Double Pendulum
Project #4

Inst. Pavel Snopok, Spring 2015

Michael Lee

April 15, 2015

Abstract

Nonlinear dynamics has great importance in mathematics and is an essential tool in modern computational science. It allows us to understand phenomena such as fractals and chaotic motion and learn about systems beyond our normal comprehension. The double pendulum is a famous experiment in chaos theory and shows us the disparity between two outcomes given different initial conditions. We simulate the double pendulum and observe its behavior under different parameters to better understand nonlinear dynamics.

1 Introduction

Complex systems, nonlinear dynamics and chaos theory are all names to describe the indeterminism that occurs in physics. Predictability is the essence of a good theory but many physical systems exhibit processes that are chaotic and intractable to mathematical formalism. Perhaps the most famous problem in nonlinear dynamics is turbulence, described by Richard Feynman as "the most important problem in classical mechanics." (He worked on turbulence for many years hoping to resolve the problem but was unsuccessful in his attempts.)

An easier problem in classical mechanics is the double pendulum. It exemplified the chaotic nature of dynamical systems and it has a characteristic that many people colloquially refer to as the *the butterfly effect*, or small perturbations that cause significant changes to the outcome.

2 The Double Pendulum

The double pendulum is a device that uses one pendulum as the driving force of another connected to its weight. It produces chaotic motion.

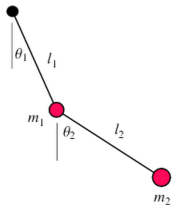

Figure 1: The double pendulum.

The equations of motion are derived directly from Lagrangian mechanics. If we have:

$$x_1 = l_1 \sin\theta_1,\ y_1 = -l_1 \cos\theta_1$$

$$x_2 = l_2 \sin\theta_2,\ y_2 = -l_2 \cos\theta_2$$

then we can express the potential energy as

$$PE = m_1 g y_1 + m_2 g y_2 =$$

$$-[(m_1 + m_2) g l_1 \cos\theta_1 + m_2 g l_2 \cos\theta_2]$$

and the kinetic energy as

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

$$\frac{1}{2} m_1 l_1^2 \dot{\theta_1^2} + \frac{1}{2} m_2 [l_1^2 \dot{\theta_1^2}$$

$$+ l_2^2 \dot{\theta_2^2} + 2 l_1 l_2 \dot{\theta_1} \dot{\theta_2} \cos(\theta_1 - \theta_2)]$$

We then express the Lagragian as

$$\mathcal{L} = KE - PE = \frac{1}{2}(m_1 + m_2) l_1^2 \dot{\theta_1^2} + \frac{1}{2} m_2 l_2^2 \dot{\theta_2^2} +$$

$$m_2 l_1 l_2 \dot{\theta_1} \dot{\theta_2} \cos(\theta_1 - \theta_2) + (m_1 + m_2) g l_1 cos\theta_1 + m_2 g l_2 cos\theta_2$$

Taking the derivative of $\mathcal{L}$ with respect to $\dot{\theta_1}$ and setting it equal to zero will give us the equation of motion for the m_1 pendulum, and likewise for $\dot{\theta_2}$ for m_2.

3 Results

For this report, we used the example double pendulum python file provided by matplotlib in their documentation. We made adjustments to the code to analyze the behavior of the pendulum under different conditions.

In Figure 2, when θ_1 and θ_2 were set to 0, we saw no motion. This is what we should expect and this means that the program is correctly functioning. A double pendulum with the initial conditions of $\theta_1 = 1$ and $\theta_2 = 1$ gave us a graph that looks like a normal pendulum but with small perturbations. This is exactly as it should be for small starting angles.
As we changed the angles to larger starting point, we saw that the motion became more chaotic as shown in Figure 4. As we added a small change to its initial condition, where θ_1 now equaled 119 instead of 120,

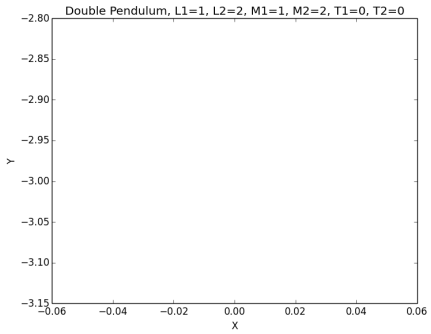

Figure 2: When θ_1 *and* θ_2 are both zero.

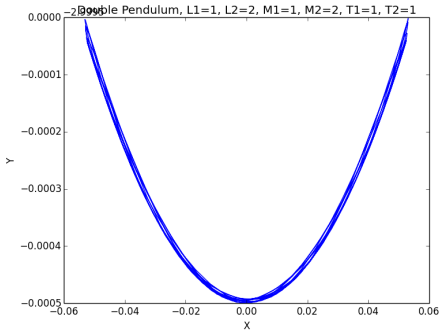

Figure 3: The double pendulum at a small angle.

we received a different plot of the weight's motion. A small change produced a large difference.

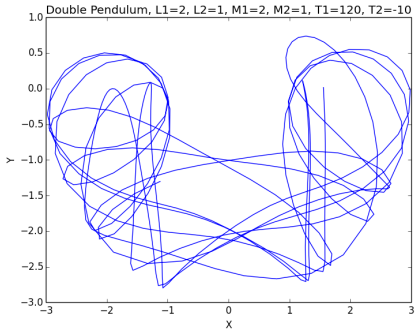

Figure 4: The double pendulum with the given parameters. $\theta_1 = 120$.

When the weight of the lower pendulum was increased, we saw it begin to produce an almost harmonic motion, driven by the top pendulum. This may be because we started it at a large angle, allowing itself to pull itself up through inertia and the motion could be almost described as *orbiting* the top pendulum.

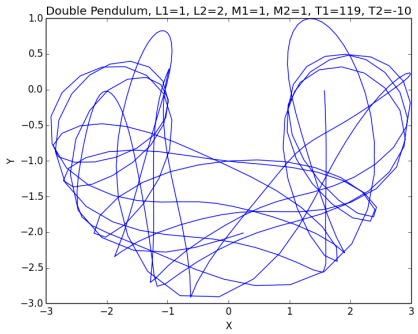

Figure 5: A small perturbation from figure 4. $\theta_1 = 119$.

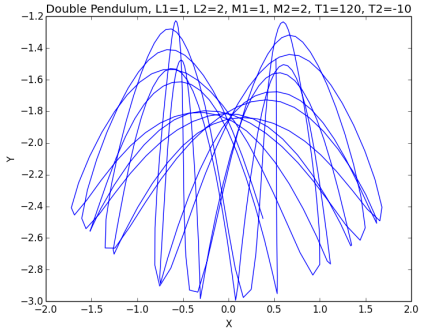

Figure 6: Exhibiting almost harmonic motion.

4 Summary

We were able to analyze the double pendulum for a variety of cases such as for the small angle and examined the case for the difference between two plots given a small change in the initial conditions. We saw that although we changed a parameter a very small amount, the difference we produce was great. We also saw that the motion of the top pendulum was governed more and more by the bottom pendulum the heavier we changed the bottom mass.

References

[1] R. Landau, Manual J. Paez, Cristian C. Bordeianu. *A Survey of Computational Physics*, 2010: Princeton University Press.

[2] D. Beazley and Brian K. Jones. *The Python Cookbook*, 2013: O'Reilly.

[3] Newman, M. E. J. *Computational Physics*, 2013: Createspace.

5 Appendix

5.1 Double Pendulum

```
"""
Written by Michael Lee.
Adapted from Double Pendulum simulation by matplotlib.

"Double pendulum formula translated from the C code at
http://www.physics.usyd.edu.au/~wheat/dpend_html/solve_dpend.c"

PHYS440: Computational physics
Project 4, Double Pendulum

"""

import matplotlib
matplotlib.use('TkAgg')
from numpy import sin, cos, pi, array
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import matplotlib.animation as animation

G = 9.8 # Acceleration due to gravity, in m/s^2

def double_pendulum(state, t, L1, L2, M1, M2):
    dydx = np.zeros_like(state)
    dydx[0] = state[1]
    del_ = state[2]-state[0]
    den1 = (M1+M2)*L1 - M2*L1*cos(del_)*cos(del_)
    dydx[1] = (M2*L1*state[1]*state[1]*sin(del_)*cos(del_)
               + M2*G*sin(state[2])*cos(del_) + M2*L2*state[3]*state[3]*sin(del_)
               - (M1+M2)*G*sin(state[0]))/den1
    dydx[2] = state[3]
    den2 = (L2/L1)*den1
    dydx[3] = (-M2*L2*state[3]*state[3]*sin(del_)*cos(del_)
               + (M1+M2)*G*sin(state[0])*cos(del_)
               - (M1+M2)*L1*state[1]*state[1]*sin(del_)
               - (M1+M2)*G*sin(state[2]))/den2
    return dydx

def parameters(theta1, theta2, ang_v1, ang_v2, L1, L2, M1, M2):
    return {'angle1': theta1, 'angle2': theta2, 'velocity1': ang_v1, \
    'velocity2': ang_v2, 'L1': L1, 'L2': L2, 'M1': M1, 'M2': M2}

def init():
    line.set_data([], [])
    time_text.set_text('')
    return line, time_text

def animate(i):
    thisx = [0, x1[i], x2[i]]
    thisy = [0, y1[i], y2[i]]
    line.set_data(thisx, thisy)
    time_text.set_text(time_template%(i*dt))
    return line, time_text

```

```
55 if __name__ == '__main__':
56
57   """
58   Parameters: angle of top pendulum, angle of bottom pendulum, angular velocity of
59   top pendulum, angular velocity of bottom pendulum, length of string on top pendulum,
60   length of string on bottom pendulum, mass of top pendulum, and mass of bottom
61   pendulum.
62
63   Change parameters for different situations. Use float or better precision for
64   better results.
65   """
66
67   initial_parameters = parameters(30., 20., 0., 0., 1, 2, 1, 3)
68   state = np.array([initial_parameters['angle1'], initial_parameters['velocity1'],
69           initial_parameters['angle2'], initial_parameters['velocity2']])*pi/180.
70
71   dt = 0.05
72   t = np.arange(0.0, 20, dt)
73
74   y = integrate.odeint(double_pendulum, state, t, (initial_parameters['L1'],
75         initial_parameters['L2'], initial_parameters['M1'], initial_parameters['M2']))
76
77   x1 = initial_parameters['L1']*sin(y[:,0])
78   y1 = -initial_parameters['L1']*cos(y[:,0])
79
80   x2 = initial_parameters['L2']*sin(y[:,2]) + x1
81   y2 = -initial_parameters['L2']*cos(y[:,2]) + y1
82
83   fig = plt.figure()
84   ax = fig.add_subplot(111, autoscale_on=False, xlim=(-4, 4), ylim=(-4, 1))
85   ax.grid()
86   plt.title('Double Pendulum; L1=%s, L2=%s, M1=%s, M2=%s,\n Ang Velocity Top=%s,\
87       Ang Velocity Bot=%s, Angle1=%s, Angle2=-%s' % (initial_parameters['L1'],\
88       initial_parameters['L2'], initial_parameters['M1'], initial_parameters['M2'],\
89       initial_parameters['velocity1'], initial_parameters['velocity2'], \
90       initial_parameters['angle1'], initial_parameters['angle2']))
91   plt.xlabel('X')
92   plt.ylabel('Y')
93   plt.plot(x2, y2)
94   line, = ax.plot([], [], 'o-', lw=2)
95
96   time_template = 'time = %.1fs'
97   time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)
98
99   ani = animation.FuncAnimation(fig, animate, np.arange(1, len(y)),
100      interval=25, blit=True, init_func=init)
101
102  plt.show()
```