Investigation of the motion of a double pendulum

Efe Yelesti[1]

[1] *Department of Physics, 21802069, Bilkent University, Bilkent, 06800, Ankara, Turkey*

(Dated: July 20, 2019)

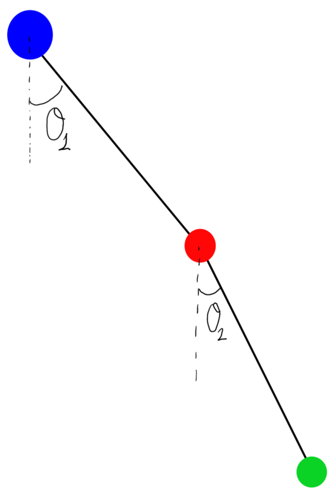
A computational method was used to obtain the trajectories of two masses of a double pendulum, which is a dynamical system with chaotic behavior. Different inital parameters, mostly different angles, were used. Program calculated 400 frames per simulation of double pendulum with given specific initial conditions. Chaotic motion had been expected to be observed with large inital angles. And on the contrary, linear alike motion had been expected to be observed with small initial angles. Chaotic motion was observed, but linear motion was not as significant as expected due to some numerical errors.

PACS numbers:

INTRODUCTION

Chaotic systems are of dynamical systems that are highly sensitive to initial conditions. That means a small change in initial parameters of the system will result a great difference in final status of the system. Since the final status of the system can differ from others within a small change of initial parameter, it is a common misconception that it is impossible to predict the behavior of the system. However some chaotic systems, like double pendulum, can be deterministic. This means that if its initial conditions are known, and can integrate it forward in time with infinite precision, we could predict its motion. For further information about chaotic motion and its applications in real life, referenced as [5] website can be visited. It shows papers on the applications of chaotic motion which can be helpful for further applications. Also Akerlof's experiment on double pendulum [7] gives rigorous explanations for those who want advanced explanations.

THEORY



In the diagram blue circle is fixed. Red is attached to blue and green is attached to red. Both red and green circles can move. Now call the position of red circle as (x_r, y_r), position of green circle as (x_g, y_g), distance between blue and red circle as l_1 and distance between red and green as l_2. Their position can also be written as;

$$x_r = l_1 sin(\theta_1) \tag{1}$$

$$y_r = -l_1 cos(\theta_1) \tag{2}$$

$$x_g = l_1 sin(\theta_1) + l_2 sin(\theta_2) \tag{3}$$

$$y_g = -l_1 cos(\theta_1) - l_2 cos(\theta_2) \tag{4}$$

To find the instantaneous velocities we take the derivatives of the equations 1-4 with respect to time.

$$\frac{dx_r}{dt} = l_1 cos(\theta_1)\frac{d\theta_1}{dt} \tag{5}$$

$$\frac{dy_r}{dt} = l_1 sin(\theta_1)\frac{d\theta_1}{dt} \tag{6}$$

$$\frac{dx_g}{dt} = l_1 cos(\theta_1)\frac{d\theta_1}{dt} + l_2 cos(\theta_2)\frac{d\theta_2}{dt} \tag{7}$$

$$\frac{dy_g}{dt} = l_1 sin(\theta_1)\frac{d\theta_1}{dt} + l_2 sin(\theta_2)\frac{d\theta_2}{dt} \tag{8}$$

To solve the Lagrange's Equation, it is necessary to find Lagrangian quantity which is the difference of kinetic and potential energies of the system.

$$\mathcal{L} = E_K - E_P = T - V \tag{9}$$

$$V = m_r g y_r + m_g g y_g$$

m_r = mass of the red circle, m_g=mass of the green circle

We can insert equation 2 and 4 to Potential equation:

$$V = -(m_r + m_g)gl_1cos(\theta_1) - m_ggl_2cos(\theta_2) \tag{10}$$

Kinetic energy(T):

$$T = \frac{m_rV_1^2 + m_gV_2^2}{2}$$

$$T = \frac{m_r((\frac{dx_r}{dt})^2 + (\frac{dy_r}{dt})^2) + m_g((\frac{dx_g}{dt})^2 + (\frac{dy_g}{dt})^2)}{2}$$

Substitute equations 5-8 into the kinetic energy equation,

$$T = \frac{m_r[(l_1cos(\theta_1)\frac{d\theta_1}{dt})^2 + (l_1sin(\theta_1)\frac{d\theta_1}{dt})^2]}{2} \tag{11}$$

$$+$$

$$\frac{m_g[(l_1cos(\theta_1)\frac{d\theta_1}{dt} + l_2cos(\theta_2)\frac{d\theta_2}{dt})^2]}{2} +$$

$$+$$

$$\frac{(l_1sin(\theta_1)\frac{d\theta_1}{dt} + l_2sin(\theta_2)\frac{d\theta_2}{dt})^2}{2}$$

Then we can substitute equation 10 and 11 into equation 9.

For the last step we can apply Lagrangian's Equations for θ_1 and θ_2 to find angular speeds and angular accelerations

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

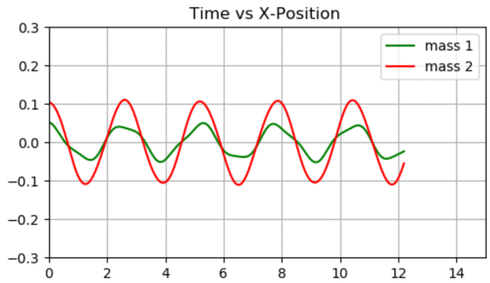
METHOD

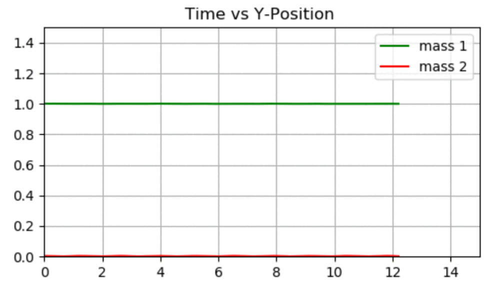
Python program will perform continuous calculations to find the positions of the masses. To find the the angular accelaration and angular speeds of the masses, lagrangian of the system will be solved. And the equations that will be obtained from the lagrangian equation will be used to find the angles for given times. Python's built in differentail equation solver will be used. User will give initial conditions, such as initial angles, masses of the objects and lengths of the rods. However masses of the rods will be neglected. Program then simulate the motion of the pendulum, plot the graphes of total energy of the system, x and y positions vs time.

THE COMPUTATIONAL RESULTS & INTERPRETATION

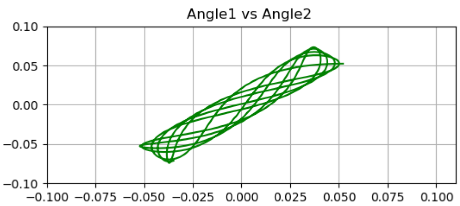
Initially, masses of the objects were set to 1 kg each and the lenths of the rods were set to 1 meters. The gravitational acceleration was set to $9.8m/s^2$. Then, for the first simulation, small angles were assigned as parameters. $\theta_1 = 3°$ and $\theta_2 = 3°$.

Double pendulum started to oscillate with small angles. Since the angle was small system behaved like a linear double spring.(Since double spring is out of the scope of this experiment, no detail was given. However double spring can be observed in the website with reference number [4]). The x and y positions of the masses were,





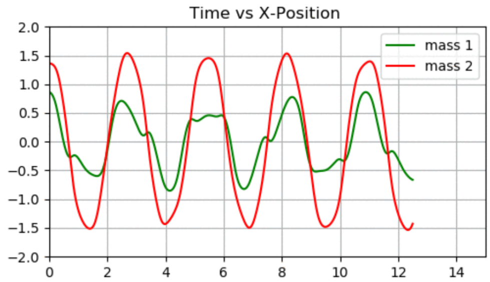
From the plot of time vs x-position, mass 2 oscilated like a simple pendulum. However, first mass oscilated strangely. The differential equation solver module could be the cause of the strange oscillations . Or it could be normal. When θ_1 vs θ_2 graph plotted,

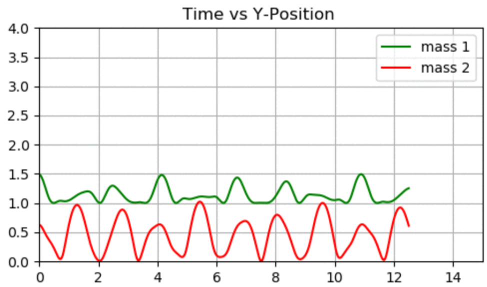


Both angles are in radians

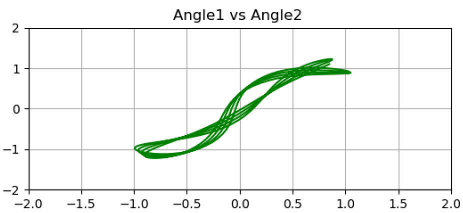
it traced out the familiar form of a Lissajous curve. This small motion is structered well and does not look like a chaotic system. This is due to the small angle approximation. When θ_1 and θ_2 are small enough their sin and cos functions become θ and 0 respectively. Also, since their momenta are small, the product of their momenta will be very small. Thus their momenta and product of momentas can be ignored. When all these considered, nonlinear systems of equations become linear. And motion of the pendulum becomes predictable.

When we enlarged the initial angles to $\theta_1 = 50°$ and $\theta_2 = 70°$, the motion of the masses were





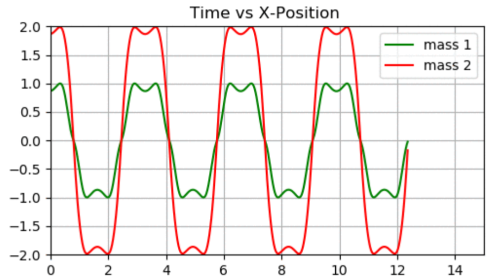
From the plot of time vs x-position, mass 2 oscilated like a simple pendulum. However, first mass oscilated strangely even more than the the oscilation of the small angle. In here, small angle approximation can still be used. However, the more the angles grow the more the error becomes. Also its Lissajous curve were plotted as,

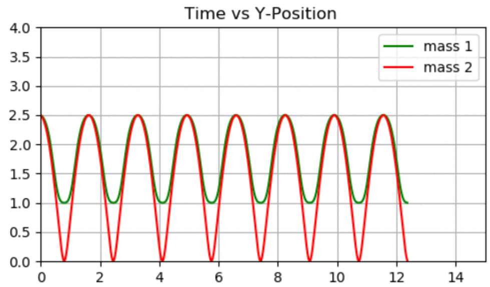


Both angles are in radians

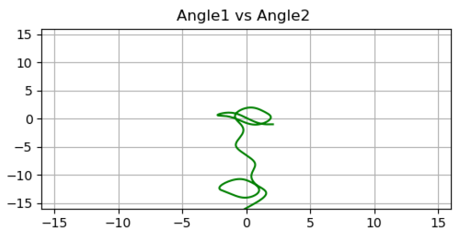
Its Lissajous curve is not as rectengular as the one with small inital angles. Its curve is stretched and bent. The curve still looks structered however it is a complex nonlinear shape.

When the initial angles were enlarged even moreto $\theta_1 = 120°$ and $\theta_2 = 90°$, the motion of the masses were become as,





Both masses oscilated strangely. This can be seen from the x positions of masses in different times. Also its Lissajous curve was plotted as

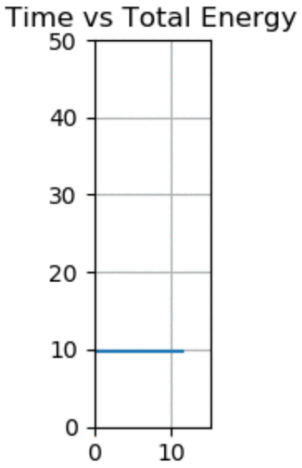


Both angles are in radians

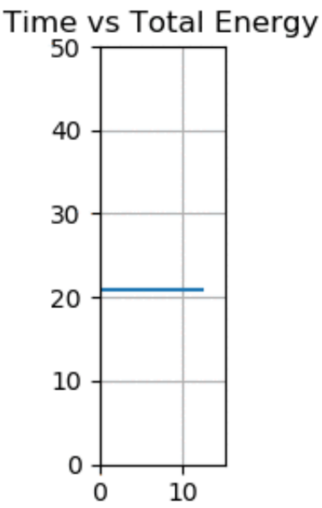
From its Lissajous curve, it can be seen that its structure is nonlinear shape.

ERRORS

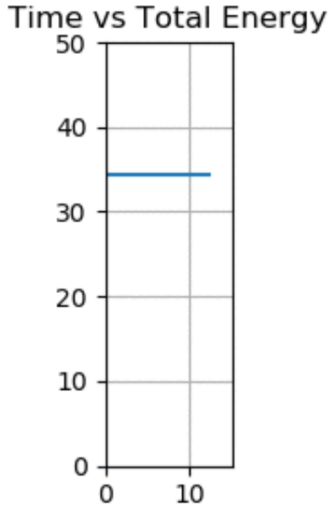
For the program, pythons scipi module imported and its built in differential solver was used. This module approximated the solution better than "euler-cromer" and "leap-frog" algorithms. Although advanced error calculations were not performed, energy vs time plots were plotted. If any fluctuation in the graph were spotted, it could have been understood that the calculations have errors.



Total Energy vs. Time of the pendulum with initial angles of $\theta_1 = 0.5°$, $\theta_2 = 0.5°$



Total Energy vs. Time of the pendulum with initial angles of $\theta_1 = 60°$, $\theta_2 = 30°$



Total Energy vs. Time of the pendulum with initial angles of $\theta_1 = 60°$, $\theta_2 = 120°$

However there weren't any fluctuations spotted in any of the simulations. So it can be concluded that error was small.

APPENDIX - THE COMPUTER CODE

```python
from numpy import sin,cos
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import *
import scipy.integrate as integrate
import matplotlib.animation as animation

class DoublePendulum:
    def __init__(self,
                 init_state = [0.0, 0.0, 0.0, 0.0],
                 L1 = 1.0,
                 L2 = 1.0,
                 M1 = 1.0,
                 M2 = 1.0,
                 G = 9.8,
                 origin =(0,2)):
        self.init_state = np.asarray(init_state, dtype='float')
        self.params=(L1, L2, M1, M2, G)
        self.origin = origin
        self.time_elapsed=0
        self.state = self.init_state*np.pi/180

    def position(self):
        (L1, L2, M1, M2, G) = self.params

        x = np.cumsum([self.origin[0],
                       L1 * sin(self.state[0]),
                       L2 * sin(self.state[2])])
        y = np.cumsum([self.origin[1],
                       -L1 * cos(self.state[0]),
                       -L2 * cos(self.state[2])])
        return (x,y)

    def energy (self):
        (L1, L2, M1, M2, G) = self.params
        x = np.cumsum([L1 * sin(self.state[0]),
                       L2 * sin(self.state[2])])
        y = np.cumsum([-L1 * cos(self.state[0]),
                       -L2 * cos(self.state[2])])

        vx = np.cumsum([L1 * self.state[1]*cos(self.state[0]),
                        L2 * self.state[3] * cos(self.state[2])])
        vy = np.cumsum([L1 * self.state[1] * sin(self.state[0]),
                        L2 * self.state[3] * sin(self.state[2])])

        U = G * (M1 * (2 + y[0]) + M2 * (2+y[1]))
        K = 0.5 * (M1 * np.dot(vx, vx) + M2 * np.dot(vy, vy))

        return U + K

    def dstate_dt(self, state, t):
        (L1, L2, M1, M2, G) = self.params

        dydx = np.zeros_like(state)
        dydx[0] = state[1]
        dydx[2] = state[3]

        cos_delta = cos(state[2]-state[0])
        sin_delta = sin(state[2]-state[0])

        den1 = (M1 + M2) * L1 - M2 * L1 * cos_delta * cos_delta
        dydx[1] = (M2 * L1 * state[1] * state[1] * sin_delta * cos_delta
                   + M2 * G * sin(state[2]) * cos_delta
                   + M2 * L2 * state[3] * state[3] * sin_delta
                   - (M1 + M2) * G * sin(state[0]))/den1

        den2 = (L2 / L1) * den1
        dydx[3] = (- M2 * L2 * state[3] * state[3] * sin_delta * cos_delta
                   + (M1 + M2) * G * sin(state[0]) * cos_delta
                   - (M1 + M2) * L1 * state[1] * state[1] * sin_delta
                   - (M1 + M2) * G * sin(state[2])) / den2

        return dydx

    def step(self, dt):
        self.state = integrate.odeint(self.dstate_dt, self.state, [0, dt])[1]
        self.time_elapsed += dt


pendulum = DoublePendulum([90.0, 0.0, 90.0, 0.0])
dt = 1./30


# Setup figure and subplots
fig = figure(num = 0, figsize = (12, 8))#, dpi = 100)
fig.suptitle("Double Pendulum", fontsize=12)
ax01 = subplot2grid((4, 4), (0, 0), colspan=2, rowspan=2, autoscale_on=False, aspect = 'equal', xlim = (-2,2), ylim = (0,4))
ax01.grid()
ax01.set_title("Simulation")
line, = ax01.plot([],[],'o-', lw = 2)
time_text = ax01.text(0.02, 0.95, '', transform=ax01.transAxes)
energy_text = ax01.text(0.02, 0.90, '', transform = ax01.transAxes)


# Data Placeholders
en1=zeros(0)
t=zeros(0)
xpos1 = zeros(0)
xpos2 = zeros(0)
ypos1 = zeros(0)
ypos2 = zeros(0)


x_max = 5
ax02 = subplot2grid((5, 4), (0, 2), colspan = 2, rowspan = 2, aspect = 'equal', xlim = (0,15), ylim = (0,50))
ax02.grid()
ax02.set_title("Time vs Total Energy")
energygraph, = ax02.plot(t,en1)
```

```
ax03 = subplot2grid((5, 4), (3, 0), colspan = 2, rowspan = 2, xlim = (0,15), ylim = (-2,2))
ax03.grid()
ax03.set_title("Time vs X-Position ")
xpos1graph, = ax03.plot(t,xpos1, "g-", label="mass 1")
xpos2graph, = ax03.plot(t,xpos2, "r-", label="mass 2")

ax04 = subplot2grid((5, 4), (3, 2), colspan = 2, rowspan = 2, xlim = (0,15), ylim = (0,4))
ax04.grid()
ax04.set_title("Time vs Y-Position ")
ypos1graph, = ax04.plot(t,ypos1, "g-", label="mass 1")
ypos2graph, = ax04.plot(t,ypos2, "r-", label="mass 2")

# Setting Lagends
ax03.legend([xpos1graph,xpos2graph], [xpos1graph.get_label(),xpos2graph.get_label()])
ax04.legend([ypos1graph,ypos2graph], [ypos1graph.get_label(),ypos2graph.get_label()])

# Data Update

time1 = 0.0


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


    return line, time_text, energy_text


def animate(i):
    global pendulum, dt, time1, en1, t, x_max, xpos1, xpos2, ypos1, ypos2
    pendulum.step(dt)

    line.set_data(*pendulum.position())
    time_text.set_text('time = %.1f' % pendulum.time_elapsed)
    energy_text.set_text('energy = %.3f' % pendulum.energy())

    en1 = append(en1, pendulum.energy())
    t = append(t, time1)
    time1 += dt

    energygraph.set_data(array(t), array(en1))

    xpos1 = append(xpos1, pendulum.position()[0][1])
    xpos2 = append(xpos2, pendulum.position()[0][2])
    xpos1graph.set_data(t, xpos1)
    xpos2graph.set_data(t, xpos2)

    ypos1 = append(ypos1, pendulum.position()[1][1])
    ypos2 = append(ypos2, pendulum.position()[1][2])
    ypos1graph.set_data(t, ypos1)
    ypos2graph.set_data(t, ypos2)

    #if time1 >= x_max - 1.00:
        #energygraph.axes.set_xlim(time1 - x_max + 1.0, time1 + 1.0)
        #xpos1graph.axes.set_xlim(time1 - x_max + 1.0, time1 + 1.0)

    return line, time_text, energy_text, energygraph, xpos1graph, xpos2graph, ypos1graph, ypos2graph


from time import time


t0 = time()
animate(0)
t1 = time()
interval = 1000 * dt - (t1 - t0)

ani = animation.FuncAnimation(fig, animate, frames = 400,
                              interval = interval, blit = False,
                              init_func=init)


#ani.save("p90_p90.gif", writer="imagemagick", fps=30)
plt.show()
```


[1] Gould, H., Tobochnik, J., & Christian, W. (2007). *An introduction to computer simulation methods: Applications to physical systems.* San Francisco: Pearson Addison Wesley.

[2] Train, The Coding. *"Coding Challenge #93: Double Pendulum." YouTube,* YouTube, 13 Feb. 2018,

[3] Morin, D. J. (2015). *Introduction to classical mechanics: With problems and solutions.* Cambridge: Cambridge University Press.

[4] *Double Spring* https://www.myphysicslab.com/springs/double-spring-en.html

[5] *Chaotic Motion* https://www.sciencedirect.com/topics/engineering motion

[6] *Matplotlib animation example* matplotlib.sourcefourge.net/examples/animation/double_pendulu (Helped me with animating my pendulum simply without using vpython)

[7] Akerlof, C. W. (2012) *The Chaotic Motion of a Double Pendulum*

[8] VanderPlas, J. *https://github.com/jakevdp*