# An Analysis of a Double Pendulum System

## 1. Introduction

In this project, I examined the motion of a double pendulum by solving the *Euler-Lagrange* equations of the system numerically. I inspected several initial value problems and analyzed the impact of changing parameters and the limitations of *scipy.odeint*'s solver with this particular system. The main point of the project was to improve my coding with *Python* by solving the ordinary differential equations that I found with *Lagrange's* method. I took the opportunity to learn how to plot the motion and how to stitch together an animation with *ffmpeg*. I finished my project by improving my non-existant knowledge of *LaTex*. I will go through the derivation of the equations of motion for the system and then explain my code and how it works (code will be provided along the way.)

## 2. Derivation

The kinetic and potential energy of the system must be found first so that a Lagrangian can be defined and the equations of motion can be found. The first thing to do is to draw a picture,

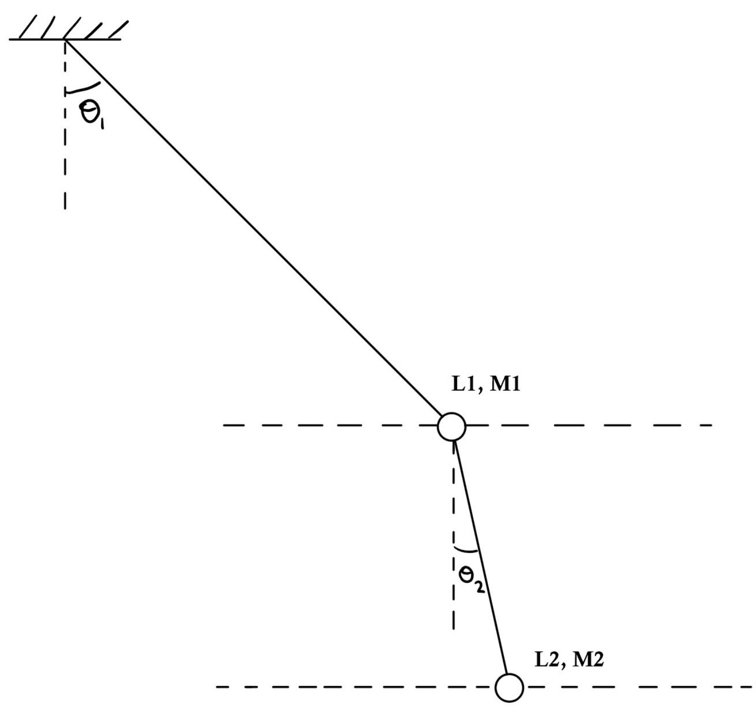


Figure 1: A double pendulum set-up.

It is now easy to define our coordinates of this system as,

$$x_1 = l_1 sin\theta_1 \tag{1}$$
$$y_1 = -l_1 cos\theta_1 \tag{2}$$
$$x_2 = l_1 sin\theta_1 + l_2 sin\theta_2 \tag{3}$$
$$y_2 = -l_1 cos\theta_1 - l_2 cos\theta_2 \tag{4}$$

We now must differentiate these coordinates to prepare for the kinetic energy component of the Lagrangian.

$$\dot{x}_1 = l_1\dot{\theta}_1 cos\theta_1 \tag{5}$$
$$\dot{y}_1 = l_1\dot{\theta}_1 sin\theta_1 \tag{6}$$
$$\dot{x}_2 = l_1\dot{\theta}_1 cos\theta_1 + l_2\dot{\theta}_2 cos\theta_2 \tag{7}$$
$$\dot{y}_2 = l_1\dot{\theta}_1 sin\theta_1 + l_2\dot{\theta}_2 sin\theta_2 \tag{8}$$

Now, it is easy to form the Lagrangian,

$$L = T - V \tag{9}$$
$$L = \frac{1}{2}m_1(\dot{x}_1^2 + y_1^2) + \frac{1}{2}m_2(\dot{x}_2^2 + y_2^2) - m_1 g y_1 - m_2 g y_2 \tag{10}$$

and therefore simplifying and putting in terms of one coordinate,

$$\begin{aligned} L = \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2[l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1l_2\dot{\theta}_1\dot{\theta}_2 cos(\theta_1 - \theta_2)] \\ + (m_1 + m_2)gl_1 cos\theta_1 + m_2 g l_2 cos\theta_2 \end{aligned} \tag{11}$$

Now the equations of motion are found using Euler-Lagrange's equations. $\theta_1$ and $\theta_2$ respectively,

$$\begin{aligned} \text{For } \theta_1\text{: } 0 = (m_1 + m_2)l_1\ddot{\theta}_1 + m_2 l_2\ddot{\theta}_2 cos(\theta_1 - \theta_2) + m_2 l_2\dot{\theta}_2^2 sin(\theta_1 - \theta_2) \\ + (m_1 + m_2)g sin\theta_1 \end{aligned} \tag{12}$$

$$\text{For } \theta_2\text{: } 0 = m_2 l_2\ddot{\theta}_2 + m_2 l_1\ddot{\theta}_1 cos(\theta_1 - \theta 2) - m_2 l_1\dot{\theta}_1^2 sin(\theta_1 - \theta_2) + m_2 g sin\theta_2 \tag{13}$$

These equations of motion are not analytically solvable (to my knowledge.) However, these equations are second order ordinary differential equations and can easily be converted into a system of first order differential equations with the trick that we learned from Mike Gussert.

## 3. Numerical Solution

If we use the trick that we learned for transforming second order ordinary differential equations into first order ones,

$$\ddot{\theta}_1 = \dot{z}_1 \text{ and } \ddot{\theta}_2 = \dot{z}_2 \tag{14}$$

Solving for $\dot{z}_1$ and $\dot{z}_2$,

$$\dot{z}_1 = \frac{m_2 g sin\theta_2 cos(\theta_1 - \theta_2) - m_2 sin(\theta_1 - \theta_2(l_1 z_1^2 cos(\theta_1 - \theta_2) + l_2 z_2^2) - (m_1 + m_2) g sin_1}{l_1(m_1 + m_2 sin^2(\theta_1 - \theta_2))} \tag{15}$$

$$\dot{z}_2 = \frac{(m_1 + m_2)(l_1 z_1^2 sin(\theta_1 - \theta_2) - g sin\theta_2 + g sin\theta_1 cos(\theta_1 - \theta_2)) + m_2 l_2 z_2^2 sin(\theta_1 - \theta_2) cos(\theta_1 - \theta_2)}{l_2(m_1 + m_2 sin^2(\theta_1 - \theta_2))} \tag{16}$$

We now have a system of first order ordinary differential equations that can be solved with one of *Python's* ordinary differential equation integral solvers, in this case, *integrate.odeint*.

Listing 1: Constants are defined and the differential equations of the system are solved.

```
import sys
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

L1 = 1
L2 = 1
m1 = 1
m2 = 1
g = 9.8

def ODE(y, t, L1, L2, m1, m2):
   y = theta1, z1, theta2, z2

    cos = np.cos(theta1-theta2)
    sin = np.sin(theta1-theta2)

    theta1dot = z1
    z1dot = (m2*g*np.sin(theta2)*cos - m2*sin*(L1*z1**2*cos + L2*z2**2) -
             (m1+m2)*g*np.sin(theta1)) / L1 / (m1 + m2*sin**2)
    theta2dot = z2
    z2dot = ((m1+m2)*(L1*z1**2*sin - g*np.sin(theta2) + g*np.sin(theta1)*cos) +
             m2*L2*z2**2*sin*cos) / L2 / (m1 + m2*sin**2)
    return theta1dot, z1dot, theta2dot, z2dot

tmax = 25
dt = 0.008
t = np.arange(0, tmax+dt, dt)
```

```

y0 = np.array([3*np.pi/7, 5, 3*np.pi/4, 0])
y = odeint(ODE, y0, t, args=(L1, L2, m1, m2))
theta1, theta2 = y[:,0], y[:,2]
```

This was the first differential equation I have solved numerically. Going through the lines of code, I started off by importing the libraries needed and defining the constants in the equation: lengths of strings, gravity, and masses of bobs. I couldn't get Mike Gussert's *Runge-Kutta 4* (RK4) example to work because I'm not sufficiently talented with scientific *Python*.

However, I instead defined a function for the specific set of differential equations that I found while studying the *Langrangian* of the system. Then, I defined a range of values corresponding to time-steps. I was unsure about how to define a *vector* of these numerical steps that I could input into my equation solver. I ended up utilizing a *numpy* function called *numpy.arange*. This built in function takes an interval and returns a evenly spaced values in between (a convenient way to save work and lines of code compared to using a for-loop.) I took the $t$ vector and plugged it back into my function for the motion. I also defined a $y_0$ array which corresponds to initial conditions for $\theta_{1_i}$, $\omega_{1_i}$, $\theta_{2_i}$, and $\omega_{2_i}$, respectively. The code took these initial values and evolved with the equations of motion. Finally, the last line of code in this uses slicing an array to define $\theta_1$ and $\theta_2$ as, respectively, the most left column and third column of my $y$ array. These $\theta$ arrays were used to plot the movies that I am attaching with this paper. The time-step array allowed the code to run 300 times and to take snapshots at every equally spaced point.

The equations of motion were solved with the *scipy.integrate.odeint* function. The *odeint* function takes the time and system of equations as arguments and solves them using *lsoda* a *FORTRAN* differential equation solving program. Because I could not understand how to implement my own RK4 methods, I needed to know the limitations of this method. Before I began solving my equations I analyzed them for singularities (regular or irregular). Luckily, there were no singular points that would make a *stiff* solver like *odeint* fail. learned that there is a new *scipy* function called *scipy.integrate.solve_ivp* that is recommended by the library's authors for future use. At first, I was discouraged that I could not write my own ODE solver, but after attending Wolfgang Bangerth's talk, I realized that using and building off of other's code is encouraged and necessary, as an amateur coder.

## 4. Analysis

My code created 300 *.jpeg* frames which I then needed to stitch together. I could have used an automatic program, like *Quicktime*, but this being a numerical project, I decided to use a command line tool, *ffmpeg*. *ffmpeg* is a versatile open-source tool that converts, stitches, and plays almost every type of multimedia format.

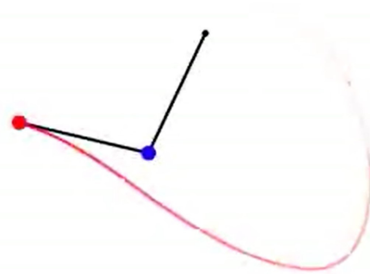

Figure 2: An example frame before being processed with *ffmpeg*.

I ran my program four times with different initial conditions and made a movie for each result. The four initial value problems had varying masses, lengths of strings, and initial velocities (the movies are labeled with their values.) The plotting that I used for each frame was borrowed from a common *scipy* method that appends a small motion blur tail on each bob. When these tails are merged together into an animation, it creates a fantastic sense of movement. I had some issues with energy conservation at the long timescales that I was observing the system. Because the solver uses numerical methods, there is an accumulation of error after each step. For some initial values, the system had a large energy drift where it would either stop completely or spiral out of control in a negative energy feedback loop. I avoided this problem by choosing small initial angular velocities and similar masses for each bob. I am unsure how to fix this issue completely. I would have to research further where the energy drift was coming from and how to eliminate it.

# 5. Conclusion

This task took countless hours of research into numerical methods, arrays, *Python's* differential equation solving methods, indexing, syntax errors, and logistical issues. I took out most of the *Python* books currently at the CSU library and I plan on continuing to read them over Christmas break. It helped me learn about different ways to look at computational problems and I hope to continue my research and to explore methods to solve partial differential equations. I believe the importance of visualisation of data with numerical methods (like making animations) will benefit my research at CSU and I hope to use it for scientific outreach in the future. This project enabled me to visualize one of our class and homework problems and helped cement a Lagrangian mechanics problem in my brain. This is also the first *LaTex* code that I have written. I learned how to make lists, organize sections, insert images, format code, and write equations. I think it will be useful for future paper submissions, grant proposals, and whenever I need to include equations with a document.

## 6. Acknowledgements

I'd like to thank you John (and the CSU faculty!) for encouraging my class to pursue learning numerical methods. I am absolutely certain that this will help me with my research and finding a job in the field when I graduate. I have always wanted to learn more but I needed a little push.