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.

$$
\begin{aligned}
\dot\theta_1 &= \omega_1, \quad \dot\theta_2 = \omega_2,\\
\dot\omega_1 &= \frac{-g(2m_1+m_2)\sin\theta_1 - m_2 g\sin(\theta_1-2\theta_2) - 2m_2\sin(\theta_1-\theta_2)\big(\omega_2^2 L_2 + \omega_1^2 L_1\cos(\theta_1-\theta_2)\big)}{L_1\big(2m_1+m_2-m_2\cos(2\theta_1-2\theta_2)\big)},\\
\dot\omega_2 &= \frac{2\sin(\theta_1-\theta_2)\Big(\omega_1^2 L_1(m_1+m_2) + g(m_1+m_2)\cos\theta_1 + \omega_2^2 L_2\, m_2\cos(\theta_1-\theta_2)\Big)}{L_2\big(2m_1+m_2-m_2\cos(2\theta_1-2\theta_2)\big)}.
\end{aligned}
$$

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.py`, which implements the double pendulum with a custom pastel palette.

```python
#!/usr/bin/env python3
import math
import numpy as np
from vpython import (
    canvas, vector, color, sphere, cylinder,
    rate, distant_light, local_light, box
)

def hex_to_rgbnorm(hex_str):
    """Convert a hex color string to a normalized RGB vector for VPython."""
    hex_str = hex_str.lstrip('#')
    r = int(hex_str[0:2], 16) / 255.0
    g = int(hex_str[2:4], 16) / 255.0
    b = int(hex_str[4:6], 16) / 255.0
    return vector(r, g, b)

# Pastel color palette
BACKGROUND_PASTEL_HEX = '#FFF0F5' # Lavender blush for a light background
PIVOT_COLOR_HEX = '#EE82EE' # Violet
```

```
ROD1_COLOR_HEX = '#8F8788' # Pastel pink
ROD2_COLOR_HEX = '#8F8788' # Pastel pink
MASS1_COLOR_HEX = '#9370DB' # Medium purple
MASS2_COLOR_HEX = '#DA70D6' # Orchid
MASS2_TRAIL_COLOR_HEX = '#DA70D6' # Light orchid
FLOOR_COLOR_HEX = '#D3D3D3' # Light gray for the floor

def double_pendulum_derivs(y, params):
    """Derivatives of the planar double pendulum."""
    theta1, omega1, theta2, omega2 = y
    m1, m2, L1, L2, g = params

    sin1 = np.sin(theta1)
    sin2 = np.sin(theta2)
    sin12 = np.sin(theta1 - theta2)
    cos12 = np.cos(theta1 - theta2)

    denom = 2*m1 + m2 - m2 * np.cos(2*theta1 - 2*theta2)

    alpha1 = (
        -g*(2*m1 + m2)*sin1
        - m2*g*np.sin(theta1 - 2*theta2)
        - 2*sin12*m2*(omega2**2*L2 + omega1**2*L1*cos12)
    ) / (L1 * denom)

    alpha2 = (
        2*sin12 * (
            omega1**2*L1*(m1 + m2)
            + g*(m1 + m2)*np.cos(theta1)
            + omega2**2*L2*m2*cos12
        )
    ) / (L2 * denom)

    return np.array([omega1, alpha1, omega2, alpha2], dtype=float)

def rk4_step(y, dt, derivs_func, params):
    """One RK4 integration step."""
    k1 = derivs_func(y, params)
    k2 = derivs_func(y + 0.5*dt*k1, params)
    k3 = derivs_func(y + 0.5*dt*k2, params)
    k4 = derivs_func(y + dt*k3, params)
    return y + (dt/6.0)*(k1 + 2*k2 + 2*k3 + k4)

def run_simulation():
    # Physical parameters
    m1, m2 = 2.0, 1.0
    L1, L2 = 1.5, 1.5
```

```
    g = 9.81
    params = (m1, m2, L1, L2, g)

    # Initial conditions
    theta1_0, omega1_0 = 1.2, 0.0
    theta2_0, omega2_0 = 2.1, 0.0
    y = np.array([theta1_0, omega1_0, theta2_0, omega2_0], dtype=float)

    dt = 0.0005
    sim_duration = 40.0
    N_SUBSTEPS = 10
    steps = int(sim_duration / dt / N_SUBSTEPS)

    # 3D scene
    scene = canvas(
        width=1280,
        height=720,
        center=vector(0, 1.0, 0),
        background=hex_to_rgbnorm(BACKGROUND_PASTEL_HEX),
        fov=0.0075
    )

    # Custom lights
    scene.lights = []
    distant_light(direction=vector(1, 1, 1), color=color.white)
    local_light(pos=vector(-2, 3, 2), color=vector(0.7, 0.7, 0.7))
    local_light(pos=vector(2, -3, 3), color=vector(0.5, 0.5, 0.5))

    floor = box(
        pos=vector(0, -2.5, 0),
        size=vector(5, 0.05, 5),
        color=hex_to_rgbnorm(FLOOR_COLOR_HEX),
        opacity=0.2
    )

    pivot = vector(0, 1.5, 0)
    pivot_sphere = sphere(
        pos=pivot,
        radius=0.015,
        color=hex_to_rgbnorm(PIVOT_COLOR_HEX),
        shininess=0.6
    )

    rod1 = cylinder(
        pos=pivot,
        axis=vector(0, 0, 0),
        radius=0.03,
```

```
        color=hex_to_rgbnorm(ROD1_COLOR_HEX),
        shininess=0.6
    )
    rod2 = cylinder(
        pos=pivot,
        axis=vector(0, 0, 0),
        radius=0.03,
        color=hex_to_rgbnorm(ROD2_COLOR_HEX),
        shininess=0.6
    )

    mass1 = sphere(
        pos=vector(0, 0, 0),
        radius=0.1*math.sqrt(2),
        color=hex_to_rgbnorm(MASS1_COLOR_HEX),
        shininess=0.6,
        make_trail=False
    )
    mass2 = sphere(
        pos=vector(0, 0, 0),
        radius=0.1,
        color=hex_to_rgbnorm(MASS2_COLOR_HEX),
        shininess=0.6,
        make_trail=True,
        trail_radius=0.01,
        trail_color=hex_to_rgbnorm(MASS2_TRAIL_COLOR_HEX),
        retain=5000
    )

    mass2.clear_trail()

    rod1.length = L1
    rod2.length = L2

    CAPTURE_FRAMES = False
    frame_count = 0

    # Main loop
    for i in range(steps):
        # RK4 substeps
        for _ in range(N_SUBSTEPS):
            y = rk4_step(y, dt, double_pendulum_derivs, params)

        rate(200)

        theta1, omega1, theta2, omega2 = y
        x1 = L1 * np.sin(theta1)
```

```python
        y1 = -L1 * np.cos(theta1)
        x2 = x1 + L2 * np.sin(theta2)
        y2 = y1 - L2 * np.cos(theta2)

        pos1 = vector(x1, y1, 0) + pivot
        pos2 = vector(x2, y2, 0) + pivot

        rod1.pos = pivot
        rod1.axis = pos1 - pivot
        rod2.pos = pos1
        rod2.axis = pos2 - pos1

        mass1.pos = pos1
        mass2.pos = pos2

        try:
            mass2.trail_object.opacity = 0.5
        except Exception:
            pass

        if CAPTURE_FRAMES:
            scene.capture(f"frame{i:04d}.png")
            frame_count += 1

    print("Simulation complete.")

if __name__ == "__main__":
    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`

Acknowledgments: OpenAI's ChatGPT (model gpt-4o) was utilized to assist in adding comments and docstrings to Python functions, explain code implemented by me and ensure its correctness and alignment with equations of motion, and help debug code errors. The outputs from this AI model were modified with major changes. I actively reviewed, tested, and adjusted the generated code and explanations to reflect my own understanding.