Solving and Visualizing Chaotic Systems with Range-Kutta 4th Order Integration

Gowshik Rajan 60307390

Abstract

I will simulate the evolution of the Lorenz equations and other dynamical systems over time. The differential equations will be integrated over time with the Range-Kutta 4th Order Method and therefore trajectories of initial conditions can be plotted. The resulting plot is expected to be a visualization of the Lorenz attractor. The main focus of chaotic dynamical systems is how slight variation can compound to more drastic changes over the course of time. This idea will be explored with the analysis of multiple initial conditions with methods such as computing Lyapunov Exponents. In addition to the Lorenz Equations, I will also explore other equations in dynamic systems such as the double pendulum, and apply the necessary analysis on those equations as well. The underlying methods can be used to study more chaotic systems, and therefore can be applied in fields like electric circuits, finance/economics, neuroscience, and weather modeling - the original motivation for Lorenz Equations.

Table Of Contents

1 Introduction

Chaotic Systems are systems that exhibit chaotic behavior governed by deterministic laws or equations (Wikipedia contributors, 2025a). These systems are often characterized by extreme sensitivity to initial conditions, and their evolution throughout appears to be quite random. This phenomena is famously summarized as the "Butterfly Effect" - used with a metaphor that a butterfly can cause a hurricane in the other side of the world (Wikipedia contributors, 2025b).

These types of equations have many applications, mostly used in nonlinear dynamics to model phenomena in physics, engineering, economics, and so on. This report explores two famous chaotic systems: **Lorenz system** and **double pendulum**. The Lorenz system originated from the simplified Navier-Stokes equation for atmospheric convection and hence is applied in weather modeling and fluid dynamics. The double pendulum is a mechanical system that consists of two pendulums or bobs attached to each other which exhibits extreme sensitivity to initial conditions, which is a textbook example in study of classical mechanics.

These equations can be analyzed numerically and therefore visualized using the tools of Python. Through computing, we can demonstrate their chaotic nature, and gain insight on how unpredictable behavior arises and evolves through time.

1.1 Recap on RK4 Integration

Consider a first-order differential equation with the dependent variable y and independent variable x. Then, for an initial-value problem:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y), \quad y(x_o) = y_0$$

With the Range-Kutta 4th Order method, the equations can be solved numerically. Now, the variable x can be discretized into steps, each with step size h with initial value x_0. Then, y_i can be solved numerically with:

$$y_{i+1} = y_i + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$
$$x_{i+1} = x_i + h$$

where

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)$$
$$k_4 = f\left(x_i + h, y_i + hk_3\right)$$

2 Lorenz System

2.1 Introduction

The Lorenz System is modeled by system of Ordinary Differential Equations originally developed to model simplified atmospheric convection (Lorenz, 1963). They were first studied by meteorologist Edward Lorenz, who noticed that upon numerically solving these equations, a small change to the initial conditions produced very different results over time , even though they appeared to evolve similarly at the beginning (Wikipedia contributors, 2025a). However, upon using precisely the same initial conditions, the equations provided the exact same trajectories. This discovery led to the emergence of chaos theory and study of chaotic systems. It is often characterized by its iconic butterfly-shaped plot, which is often used in conjunction with the Butterfly Effect. Even with the unpredictability of the solutions, they tend to orbit around the Lorenz attractor producing the iconic plot.

The equations are a result of the simplification of Saltzman's equations, which are in-turn simplified forms of Navier-Stokes equation for convection (Lorenz, 1963). The model is represented by three differential equations which describe the evolution of: rate of convection x, horizontal temperature y and vertical temperature z variation.

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y - x)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho - z) - y\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy - \beta z\end{aligned}$$

All the constants are dimensionless quantities which can be related to physically model convection (Wikipedia contributors, 2025b). σ is the *Prandtl number*, which is the ratio of viscous and thermal diffusion rates. ρ is the *Rayleigh number* where $\rho = Gr_x\sigma$, and Gr_x is the *Grashof number*. This term describes relation between buoyancy and viscous force on fluid. β is related to the physical height of the fluid layer.

2.2 Method of Solution

Consider a state vector $\mathbf{X}(t)$ representing the state variables x, y, and z. Then, the Lorenz Equations can be modeled as a vector function:

$$\mathbf{X}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} \quad \text{and} \quad f(\mathbf{X}) = \begin{bmatrix} \sigma(y - x) \\ x(\rho - z) - y \\ xy - \beta z \end{bmatrix} \tag{1}$$

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = f(\mathbf{X})$$

Here, the equation is analogous to a first-order differential equation. Now it can be used to solve the initial value problem $\mathbf{X}(0) = \mathbf{X}_0$.

Using numerical methods, it is now possible to perform time integration using Range-Kutta 4th Order Method:

$$\mathbf{X}_{i+1} = \mathbf{X}_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2}$$

The solution relies on numerical methods as there is no reliable way to analytically solve these equations.

2.3 Chaotic Nature: Multiple Trajectories

From the plots given below, we can see the very unique nature of the solutions in phase space. This is because the Lorenz System has a *strange attractor*. An *attractor* is where all points in the neighborhood of the attractor converge to overtime (Taylor, 2011). A strange attractor has a fractal structure where no point on the orbit of the attractor is visited twice. Therefore, a trajectory is not self-intersecting and continues on forever. Hence, it can be said that the system has a solution set of infinite area but zero volume.

In a chaotic system, it is known that solutions or trajectories of the system are very sensitive to the initial conditions. Consequently, this means that a small change in initial conditions can result in very different results over time. However, the difference between trajectories in chaotic systems with strange attractors, like in Lorenz system, continuously diverge and converge. This is because, since it's a chaotic system, the

solutions must diverge exponentially, but since it has an attractor, they must also converge to some attracting set. This causes a cycle of folding and stretching in the orbits, which in-turn gives the attractor a fractal structure.

The *Lyapunov Exponent* is used to measure the sensitivity of a chaotic system. This is important for identifying chaos as its directly related to the nature of the system. Suppose there exists two trajectories P and Q. Let the initial difference between those trajectories be d_0 at time t_0. Then, the distance d between those trajectories at time $t > t_0$ will deviate at an exponential rate (Taylor, 2011).

$$d = d_0 \cdot e^{\lambda(t-t_0)}$$

where λ is considered to be the Lyapunov Exponent. Solving for λ,

$$\lambda = \frac{1}{t - t_0} \ln \left| \frac{d}{d_0} \right| \tag{3}$$

It can be immediately inferred that $\lambda > 0$ means chaotic behavior and $\lambda \leq 0$ means non-chaotic behavior. The details of the calculation are given below in the next section.

2.4 Translating into Code and Results

Here, I use Python code to numerically solve the equation. The main package used is NumPy, which is great for representing large arrays. I will use it to represent a list of points in the *phase space* where each entry in the array corresponds to increasing time. The phase space is where we plot x, y, and z on their own, but as a parametric function of time. For this example, I use the values $\sigma = 10$, $\rho = 28$, $\beta = \frac{8}{3}$, which were the original values used by Lorenz.

The equations in 2.2 (1) can be written in Python as follows:

```
from typing import Callable
import numpy as np
from numpy._typing import NDArray

def rk4(X_i: NDArray, dt: float, f: Callable) -> NDArray:
    k1 = f(X_i)
    k2 = f(X_i + dt*k1/2)
    k3 = f(X_i + dt*k2/2)
    k4 = f(X_i + dt*k3)
    X_n = X_i + dt/6 * (k1 + 2*k2 + 2*k3 + k4)
return X_n

def lorenz(X: NDArray) -> NDArray:
    rho = 28
    sigma = 10
    beta = 8/3
    x, y, z = X

    dxdt = sigma*(y-x)
    dydt = x*(rho-z) - y
    dzdt = x*y - beta*z

    return np.array([dxdt, dydt, dzdt])
```

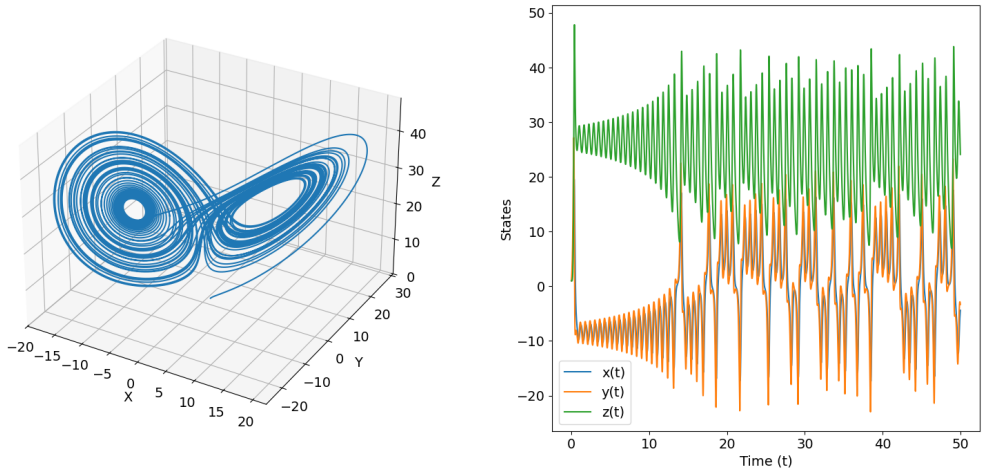
The main part of the code uses the initial state vector $\mathbf{X}_0 = [1, 1, 1]$. Note that `dt` is equivalent to h. Then, an array of N elements will be created with each entry corresponding to t_i.

```
X0 = np.array([1, 1, 1])
t_n = 50              # final time in seconds
dt = 0.001
N = int(t_n/dt)       # no. of elements/points

X = np.empty((N+1, 3))
X[0] = X0

for i in range(0, N):
    X[i+1] = rk4(X[i], dt, lorenz)
```

Here, the `X` array is an array of the states of the system. This can be plotted with `matplotlib` to create the plot of the phase space of the Lorenz attractor as follows:



(a) Phase space of Lorenz System

(b) Time series of Lorenz System

Figure 1: Plots of Lorenz System states

For multiple trajectories, as discussed above, it is possible to visualize using `matplotlib`. With a small displacement [0.1, 0, 0], the resultant plot of the evolution

of solution is produced:

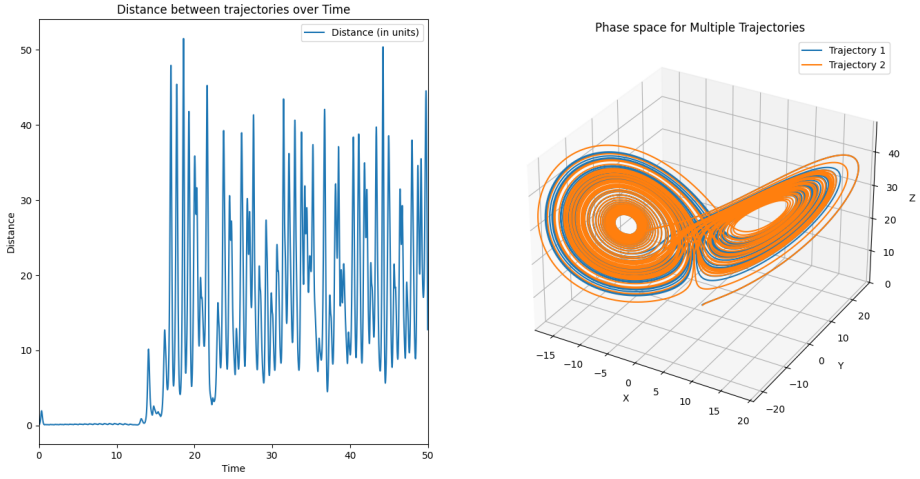


Figure 2: Evolution of Lorenz System - Distance between trajectories and phase space plot of trajectories over time

However, as seen above, the difference between trajectories have non-periodic oscillation - which means that the trajectories aren't always diverging exponentially. So, in calculation of Lyapunov Exponents, it has to be done in a time step t_{step}. Then, we proceed to take the average Lyapunov exponent from the calculations.

The basic algorithm used is given by Taylor (2011). In essence, the distance between the trajectories are calculated, beginning with distance d_0. After each time step t_{step}, the new distance d is calculated and λ is calculated with equation (3). Then, the second orbit is readjusted such that the difference is d_0 but in direction of d. This process is repeated n times and then the average Lyapunov Exponent can be computed. This algorithm is implemented in code to give the value of the largest Lyapunov exponent from the Lyapunov spectrum, which is the value that matters in determining the chaotic nature of the system. Calculating with time step $t_{step} = 0.1$ and 10000 iterations results in the average Lyapunov Exponent $\lambda \approx 0.89936$. This is close to the agreed upon value of $\lambda \approx 0.9056$ (Viswanath, 1998).

2.5 Summary

To sum, the Range-Kutta 4th order method is used to solve the first order differential equations of the Lorenz system. Using the tools of numerical methods and Python, visualizations of the solutions to Lorenz system has been plotted. The chaotic behavior is explained and shown using Python including the calculation of the Lyapunov Exponent to show chaos. Since the Lorenz equations is derived from atmospheric convection equations, it has applications for weather modeling and the methods used here can be applied to more complex differential equations to accurately model the physical world. Using these methods, in the next section, the double pendulum will be solved and visualized.

3 Double Pendulum

3.1 Introduction

The methods presented in this section is almost identical to the previous section, only with a few adjustments. The double pendulum is a textbook problem studied in classical mechanics. It consists of two pendulums separated by different lengths and masses. The equation of motions will be derived from the Euler-Lagrange equation as given below.

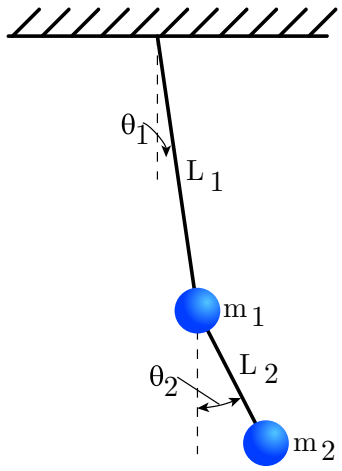


Figure 3: Double Pendulum Diagram

Note. A diagram of double pendulum by JabberWok, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1601029 Wikimedia Commons contributors, 2006

Similar to the Lorenz system, this is an example of a chaotic system, which means its also sensitive to initial conditions when exhibiting chaotic behavior. It is also governed by a system of differential equations as given below.

3.2 Brief Derivation

To avoid getting too much into the details, only a brief derivation is provided to provide some background. For detailed derivation refer to Gonzalez (2006).
Consider two bobs with mass m_1, m_2 suspended from strings lengths L_1, L_2 respectively. Let their angular displacements be θ_1, θ_2 and the positions of the centers of the bob be

given by (x_1, y_1) and (x_2, y_2) respectively. Then,

$$x_1 = L_1 \sin\theta_1 \qquad y_1 = -L_1 \cos\theta_1 \tag{4}$$

$$x_2 = L_1 \sin\theta_1 + L_2 \sin\theta_2 \qquad y_1 = -L_1 \cos\theta_1 - L_2 \cos\theta_2 \tag{5}$$

Let $\dot{x}$ represent the time derivative $\dot{x} = \frac{\mathrm{d}x}{\mathrm{d}t}$ and let $\Delta\theta = \theta_1 - \theta_2$. The Lagrangian is then given by the difference between the kinetic and potential energy.

$$\begin{aligned} L &= T - U \\ &= \frac{1}{2} m_1 L_1^2 \dot{\theta}_1^2 + m_1 g L_1 \cos\theta_1 + \frac{1}{2} m_2 \left(L_1^2 \dot{\theta}_1^2 + L_2^2 \dot{\theta}_2^2 + 2 L_1 L_2 \dot{\theta}_1 \dot{\theta}_2 \cos\Delta\theta \right) \\ &+ m_2 g (L_1 \cos\theta_1 + L_2 \cos\theta_2) \end{aligned}$$

Now, the Euler-Lagrange equation can be used to get the equations of motion.

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i} = 0, \quad i = 1, 2$$

Plugging in L and solving for each θ_i,

$$(m_1 + m_2) L_1^2 \ddot{\theta}_1 + m_2 L_1 L_2 \ddot{\theta}_2 \cos\Delta\theta = -(m_1 + m_2) g L_1 \sin\theta_1 - m_2 L_1 L_2 \dot{\theta}_1^2 \sin\Delta\theta \tag{6}$$

$$m_2 L_1 L_2 \ddot{\theta}_1 \cos\Delta\theta + m_2 L_2^2 \ddot{\theta}_2 = m_2 L_1 L_2 \dot{\theta}_1^2 \sin\Delta\theta - m_2 g L_2 \sin\theta_2 \tag{7}$$

Without approximations, these equations are impossible to solve analytically. To simulate the chaotic behavior, the equations have to be solved numerically.

3.3 Method of Solution

It can be seen that (6) and (7) are a system of equations of $\ddot{\theta}_1$ and $\ddot{\theta}_2$. In order to get the equations to solve numerically, $\ddot{\theta}_1$ and $\ddot{\theta}_2$ must be isolated:

$$\begin{aligned} \ddot{\theta}_1 = &\frac{-m_2 (L_1 \dot{\theta}_1^2 \sin\Delta\theta \cos\Delta\theta + L_2 \dot{\theta}_2^2 \sin\Delta\theta)}{m_1 L_1 + m_2 L_2 \sin\Delta\theta} \\ &+ \frac{g(-(m_1 + m_2)\sin\theta_1 + m_2 \cos\Delta\theta \sin\theta_2)}{m_1 L_1 + m_2 L_2 \sin\Delta\theta} \end{aligned}$$

$$\ddot{\theta}_2 = \frac{(m_1+m_2)L_1\dot{\theta}_1^2 \sin\Delta\theta + m_2 L_2 \dot{\theta}_2^2 \sin\Delta\theta\cos\Delta\theta}{m_1 L_2 + m_2 L_2 \sin\Delta\theta} + \frac{(m_1+m_2)g(\sin\theta_1\cos\Delta\theta - \sin\theta_2)}{m_1 L_2 + m_2 L_2 \sin\Delta\theta}$$

Let $\omega_1 = \dot{\theta}_1$ and $\omega_2 = \dot{\theta}_2$. Let the right sides of the above equations be functions $\alpha_1(\theta_1, \theta_2, \omega_1, \omega_2)$ and $\alpha_2(\theta_1, \theta_2, \omega_1, \omega_2)$. That results in this system of differential equations

$$\begin{aligned}
\dot{\theta}_1 &= \omega_1 \\
\dot{\theta}_2 &= \omega_2 \\
\dot{\omega}_1 &= \alpha_1(\theta_1, \theta_2, \omega_1, \omega_2) \\
\dot{\omega}_2 &= \alpha_2(\theta_1, \theta_2, \omega_1, \omega_2)
\end{aligned}$$

Now, just like before, it is possible to put this into a state vector

$$\mathbf{Y} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \omega_1 \\ \omega_2 \end{bmatrix} \quad \text{and} \quad \phi(\mathbf{Y}) = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \alpha_1(\theta_1, \theta_2, \omega_1, \omega_2) \\ \alpha_2(\theta_1, \theta_2, \omega_1, \omega_2) \end{bmatrix}$$

$$\frac{\mathrm{d}\mathbf{Y}}{\mathrm{d}t} = \phi(\mathbf{Y})$$

Now, it is possible to apply RK4 by discretizing $\mathbf{Y}$:

$$\mathbf{Y}_{i+1} = \mathbf{Y}_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{8}$$

3.4 Translating into Code and Results

Once again, I implemented the above with Python and NumPy for array operations. First, I implemented the double pendulum as a data class:

```
class DoublePendulum:
    def __init__(self, state_vector, masses, lengths):
        self.state_vector = state_vector
        self.m1 = masses[0]
        self.m2 = masses[1]
        self.l1 = lengths[0]
        self.l2 = lengths[1]
```

Implementing the ϕ function and Range-Kutta 4th Order:

```
def phi(dp_state: DoublePendulum, Y: NDArray) -> NDArray:
    m1, m2 = dp_state.m1, dp_state.m2
    L1, L2 = dp_state.l1, dp_state.l2
    theta1, theta2, omega1, omega2 = Y
    dtheta = theta1 - theta2
    Y_n = np.empty(4)

    Y_n[0] = omega1
    Y_n[1] = omega2
    Y_n[2] = ( -m2*(L1*omega1**2*sin(dtheta)*cos(dtheta) + L2*omega2**2*
                                        sin(dtheta)) + g*(-(m1+m2)*sin(
                                        theta1) + m2*cos(dtheta)*sin(
                                        theta2)) ) / (m1*L1 + m2*L1*sin(
                                        dtheta)**2)
    Y_n[3] = ( (m1+m2)*L1*omega1**2*sin(dtheta) + m2*L2*omega2**2*sin(
                                        dtheta)*cos(dtheta) + (m1+m2)*g*
                                        (sin(theta1)*cos(dtheta) - sin(
                                        theta2)) ) / (m1*L2 + m2*L2*sin(
                                        dtheta)**2)

    return Y_n

def rk4_step(y_i: float, dt: float, f: Callable):
```

```
    k1 = f(y_i)
    k2 = f(y_i + dt*k1/2)
    k3 = f(y_i + dt*k2/2)
    k4 = f(y_i + dt*k3)
    y_n = y_i + dt/6 * (k1 + 2*k2 + 2*k3 + k4)
    return y_n
```

Then, I created the function to solve for given pendulum state:

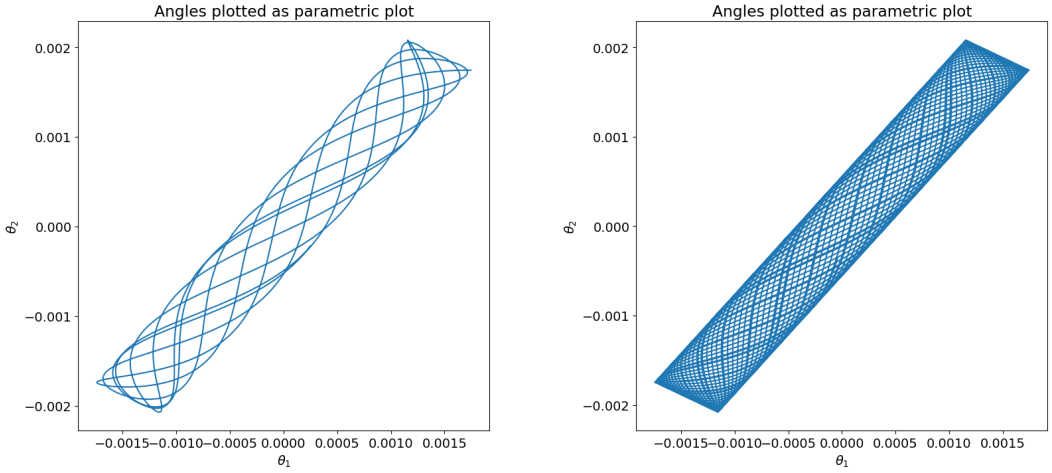
```
def solve_pendulum(dp_state: DoublePendulum, dt, N_steps) -> NDArray:
    Y_i = np.empty((N_steps+1, 4))
    Y_i[0] = dp_state.state_vector
    partial_phi = partial(phi, dp_state)
    for i in range(N_steps):
        Y_i[i+1] = rk4_step(Y_i[i], dt, partial_phi)
    return Y_i
```

Now, I provide the initial conditions:

```
theta1_0, theta2_0 = 0.1, 0.1
omega1_0, omega2_0 = 0, 0
g = 9.81
initial_state_vector = np.array([theta1_0, theta2_0, omega1_0, omega2_0])
dp_state = DoublePendulum(
    state_vector=initial_state_vector,
    masses=[10, 20],
    lengths=[10, 20]
)
dt = 0.01
t_n = 50    # in seconds
N = int(t_n/dt)

Y = solve_pendulum(dp_state, dt, N)
```

Note that the values are initially provided in degrees and converted to radians. I will include both the radians and degrees while mentioning them in the text. Now, plotting θ_1 and θ_2 from `Y`, where $\theta_1 = \theta_2 = 0.1°(\approx 0.00174\,\text{rad})$:



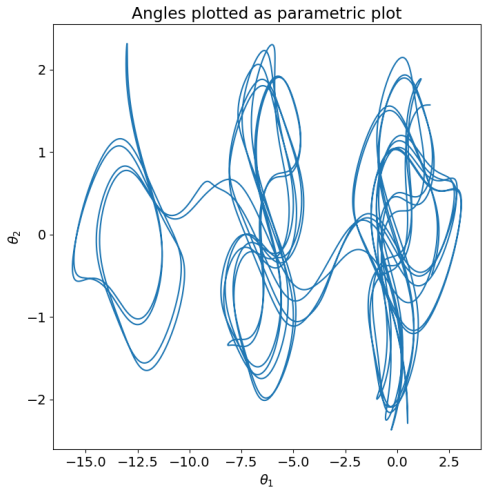
(a) Phase space ($t = 50$)

(b) Phase space ($t = 250$)

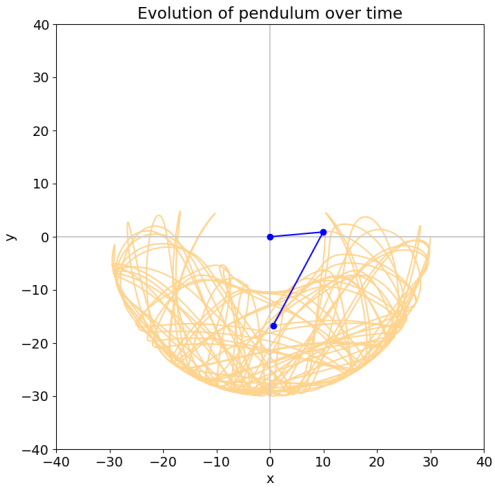
Figure 4: Plots of Partial Phase Spaces with $\theta_1 = 0.1°, \theta_2 = 0.1°$

While the above figures may imply that the double pendulum is stable, this is special behavior. That is, double pendulum is conditionally chaotic. For instance, consider the initial conditions with $\theta_1 = \theta_2 = 90°(\approx 1.57\,\text{rad})$.

Plugging those values into the code given above, it is possible to once again plot the phase space. Using the position equations of each pendulums from (4) and (5), I plotted the pendulum itself, with the final position as well as the trace of all positions. These initial conditions yield chaotic graphs as seen below:



(a) Phase space ($t = 250$)



(b) Phase space ($t = 250$)

Figure 5: Plots of Phase space and Time Evolution with $\theta_1 = 90°, \theta_2 = 90°$

To quantitatively predict when a given initial angle, Lyapunov Exponents can once again be of utility. The algorithm is pretty much unchanged, but the trajectories in phase space are now 4-dimensional (includes ω as well).

Since this double pendulum system seems to exert both chaotic and non-chaotic behavior, it is expected that there are different Lyapunov exponents based on the initial conditions. Therefore, Lyapunov Exponents for each varying initial conditions can be calculated. The initial parameters varied here are θ_1 and θ_2, from 0 to 2π each. Plotting this in Python with a Matplotlib color-map, we get the diagram below.

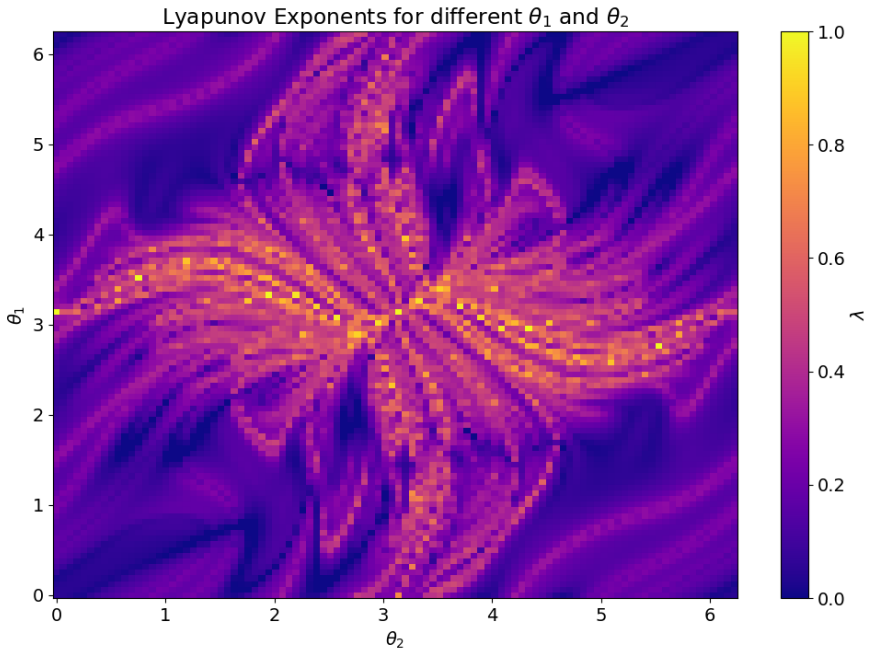


Figure 6: Lyapunov Exponents for varying θ_1, θ_2

From the diagram, it is possible to see the different Lyapunov exponents for each initial condition. Note that the graph exhibits some symmetry to $\theta_1 = \pi$ and $\theta_2 = \pi$. The colormap is edited such that all $\lambda \leq 0$ is purple and $\lambda \geq 1$ is yellow. These regions indicate whether the given initial condition will result in a chaotic trajectory.

3.5 Summary

To sum, the tools of numerical methods and Python have been utilized to solve the equations of motion for the double pendulum. Basic analysis on the chaotic nature of the system, with the Lyapunov Exponents, have also been done with the help of these tools. This is a textbook example in classical mechanics, and hence more complicated and convoluted differential equations can be solved in the study of classical mechanics.

4 Conclusion

As shown above, the powerful nature of numerical methods is very apparent. Despite the chaotic nature of these dynamical systems, they are still able to be accurately predicted by the tools offered by numerical methods and computing, even though there is no analytical solution for the above. The field of chaos theory is not only limited to meteorology and physics, but a variety of other fields. In advanced study of many fields, there arises a need for complex dynamical systems. The fields applicable include, but are not limited to, computer science, biology, economics, engineering, finance, and so on. In economics, it is shown that chaos theory has applications in modeling economies during crises like COVID-19, and chaos theory is shown to provide some understandings in stock market analysis (Klioutchnikov et al., 2017). An important application of chaos theory is cryptography, where techniques are used in hash-functions, pRNGs, encryption, and so on. It is still an ever-growing field with more applications to come. The applications listed above and more can be found in Wikipedia contributors (2025a).

References

Gonzalez, G. (2006). Single and double plane pendulum.

Klioutchnikov, I., Sigova, M., & Beizerov, N. (2017). Chaos theory in finance [6th International Young Scientist Conference on Computational Science, YSC 2017, 01-03 November 2017, Kotka, Finland]. *Procedia Computer Science*, *119*, 368–375. https://doi.org/https://doi.org/10.1016/j.procs.2017.11.196

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, *20*(2), 130–141. https://doi.org/10.1175/1520-0469(1963)020⟨0130:DNF⟩2.0.CO;2

Taylor, R. (2011). Attractors: Nonstrange to chaotic. *SIAM Undergraduate Research Online*, *4*, 72–80. https://doi.org/10.1137/10S01079X

Viswanath, D. (1998). *Lyapunov exponents from random fibonacci sequences to the lorenz equations*. Cornell University.

Wikimedia Commons contributors. (2006). Double pendulum [Accessed: 2025-04-12].

Wikipedia contributors. (2025a). Chaos theory — Wikipedia, the free encyclopedia [[Online; accessed 11-April-2025]].

Wikipedia contributors. (2025b). Lorenz system — Wikipedia, the free encyclopedia [[Online; accessed 11-April-2025]].