Coded Chaos: A Symplectic Double Pendulum Simulator in Python

Scott Marino

June 9, 2021

1 Introduction

I have built a physical simulator for a double pendulum system that takes user input. The program makes use of only stock modules in python such as `numpy`, `matplotlib`, and `threading`, and requires no additional hardware to run.

2 Methods

2.1 The Symplectic Algorithm

We learned about Euler's method in class for doing numerical integration of simple differential equations, however it breaks down for systems that are inherently chaotic after short time intervals as small errors in computing add up. A *symplectic* method is one which keeps the simulation conforming with its real counterpart by preserving a known quantity, in this case the energy of the system using the Hamiltonian. In mathematical terms, the symplectic Euler's method will preserve the state flow through pq (momentum-generalized coordinate) space, which by Lioville's theorem means the system's energy is conserved.

ODE solving algorithms such as `scipy.odeint` do not have symplectic methods in them, and while there are more complicated algorithms like certain variations of Runge-Kutta I went with a symplectic Euler's method because (a) I wanted to build on the framework we started in week 10 of class, and (b) if I could work out the Hamiltonian math it would be much more simpler to program.

The Lagrangian of a double pendulum system with generalized coordinates as angles α and β is

$$\mathcal{L} = \frac{1}{2}(m_1 + m_2)\ell_1^2\dot{\alpha}^2 + \frac{1}{2}m_2\ell_2^2\dot{\beta}^2 + m_2\ell_1\dot{\alpha}\ell_2\dot{\beta}\cos(\alpha - \beta) + (m_1 + m_2)g\ell_1\cos\alpha + m_2 g\ell_2\cos\beta$$

with α being the angle between the first pendulum and the origin and β being the second pendulum with the vertical axis. We can use the definition of the

Hamiltonian and Hamilton's canonical equations to derive the Hamiltonian in terms of the momentum and angles (math being skipped for brevity):

$$\mathcal{H} = \frac{m_2\ell_2 p_\alpha^2 + (m_1+m_2)\ell_1^2 p_\beta^2 - 2m_2\ell_1\ell_2 p_\alpha p_\beta \cos(\alpha-\beta)}{2m_2\ell_1^2\ell_2^2\left[m_1 + m_2\sin^2(\alpha-\beta)\right]} - (m_1+m_2)g\ell_1\cos\alpha - m_2 g\ell_2\cos\beta$$

Taking advantage of Hamilton's canonical equations and the explicit Euler's method:

$$y_{n+1} = y_n + hf(y_n, t_n)$$

yields a symplectic Euler's method for each coordinate q

$$q_{n+1} = q_n + h\frac{\partial}{\partial p}H(p_n, q_{n+1})$$
$$p_{n+1} = p_n - h\frac{\partial}{\partial q}H(p_n, q_{n+1})$$

where each of these derivatives represents the approximate slope of that variable over a small step value h. These derivatives were done by hand and will be excluded from this paper to meet the page limits, as they were incredibly tedious, but will result in a more stable simulation in the long run.

2.2 User Input

The user input was done via a user input thread, similar to that from Homework 8, problem 5. The `while` loop has functionality to detect whether the user is entering all of the parameters, only both of the angles, or the `<Run>` command to begin the simulation. The real-time user feedback is fed into an animation function that displays the user's parameter choices, and when they are ready to proceed the thread ends the animation by signaling a frame generator function to cease. There is also user input that is taken after the prompt animation and before the simulation for whether or not the user would like to save pictures of their pendulum's trail and angle graphs.

3 Results

The threaded user input allows for a seamless visualization of a user's simulation parameters, which adds a kind of game-like quality for the user to be able to continually input their parameters until they see a pendulum they are interested in simulating. The user also has an option to save a drawing of the pendulum's trail compared to its initial position on an eps file, which gives the program greater utility in the creation of LaTeX papers or publications about related fields like chaos. These eps files are easily distinguished by custom names generated for initial conditions, so someone could continually run the program to

get an aggregate of data without having to manually change the `plt.savefig` parameters, and such files are in an easily sortable format.

It is difficult to certify the *exact* validity of the simulation without a comparable physical system (i.e. an actual frictionless double pendulum in a vacuum with the exact same parameters), however it is possible to plug in systems and see if their simulated behaviour aligns with intuition about similar systems.

3.1 Small Initial Differences and Long Term Deviations

Taking a starting example of $\alpha = 120°$ and $\beta = 120°$, with lengths and masses all at default of 1. The simulation produces the following trail and angle graph:

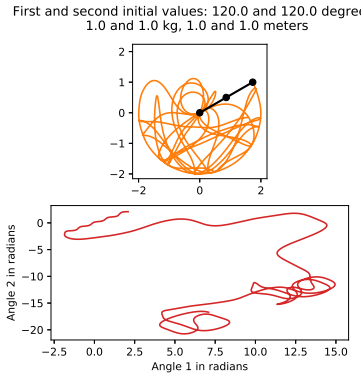

Figure 1: $\alpha = 120°, \beta = 120°$

If we were to vary α by only one degree and change no other parameters, we get the following:

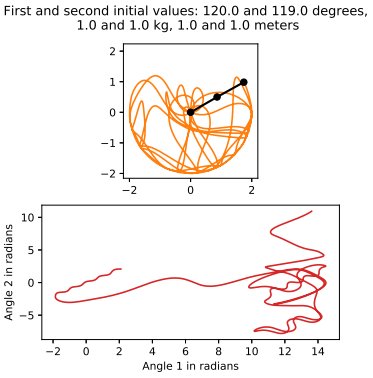

Figure 2: $\alpha = 120°, \beta = 119°$

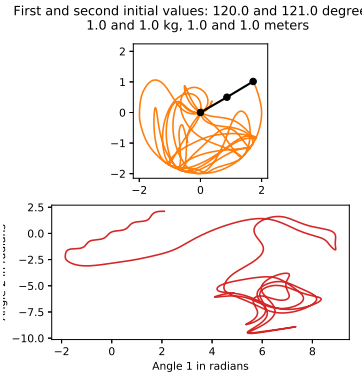

Figure 3: $\alpha = 120°, \beta = 121°$

We can clearly see from each plot's angular graphs that the pendulum angles

undergo small and steady oscillations with respect to each other at first, then once the pendulum makes its first big sweep through α, the once small initial differences compound and the simulations diverge swiftly and drastically.

3.2 Stability of Motion Between Similar Shaped Systems of Different Scales

Another interesting situation is the comparison between pendulum setups that have similar parameter scales, but drastically different effects due to the constant gravitational acceleration programmed into the Hamiltonian. The two systems below both have starting angles 220°, however one has masses 1 and lengths 0.5 and 0.3 meters, while the other masses 100 and lengths 50, 30 meters.

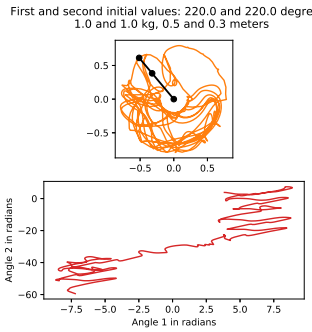

Figure 4: Small scale system

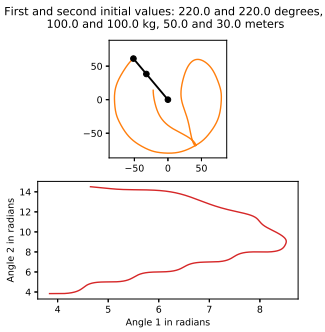

Figure 5: Large scale system

The pendulum in Fig. 4 experiences a greater force of gravity compared to its parameters and spirals quickly out of control across its possible range, while Fig. 5 barely makes a full swing in the 20 second period and stays relatively stable throughout. This aligns with intuition, as one can imagine the relatively slow-seeming motion of a pendulum swung from a 59 story building compared to one in a lab with heavy weights.

I want to note that in simulating Fig. 4 I approached the computational limit of my simulator, as the `matplotlib` animation modules are only good up to a certain frame rate, and become a ceiling for the differential equation solving algorithm systems moving considerably fast such.

3.3 Approach to Steady Oscillations for Small Angles

A solid way to examine the validity of the numerical integration algorithm is to plug in a system with known behaviour, such as one undergoing small oscillations with steady frequencies.

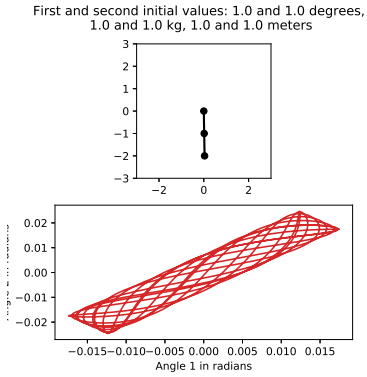

Figure 6: All parameters equal to 1

The contained and periodic shape of the angular motion shows the relative stability of these small oscillations, and gives me confidence in my algorithm's ability to accurately simulate real physical systems.