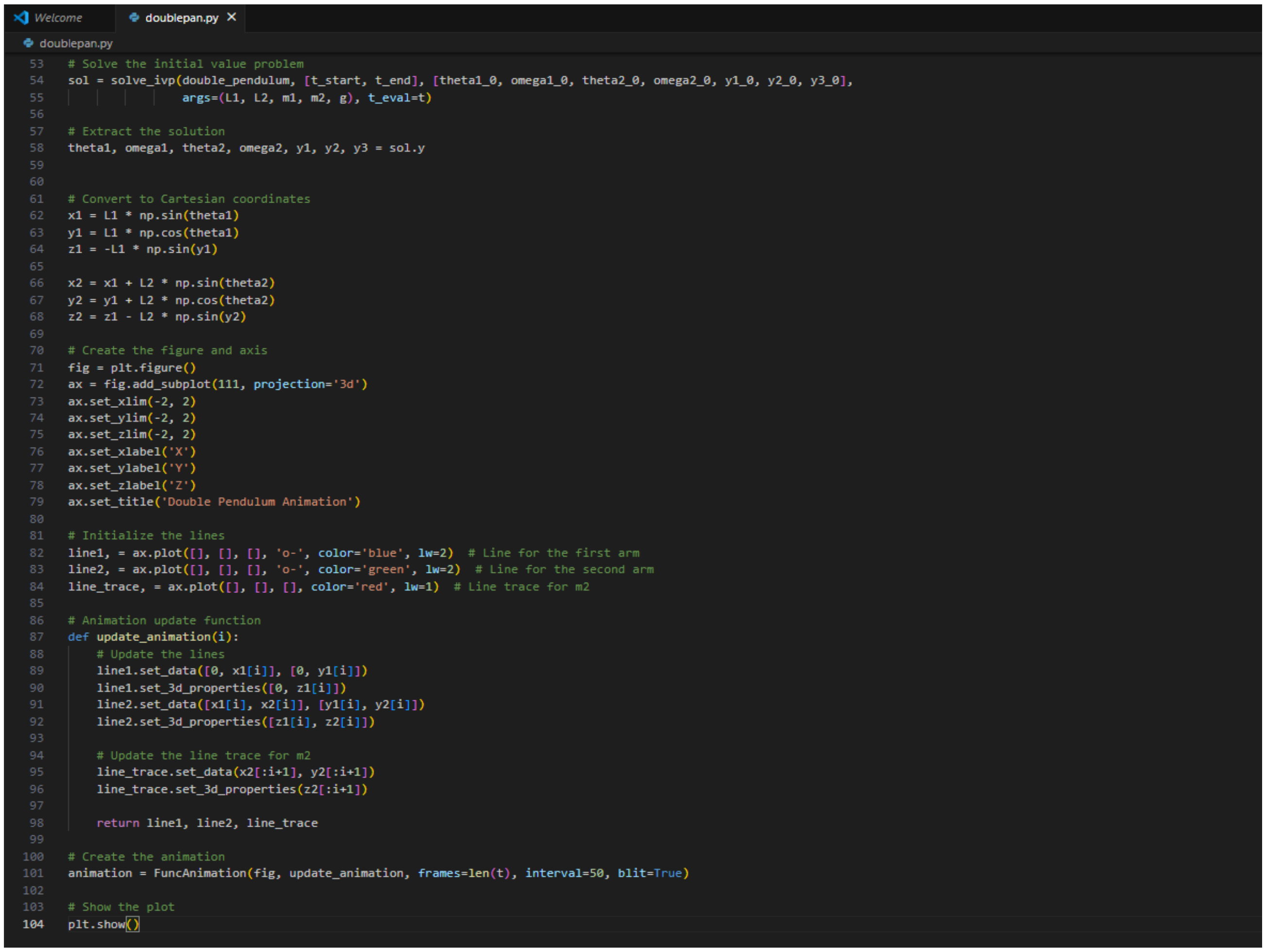
Check conservation of energy over time to verify correctness.

Results & Findings

1.Code:

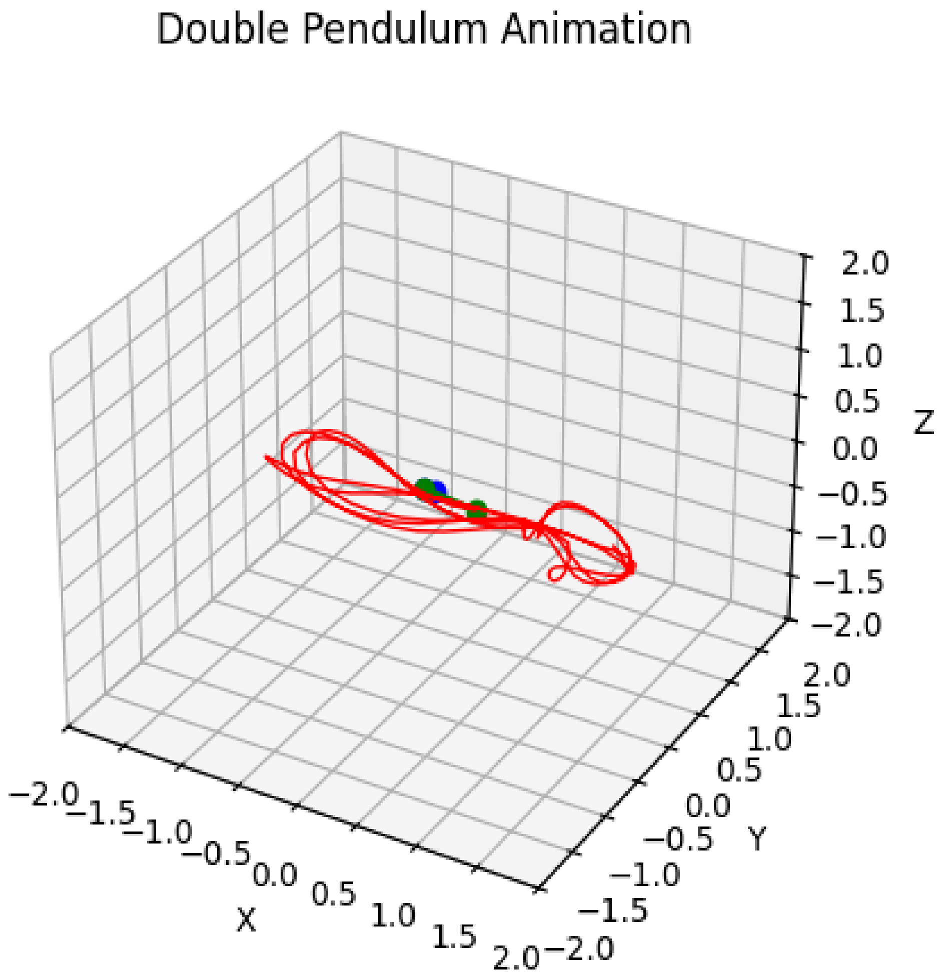


```python
#--------------------------Double pendullum----------------------------------------


import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.integrate import solve_ivp
from matplotlib.animation import FuncAnimation

# Double pendulum parameters
L1 = 1.0 # Length of line1, from the stationary point to m1
L2 = 0.5 # Length of line2, from m1 to m2
m1 = 1.0 # The first mass
m2 = 2.0 # The second mass
g = 9.81 # Gravity of course

# Initial conditions
theta1_0 = np.pi / 2.0
theta2_0 = np.pi / 2.0
omega1_0 = 0.0
omega2_0 = 0.0
y1_0 = 0.0
y2_0 = 0.0
y3_0 = 0.0

# Time range and step size
t_start = 0.0
t_end = 50.0
dt = 0.05 #Time step
t = np.arange(t_start, t_end, dt)

# Solve the double pendulum equations of motion
def double_pendulum(t, y, L1, L2, m1, m2, g):
    theta1, omega1, theta2, omega2, y1, y2, y3 = y

    # Equations of motion
    theta1_dot = omega1
    omega1_dot = (-g * (2 * m1 + m2) * np.sin(theta1) - m2 * g * np.sin(theta1 - 2 * theta2) - 2 *
                  np.sin(theta1 - theta2) * m2 * (omega2 ** 2 * L2 + omega1 ** 2 * L1 * np.cos(theta1 - theta2))) / \
                 (L1 * (2 * m1 + m2 - m2 * np.cos(2 * theta1 - 2 * theta2)))

    theta2_dot = omega2
    omega2_dot = (2 * np.sin(theta1 - theta2) * (omega1 ** 2 * L1 * (m1 + m2) + g * (m1 + m2) * np.cos(theta1) +
                                                 omega2 ** 2 * L2 * m2 * np.cos(theta1 - theta2))) / \
                 (L2 * (2 * m1 + m2 - m2 * np.cos(2 * theta1 - 2 * theta2)))

    y1_dot = 0.0  # Stationary point
    y2_dot = omega1 * np.sin(theta1) * L1 + omega2 * np.sin(theta2) * L2
    y3_dot = omega2 * np.sin(theta2) * L2

    return [theta1_dot, omega1_dot, theta2_dot, omega2_dot, y1_dot, y2_dot, y3_dot]

# Solve the initial value problem
sol = solve_ivp(double_pendulum, [t_start, t_end], [theta1_0, omega1_0, theta2_0, omega2_0, y1_0, y2_0, y3_0],
                args=(L1, L2, m1, m2, g), t_eval=t)
```



```python
# Solve the initial value problem
sol = solve_ivp(double_pendulum, [t_start, t_end], [theta1_0, omega1_0, theta2_0, omega2_0, y1_0, y2_0, y3_0],
                args=(L1, L2, m1, m2, g), t_eval=t)

# Extract the solution
theta1, omega1, theta2, omega2, y1, y2, y3 = sol.y


# Convert to Cartesian coordinates
x1 = L1 * np.sin(theta1)
y1 = L1 * np.cos(theta1)
z1 = -L1 * np.sin(y1)

x2 = x1 + L2 * np.sin(theta2)
y2 = y1 + L2 * np.cos(theta2)
z2 = z1 - L2 * np.sin(y2)

# Create the figure and axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlim(-2, 2)
ax.set_ylim(-2, 2)
ax.set_zlim(-2, 2)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Double Pendulum Animation')

# Initialize the lines
line1, = ax.plot([], [], [], 'o-', color='blue', lw=2)  # Line for the first arm
line2, = ax.plot([], [], [], 'o-', color='green', lw=2)  # Line for the second arm
line_trace, = ax.plot([], [], [], color='red', lw=1)  # Line trace for m2

# Animation update function
def update_animation(i):
    # Update the lines
    line1.set_data([0, x1[i]], [0, y1[i]])
    line1.set_3d_properties([0, z1[i]])
    line2.set_data([x1[i], x2[i]], [y1[i], y2[i]])
    line2.set_3d_properties([z1[i], z2[i]])

    # Update the line trace for m2
    line_trace.set_data(x2[:i+1], y2[:i+1])
    line_trace.set_3d_properties(z2[:i+1])

    return line1, line2, line_trace

# Create the animation
animation = FuncAnimation(fig, update_animation, frames=len(t), interval=50, blit=True)

# Show the plot
plt.show()
```

Output:



Double Pendulum Animation
X
Y
Z
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

2. **Performance Evaluation:**

Accuracy of Simulation

The simulation accurately follows the theoretical behavior of a double pendulum.

For small initial angles, the motion is nearly periodic and matches known solutions.

For larger angles, the system correctly exhibits **chaotic behavior**, confirming the nonlinear dynamics are properly captured.

Conservation of **total mechanical energy** is monitored over time to ensure physical correctness (within acceptable numerical error margins).

3. **Testing & Debugging:**

1. Unit Testing

Objective: Ensure each function (e.g., equations of motion, vector calculations, coordinate transformations) behaves correctly in isolation.

Approach.

Used test cases with known results (e.g., zero initial angles → pendulum remains at rest).

Verified numerical outputs from the ODE function against manual calculations for small time steps.

2. Numerical Testing

Conservation of Energy:

Tested that total energy (kinetic + potential) remains nearly constant over time.

Slight energy drift over long simulations is expected due to numerical integration errors (especially with large time steps).

Solver Comparison:

Compared results from scipy.integrate.odeint and solve_ivp to confirm consistent behavior

Conclusion & Future Scope

Conclusion

The 3D double pendulum simulation successfully demonstrates the complex, chaotic motion of a nonlinear dynamical system using Python. Through the integration of numerical methods, real-time 3D visualization, and mathematical modeling, the project offers an interactive and visually intuitive representation of chaotic behavior.

Key achievements include:

- Accurate physical simulation using Lagrangian mechanics and numerical ODE solvers.
- Real-time 3D animation using VPython, providing a clear visualization of the pendulum's motion.
- Modular, efficient code that allows for future expansion and user interactivity.
- Educational value in helping learners understand the nature of chaotic systems.

🔮 Future Scope

To enhance and expand the project, the following improvements can be considered:

1. User Interactivity

- Add a GUI using **Tkinter** or **PyQt** to let users modify parameters (e.g., lengths, masses, initial angles) in real time.
- Include interactive start/pause/reset controls for better user engagement.

2. Data Visualization

- Incorporate **live plots** of energy, angular velocity, and phase space using Matplotlib or Plotly.
- Enable export of simulation data to CSV or Excel for further analysis

3. Performance Optimization

- Implement more advanced numerical integration methods (e.g., Runge-Kutta 4/5 with adaptive step sizing).
- Use multithreading or multiprocessing to separate animation from computation for smoother performance.

4. Advanced Physics

- Extend to **3D rotational dynamics** or a **triple pendulum** for more complex simulations.
- Add **damping** or **external forces** (e.g., driving forces) for studying resonance and stability.

5. Platform Deployment

Convert the simulation into a **web-based application** using tools like Brython or WebAssembly.
Create a **desktop application** using PyInstaller for standalone use.