University of Utah, Mathematics Department

CHAOTIC MOTION IN DOUBLE PENDULUM SYSTEMS

A Final Computing Project

Charles Godfrey Tannon Warnick Henry Whiting

Abstract

This abstract was generated by Gemini [3] based on this papers content. This project investigates the nonlinear and chaotic dynamics of a double pendulum system. Starting from the Lagrangian formulation, we derive the coupled second-order differential equations governing the system's motion. These equations are then transformed into a first-order state-space representation suitable for numerical integration. Using a fourth-order Runge–Kutta (RK4) method implemented in Julia, we simulate the motion and analyze the system's sensitivity to initial conditions. Numerical experiments demonstrate infinitesimal perturbations in the initial angles lead to exponentially divergent trajectories. The results illustrate how a physically simple, deterministic system can exhibit rich, unpredictable behavior.

December 11, 2025

Contents

List of Figures

1. Introduction

A single pendulum is a classical example of simple harmonic motion. When released from a height, the pendulum will swing periodically and consistently. By simply adding a second pendulum at the end of the first, the system transforms into a classical example of chaotic motion. Even though these two systems are governed by the same physical laws of motion and only being acted upon by one force (gravity), a double pendulum is *heavily* dependent on initial conditions. To analyze the system's behavior, we approximate the solution using the fourth-order Runge-Kutta method."

2. Mathematical Model

We must first derive the equations of motion. Because the system involves multiple degrees of freedom, the Lagrangian formulation of mechanics is significantly more efficient than the Newtonian approach [2].

2.1. System Setup and Geometry

We consider a system of two point masses, m_1 and m_2, connected by massless rigid rods of length l_1 and l_2. The system is confined to a 2D plane and acted upon by a uniform gravitational field g pointing downwards. The system can be visualized in Figure 2.1 on this page. Note: unlike in Figure 2.1, the mathematical (and numerical) system will not intersect with itself or other objects.

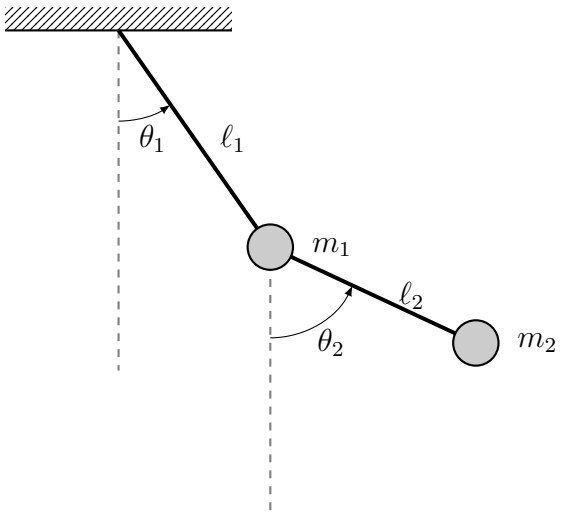

Figure 2.1: Double pendulum system

The system we will construct mathematically, and which is depicted in Figure 2.1, is a simple pendulum. A compound pendulum can be similarly derived by just substituting the positions of the point masses with the positions of the centers of mass [1].

We define the generalized coordinates as the angles θ_1 and θ_2, measured from the vertical axis. The origin is the pivot point of the first pendulum. This will be easier to compute mathematically, and therefore numerically than the Cartesian system, although it is easier to visualize as such.

$$x_1 = \ell_1 \sin\theta_1 \tag{2.1}$$
$$y_1 = -\ell_1 \cos\theta_1 \tag{2.2}$$
$$x_2 = \ell_1 \sin\theta_1 + \ell_2 \sin\theta_2 \tag{2.3}$$
$$y_2 = -\ell_1 \cos\theta_1 + \ell_2 \cos\theta_2 \tag{2.4}$$

2.2. The Lagrangian Formulation

The Lagrangian $\mathcal{L}$ is defined as the difference between the kinetic energy (T) and the potential energy (V) of the system:

$$\mathcal{L} = T - V$$

2.2.1. Kinetic Energy (T)

The kinetic energy of the system is the sum of the kinetic energies of the masses:

$$T = \frac{1}{2}m_1 v_1^2 + \frac{1}{2}m_2 v_2^2$$

Calculating the velocities squared ($v^2 = \dot{x}^2 + \dot{y}^2$):

$$\begin{aligned} v_1^2 &= (l_1\dot{\theta}_1)^2 \\ v_2^2 &= \dot{x}_2^2 + \dot{y}_2^2 \\ &= l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1 l_2 \dot{\theta}_1\dot{\theta}_2(\sin\theta_1 \sin\theta_2 + \cos\theta_1\cos\theta_2) \\ &= l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1 l_2 \dot{\theta}_1\dot{\theta}_2\cos(\theta_1 - \theta_2) \end{aligned}$$

Thus, the total kinetic energy is:

$$T = \frac{1}{2}(m_1 + m_2)l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2 + m_2 l_1 l_2 \dot{\theta}_1\dot{\theta}_2\cos(\theta_1 - \theta_2) \tag{2.5}$$

2.2.2. Potential Energy (V)

Assuming potential energy is zero at $y = 0$, we have:

$$\begin{aligned} V &= m_1 g y_1 + m_2 g y_2 \\ &= -(m_1 + m_2)g l_1 \cos\theta_1 - m_2 g l_2 \cos\theta_2 \end{aligned} \tag{2.6}$$

2.3. Equations of Motion

Applying the Euler-Lagrange equation for each coordinate θ_i:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_i}\right) - \frac{\partial \mathcal{L}}{\partial \theta_i} = 0$$

Solving these derivatives yields a system of two, non-linear second-order differential equations.

For θ_1:

$$(m_1 + m_2)l_1\ddot{\theta}_1 + m_2 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + m_2 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + (m_1 + m_2)g \sin\theta_1 = 0 \tag{2.7}$$

For θ_2:

$$m_2 l_2 \ddot{\theta}_2 + m_2 l_1 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + m_2 g \sin\theta_2 = 0 \tag{2.8}$$

These two equations completely describe the motion of the double pendulum. In Section 3, we will rearrange these terms to solve for $\ddot{\theta}_1$ and $\ddot{\theta}_2$ explicitly to implement the RK4 algorithm.

3. Preparing for Numerical Model

Implementing RK4 requires transforming the second-order differential equations derived in Section 2 into a system of first-order differential equations.

3.1. Matrix Formulation of Motion

We begin with the Euler-Lagrange equations (2.7) and (2.8). By grouping the angular acceleration terms $(\ddot{\theta}_1, \ddot{\theta}_2)$ on the left-hand side and the remaining velocity and potential terms on the right, we obtain the following linear system:

$$(m_1 + m_2)l_1\ddot{\theta}_1 + m_2 l_2 \cos(\theta_1 - \theta_2)\ddot{\theta}_2 = -m_2 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) - (m_1 + m_2)g \sin\theta_1, \tag{3.1}$$

$$m_2 l_1 \cos(\theta_1 - \theta_2)\ddot{\theta}_1 + m_2 l_2 \ddot{\theta}_2 = m_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) - m_2 g \sin\theta_2. \tag{3.2}$$

This system can be written in matrix form as $M(\theta)\ddot{\theta} = b(\theta, \omega)$, where $\omega_i = \dot{\theta}_i$:

$$\underbrace{\begin{pmatrix} (m_1 + m_2)l_1 & m_2 l_2 \cos(\theta_1 - \theta_2) \\ m_2 l_1 \cos(\theta_1 - \theta_2) & m_2 l_2 \end{pmatrix}}_{M(\theta)} \underbrace{\begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{pmatrix}}_{\ddot{\theta}} = \underbrace{\begin{pmatrix} -m_2 l_2 \omega_2^2 \sin(\theta_1 - \theta_2) - (m_1 + m_2)g \sin\theta_1 \\ m_2 l_1 \omega_1^2 \sin(\theta_1 - \theta_2) - m_2 g \sin\theta_2 \end{pmatrix}}_{b(\theta,\omega)}.$$

3.2. Explicit Acceleration Terms

There are many applicable methods we can utilize to solve for the angular accelerations $\ddot{\theta}$. For its ease, we will invert the mass matrix M. For a 2×2 matrix, the inverse is explicitly given by:

$$\ddot{\theta} = M^{-1}b = \frac{1}{\det M}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Defining the angle difference as $\Delta\theta = \theta_1 - \theta_2$, the determinant is $\det M = m_2 l_1 l_2 (m_1 + m_2 \sin^2 \Delta\theta)$. Carrying out the matrix multiplication yields the explicit accelerations:

$$\ddot{\theta}_1 = \frac{1}{\det M}\left[m_2 l_2 b_1 - m_2 l_2 \cos(\Delta\theta) b_2\right], \tag{3.3}$$

$$\ddot{\theta}_2 = \frac{1}{\det M}\left[-m_2 l_1 \cos(\Delta\theta) b_1 + (m_1 + m_2) l_1 b_2\right]. \tag{3.4}$$

Substituting the components of vector b (from (3.1) and (3.2)) into these expressions provides the necessary system allowing for numerical solvers such as the RK4 method we will use.

3.3. State-Space Representation

Finally, we convert the two second-order equations into a system of four first-order equations. We introduce the state vector X:

$$X = \begin{pmatrix} \theta_1 & \theta_2 & \omega_1 & \omega_2 \end{pmatrix}^T.$$

The time evolution of the system is governed by $\dot{X} = F(X)$:

$$\dot{X} = \frac{d}{dt}\begin{pmatrix} \theta_1 \\ \theta_2 \\ \omega_1 \\ \omega_2 \end{pmatrix} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \ddot{\theta}_1(\theta_1, \theta_2, \omega_1, \omega_2) \\ \ddot{\theta}_2(\theta_1, \theta_2, \omega_1, \omega_2) \end{pmatrix},$$

where $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are determined by equations (3.3) and (3.4). This vector $F(X)$ is passed directly to the RK4 algorithm.

3.4. Numerical Implementation: The RK4 Algorithm

With the system reduced to the first-order form $\dot{X} = F(X)$, we can now apply the Runge-Kutta-4 method to numerically integrate the equations of motion. This method is preferred over schemes like Euler's method, because its local truncation error is of order $O(h^4)$, providing the high precision necessary to track chaotic trajectories.

Given the vector X_n at time t_n and a time step h, the next state X_{n+1} is computed as a

weighted average of four slope estimates:

$$X_{n+1} = X_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

where the intermediate slopes k_i are defined as:

$$\begin{aligned} k_1 &= F(X_n), \\ k_2 &= F\left(X_n + \frac{h}{2}k_1\right), \\ k_3 &= F\left(X_n + \frac{h}{2}k_2\right), \\ k_4 &= F(X_n + hk_3). \end{aligned}$$

In these equations, the function F corresponds to the operator defined back in Section 3.3, which calculates the derivatives $[\omega_1, \omega_2, \ddot{\theta}_1, \ddot{\theta}_2]^T$ at each step. This algorithm ensures that the non-linearities of the double pendulum are captured accurately even over extended simulation times, although we will not be determining the limit of its stability, it is certainly much longer than our simulation time. The full Julia code implementing this solver can be found in Appendix A on page 7.

4. Results and Discussion

To verify the chaotic nature of the double pendulum, we used the RK4 method as follows: We initialized two *almost* identical double pendulum systems (sharing identical lengths l_1, l_2 and masses m_1, m_2) with initial angles differing by a microscopic perturbation, ϵ.

$$\Theta_A = (\theta_1, \theta_2) \quad \text{and} \quad \Theta_B = (\theta_1 + \epsilon, \theta_2)$$

where $\epsilon = 10^{-5}$ degrees.

Figure 4.1 on the following page illustrates the trajectories of these two systems over time. During the first few sections, the deviation caused by the perturbation appears negligible, and the two systems move in unison. However, as the system moves forward, the non-linear coupling terms in the equations of motion, specifically those derived in (3.1), cause this infinitesimal difference ϵ to propagate and grow exponentially for some time.

By the end of the simulation, the two pendulums exhibit completely uncorrelated behavior. This confirms that while the double pendulum is deterministic (fully described by Equations 3.3 and 3.4), it is practically unpredictable over long time horizons due to this extreme sensitivity.

Plotting the angular difference $\Delta\theta$ between the two systems further illustrates this chaotic dissociation. As shown in Figure 4.2 on the next page, the error does not grow linearly but exhibits rapid, unpredictable fluctuations, the peaks of which grow exponentially, a characteristic of the butterfly effect.

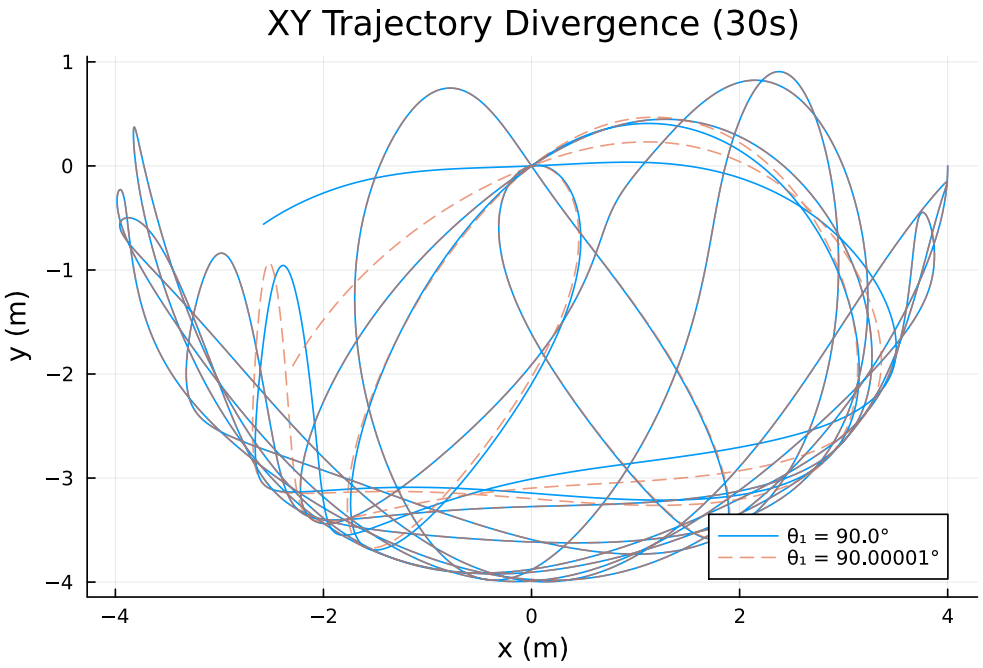

Figure 4.1: Divergence of trajectories with initial conditions differing by 10^{-5} degrees.

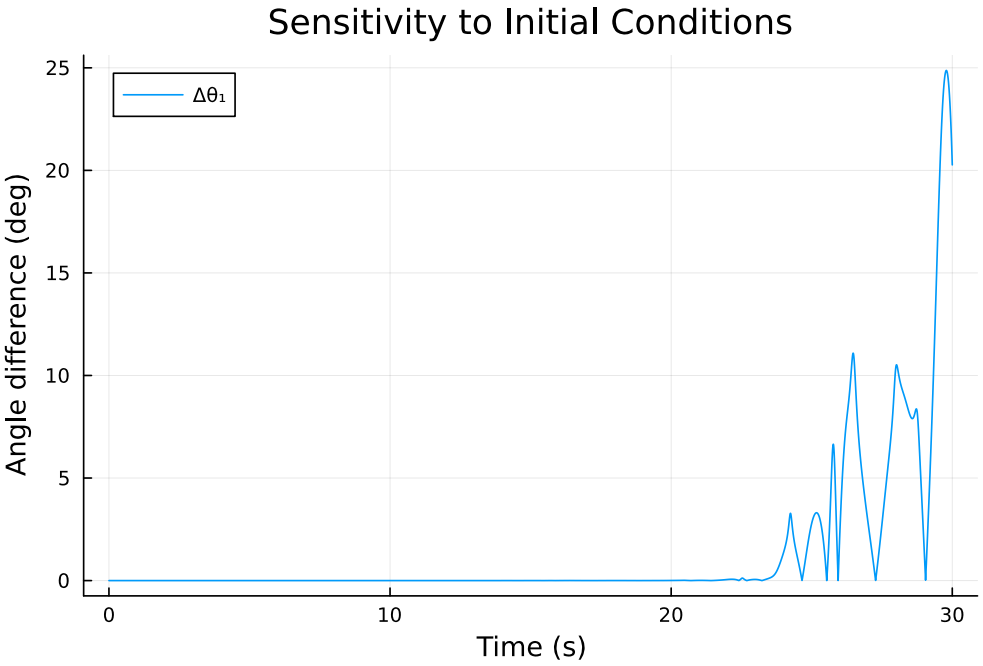

Figure 4.2: Evolution of the angular difference $\Delta\theta$ between the two systems over time.

The results validate that simple physical systems can exhibit complex, non-intuitive behavior. Future work could involve solving triple or n-pivot pendulums. This would require much higher-dimension vector fields to incorporate into the methods. Further study on n-pivot pendulums would also require analysis of whether RK4 is accurate enough for such

systems.

A. Julia Code

Here is the compilation of all Julia code needed to solve the system in this paper.

```
using Plots
gr() # Set plotting backend

# parameters
G = 9.8
lengths = [2.0, 2.0]
masses = [2.0, 2.0]
time_span = (0.0, 30.0) # 30 seconds
n_steps = 15000 # RK4 steps

# ================== RK4 Solver ================== #
function rk4(IVP, n)
    a, b = IVP.tspan
    h = (b - a)/n
    t = [a + i*h for i in 0:n]

    u0 = IVP.u0
    m = length(u0)
    u = zeros(m, n+1)
    u[:,1] .= u0

    for i in 1:n
        k1 = h * IVP.f(u[:,i], IVP.p, t[i])
        k2 = h * IVP.f(u[:,i] + k1/2, IVP.p, t[i] + h/2)
        k3 = h * IVP.f(u[:,i] + k2/2, IVP.p, t[i] + h/2)
        k4 = h * IVP.f(u[:,i] + k3, IVP.p, t[i] + h)
        u[:,i+1] = u[:,i] + (k1 + 2*k2 + 2*k3 + k4)/6
    end

    return t, u
end

# Double Pendulum ODE (returns vector)
function double_pendulum(u, p, t)
    m1, m2, l1, l2, g = p
    theta1, omega1, theta2, omega2 = u

    Delta = theta2 - theta1
    denom1 = (m1 + m2)*l1 - m2*l1*cos(Delta)^2
    denom2 = (l2/l1)*denom1

```

```
    du = zeros(4)
    du[1] = omega1
    du[2] = ( m2*l1*omega1^2*sin(Delta)*cos(Delta) +
              m2*g*sin(theta2)*cos(Delta) +
              m2*l2*omega2^2*sin(Delta) -
              (m1+m2)*g*sin(theta1) ) / denom1

    du[3] = omega2
    du[4] = ( -m2*l2*omega2^2*sin(Delta)*cos(Delta) +
              (m1+m2)*g*sin(theta1)*cos(Delta) -
              (m1+m2)*l1*omega1^2*sin(Delta) -
              (m1+m2)*g*sin(theta2) ) / denom2

    return du
end

# IVP Struct
struct ODEProblem
    f::Function
    u0::Vector{Float64}
    tspan::Tuple{Float64, Float64}
    p::Tuple
end

# Simulation Function
function simulate_pendulum(theta1_deg, theta2_deg; omega1=0.0, omega2=0.0)
    u0 = deg2rad.([theta1_deg, omega1, theta2_deg, omega2])
    p = (masses[1], masses[2], lengths[1], lengths[2], G)
    IVP = ODEProblem(double_pendulum, u0, time_span, p)
    t, u = rk4(IVP, n_steps)
    return t, u
end

# Plotting Multiple Initial Conditions
initial_conditions = [
    (90.0, 90.0),
    (91.0, 90.0),
    (90.0, 91.0)
]

plot(title="Double Pendulum Theta vs Time", xlabel="Time (s)", ylabel="Angle (deg
    )")

for (theta1, theta2) in initial_conditions
    t, u = simulate_pendulum(theta1, theta2)
    plot!(t, rad2deg.(u[1,:]), label="theta1 start=(theta1,theta2)")
    plot!(t, rad2deg.(u[3,:]), label="theta2 start=(theta1,theta2)")
```

```
end

t, u1 = simulate_pendulum(90.0, 90.0)
t, u2 = simulate_pendulum(90.00001, 90.0)
x_vals = @. lengths[1] * sin(u[1,:]) + lengths[2] * sin(u[3,:])
y_vals = @. -lengths[1] * cos(u[1,:]) - lengths[2] * cos(u[3,:])

# Plotting Delta Theta graph
plot(t, rad2deg.(abs.(u1[1,:] .- u2[1,:])), label="Delta theta1", xlabel="Time (s)", ylabel="Angle difference (deg)", title="Sensitivity to Initial Conditions")

# Plotting (x,y) graph
plot(x_vals, y_vals, xlabel="x (m)", ylabel="y (m)", title="Pendulum Trajectory", aspect_ratio=:equal, label="")
```

References

[1] Double pendulum *Wikipedia*. https://en.wikipedia.org/wiki/Double_pendulum. Accessed: 2025-12-11.

[2] John R. Taylor. *Classical Mechanics*. University Science Books, 2005.

[3] Google. *Gemini* [Large Language Model]. https://gemini.google.com. Accessed: 2025-12-11.