

PARAM SOMANE
Homework 0 - April 1, 2025

Instructor: Albert Chern

Homework 0: A Chaotic Double Pendulum Simulation in Python and VPython

1. Overview and Motivation

This document presents a double pendulum simulation implemented in Python with VPython for 3D visualization. We outline the physical equations of motion, the numerical methods used (fourth-order Runge–Kutta), and the code structure. A pastel color palette is introduced for an aesthetically pleasing visualization. The results demonstrate chaotic behavior characteristic of a double pendulum, and the system serves as a visual tool for educational and research purposes in advanced dynamics.

2. Introduction

The double pendulum is a well-known physical system that exhibits chaotic behavior for certain initial conditions. It consists of two rods and two masses, with the second mass hanging from the first. Despite its deceptively simple construction, the double pendulum can display highly sensitive dependence on initial conditions, making it a quintessential example of chaos in classical mechanics.

The primary objectives of this simulation are:

- To demonstrate chaotic motion via real-time 3D rendering.
- To provide a straightforward Python program that can be easily modified for educational or experimental purposes.
- To showcase a pastel color scheme that softens the visual appearance of the standard double pendulum demonstration.

3. Equations of Motion

Denote:

- $\theta_1(t)$: Angle of the first (upper) pendulum from the vertical.
- $\theta_2(t)$: Angle of the second (lower) pendulum from the vertical.
- $\omega_1 = \dot{\theta}_1$, $\omega_2 = \dot{\theta}_2$: Angular velocities.
- m_1, m_2 : Masses of the two bobs.
- L_1, L_2 : Lengths of the two rods.
- g : Gravitational acceleration.

The classical equations for a planar double pendulum in a gravitational field are given in Figure 1 below.

$$\begin{aligned}\dot{\theta}_1 &= \omega_1, & \dot{\theta}_2 &= \omega_2, \\ \dot{\omega}_1 &= \frac{-g(2m_1 + m_2) \sin \theta_1 - m_2 g \sin(\theta_1 - 2\theta_2) - 2m_2 \sin(\theta_1 - \theta_2)(\omega_2^2 L_2 + \omega_1^2 L_1 \cos(\theta_1 - \theta_2))}{L_1(2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))}, \\ \dot{\omega}_2 &= \frac{2 \sin(\theta_1 - \theta_2) \left(\omega_1^2 L_1 (m_1 + m_2) + g(m_1 + m_2) \cos \theta_1 + \omega_2^2 L_2 m_2 \cos(\theta_1 - \theta_2) \right)}{L_2(2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))}.\end{aligned}$$

Figure 1: Equations of motion for a planar double pendulum.

4. Numerical Integration

We employ the fourth-order Runge–Kutta (RK4) method to integrate the system in small time steps Δt :

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{f}(\mathbf{y}_n, t_n), \\ \mathbf{k}_2 &= \mathbf{f}(\mathbf{y}_n + \tfrac{1}{2}\Delta t \mathbf{k}_1, t_n + \tfrac{1}{2}\Delta t), \\ \mathbf{k}_3 &= \mathbf{f}(\mathbf{y}_n + \tfrac{1}{2}\Delta t \mathbf{k}_2, t_n + \tfrac{1}{2}\Delta t), \\ \mathbf{k}_4 &= \mathbf{f}(\mathbf{y}_n + \Delta t \mathbf{k}_3, t_n + \Delta t), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{\Delta t}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4),\end{aligned}$$

where $\mathbf{y} = [\theta_1, \omega_1, \theta_2, \omega_2]$ and \mathbf{f} encapsulates the equations of motion in Section 3.

5. Implementation in Python and VPython

Code Listing: Below is the core script, `double_pendulum_simulation.py`, which implements the double pendulum with a custom pastel palette.

```

1  #!/usr/bin/env python3
2  import math
3  import numpy as np
4  from vpython import (
5      canvas, vector, color, sphere, cylinder,
6      rate, distant_light, local_light, box
7  )
8
9  def hex_to_rgbnorm(hex_str):
10     """Convert a hex color string to a normalized RGB vector for VPython."""
11     hex_str = hex_str.lstrip('#')
12     r = int(hex_str[0:2], 16) / 255.0
13     g = int(hex_str[2:4], 16) / 255.0
14     b = int(hex_str[4:6], 16) / 255.0
15     return vector(r, g, b)
16
17  # Pastel color palette
18  BACKGROUND_PASTEL_HEX = '#FFD0F5' # Lavender blush for a light background
19  PIVOT_COLOR_HEX = '#EE82EE' # Violet

```

```
20 ROD1_COLOR_HEX = '#8F8788' # Pastel pink
21 ROD2_COLOR_HEX = '#8F8788' # Pastel pink
22 MASS1_COLOR_HEX = '#9370DB' # Medium purple
23 MASS2_COLOR_HEX = '#DA70D6' # Orchid
24 MASS2_TRAIL_COLOR_HEX = '#DA70D6' # Light orchid
25 FLOOR_COLOR_HEX = '#D3D3D3' # Light gray for the floor
26
27 def double_pendulum_derivs(y, params):
28     """Derivatives of the planar double pendulum."""
29     theta1, omega1, theta2, omega2 = y
30     m1, m2, L1, L2, g = params
31
32     sin1 = np.sin(theta1)
33     sin2 = np.sin(theta2)
34     sin12 = np.sin(theta1 - theta2)
35     cos12 = np.cos(theta1 - theta2)
36
37     denom = 2*m1 + m2 - m2 * np.cos(2*theta1 - 2*theta2)
38
39     alpha1 = (
40         -g*(2*m1 + m2)*sin1
41         - m2*g*np.sin(theta1 - 2*theta2)
42         - 2*sin12*m2*(omega2**2*L2 + omega1**2*L1*cos12)
43     ) / (L1 * denom)
44
45     alpha2 = (
46         2*sin12 * (
47             omega1**2*L1*(m1 + m2)
48             + g*(m1 + m2)*np.cos(theta1)
49             + omega2**2*L2*m2*cos12
50         )
51     ) / (L2 * denom)
52
53     return np.array([omega1, alpha1, omega2, alpha2], dtype=float)
54
55 def rk4_step(y, dt, derivs_func, params):
56     """One RK4 integration step."""
57     k1 = derivs_func(y, params)
58     k2 = derivs_func(y + 0.5*dt*k1, params)
59     k3 = derivs_func(y + 0.5*dt*k2, params)
60     k4 = derivs_func(y + dt*k3, params)
61     return y + (dt/6.0)*(k1 + 2*k2 + 2*k3 + k4)
62
63 def run_simulation():
64     # Physical parameters
65     m1, m2 = 2.0, 1.0
66     L1, L2 = 1.5, 1.5
```

```
67 g = 9.81
68 params = (m1, m2, L1, L2, g)
69
70 # Initial conditions
71 theta1_0, omega1_0 = 1.2, 0.0
72 theta2_0, omega2_0 = 2.1, 0.0
73 y = np.array([theta1_0, omega1_0, theta2_0, omega2_0], dtype=float)
74
75 dt = 0.0005
76 sim_duration = 40.0
77 N_SUBSTEPS = 10
78 steps = int(sim_duration / dt / N_SUBSTEPS)
79
80 # 3D scene
81 scene = canvas(
82     width=1280,
83     height=720,
84     center=vector(0, 1.0, 0),
85     background=hex_to_rgbnorm(BACKGROUND_PASTEL_HEX),
86     fov=0.0075
87 )
88
89 # Custom lights
90 scene.lights = []
91 distant_light(direction=vector(1, 1, 1), color=color.white)
92 local_light(pos=vector(-2, 3, 2), color=vector(0.7, 0.7, 0.7))
93 local_light(pos=vector(2, -3, 3), color=vector(0.5, 0.5, 0.5))
94
95 floor = box(
96     pos=vector(0, -2.5, 0),
97     size=vector(5, 0.05, 5),
98     color=hex_to_rgbnorm(FLOOR_COLOR_HEX),
99     opacity=0.2
100 )
101
102 pivot = vector(0, 1.5, 0)
103 pivot_sphere = sphere(
104     pos=pivot,
105     radius=0.015,
106     color=hex_to_rgbnorm(PIVOT_COLOR_HEX),
107     shininess=0.6
108 )
109
110 rod1 = cylinder(
111     pos=pivot,
112     axis=vector(0, 0, 0),
113     radius=0.03,
```

```
114     color=hex_to_rgbnorm(ROD1_COLOR_HEX),
115     shininess=0.6
116 )
117 rod2 = cylinder(
118     pos=pivot,
119     axis=vector(0, 0, 0),
120     radius=0.03,
121     color=hex_to_rgbnorm(ROD2_COLOR_HEX),
122     shininess=0.6
123 )
124
125 mass1 = sphere(
126     pos=vector(0, 0, 0),
127     radius=0.1*math.sqrt(2),
128     color=hex_to_rgbnorm(MASS1_COLOR_HEX),
129     shininess=0.6,
130     make_trail=False
131 )
132 mass2 = sphere(
133     pos=vector(0, 0, 0),
134     radius=0.1,
135     color=hex_to_rgbnorm(MASS2_COLOR_HEX),
136     shininess=0.6,
137     make_trail=True,
138     trail_radius=0.01,
139     trail_color=hex_to_rgbnorm(MASS2_TRAIL_COLOR_HEX),
140     retain=5000
141 )
142
143 mass2.clear_trail()
144
145 rod1.length = L1
146 rod2.length = L2
147
148 CAPTURE_FRAMES = False
149 frame_count = 0
150
151 # Main loop
152 for i in range(steps):
153     # RK4 substeps
154     for _ in range(N_SUBSTEPS):
155         y = rk4_step(y, dt, double_pendulum_derivs, params)
156
157     rate(200)
158
159     theta1, omega1, theta2, omega2 = y
160     x1 = L1 * np.sin(theta1)
```

```
161     y1 = -L1 * np.cos(theta1)
162     x2 = x1 + L2 * np.sin(theta2)
163     y2 = y1 - L2 * np.cos(theta2)
164
165     pos1 = vector(x1, y1, 0) + pivot
166     pos2 = vector(x2, y2, 0) + pivot
167
168     rod1.pos = pivot
169     rod1.axis = pos1 - pivot
170     rod2.pos = pos1
171     rod2.axis = pos2 - pos1
172
173     mass1.pos = pos1
174     mass2.pos = pos2
175
176     try:
177         mass2.trail_object.opacity = 0.5
178     except Exception:
179         pass
180
181     if CAPTURE_FRAMES:
182         scene.capture(f"frame{i:04d}.png")
183         frame_count += 1
184
185     print("Simulation complete.")
186
187 if __name__ == "__main__":
188     run_simulation()
```

Listing 1: Double Pendulum Simulation with Pastel Palette

Key Implementation Details:

- (i) *Pastel Palette*: All color definitions are in HEX, converted to normalized RGB for VPython.
- (ii) *Trail Visualization*: Only the second mass (`mass2`) has a trail to highlight the complexity of its motion.
- (iii) *Frame Capture*: If `CAPTURE_FRAMES` is set to `True`, each rendered frame is saved as a PNG image for creating animations.
- (iv) *Performance Tuning*: The `dt` (time step) and `N_SUBSTEPS` can be adjusted for finer or coarser simulation detail.

6. Usage and Customization

Running the Code: To run this simulation:

```
python double_pendulum.py
```

A new browser window will open, displaying the real-time 3D animation of the double pendulum.

Changing Parameters: Key parameters include masses (m_1 , m_2), rod lengths (L_1 , L_2), gravitational acceleration (g), and initial angles (θ_1 , θ_2). These can be modified directly in `run_simulation()` to explore different dynamical regimes.

Capturing Frames: If you wish to create a video, set `CAPTURE_FRAMES = True`. This will save each frame as `frame0000.png`, `frame0001.png`, etc. You can then convert them into a video (e.g., using `ffmpeg`).

7. Results and Observations

The double pendulum exhibits extreme sensitivity to initial conditions. Small changes in angles, rod lengths, or masses can result in dramatically different trajectories. Over longer simulation times, the path of the second mass often fills a significant portion of the accessible phase space, illustrating chaotic behavior.

8. Conclusion

We presented a Python-based double pendulum simulation with a focus on readability, extensibility, and visual appeal. Students and researchers in classical mechanics or chaos theory can modify the code to investigate various phenomena, such as energy transfer, Lyapunov exponents, or synchronization in coupled pendulums.

References

- [1] Guckenheimer, J. and Holmes, P. (2013). *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer.
- [2] Strogatz, S. (2018). *Nonlinear Dynamics and Chaos with Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press.
- [3] Sherwood, B. and others. (2025). *VPython Library: Real-time 3D Graphics for Python*. <https://vpython.org>