UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ciencias
Física computacional

Profesor: Erick Javier López Sánchez

Equipo:
Sánchez García Fabricio Alejandro

Doble Péndulo Acoplado con resortes. Solución con RK4

Resumen

En este trabajo, se resuelven las ecuaciones de Lagrange para un doble péndulo 2-Dimensional acoplado con resortes, con extensiones de longitud dependientes del tiempo, fueron obtenidas y resueltas de manera aproximada. Las ecuaciones resultantes se resolvieron utilizando el método de Runge-Kutta a orden 4 a través de Python. Se logró observar que los cambios en las constantes de Hook, en las masas así como en las longitudes de los péndulos tienen efectos significativos en la dinámica del sistema. El comportamiento periódico y caótico apreciado en este trabajo es consistente con la literatura actual sobre sistemas de resorte-péndulo. Finalmente, se logró simular el movimiento de los péndulos imponiendo condiciones iniciales al sistema.

Introducción

El doble péndulo bidimensional (2D) es un ejemplo típico de movimiento caótico en la mecánica clásica. Es bien conocido que el patrón de su movimiento cambia drásticamente a medida que la energía aumenta de cero a infinito . Sin embargo, a energías bajas y muy altas el sistema representa osciladores armónicos acoplados y puede considerarse como un sistema integrable. Pero, en energías intermedias, se sabe que el sistema presenta características caóticas. El péndulo de doble resorte es un sistema mecánico clásico que consta de dos cuerpos de masa m1 y m2 fijadas a los extremos de dos resortes elásticos ingrávidos con constantes de Hook k_1 y k_2 y ángulos θ_1 y θ_2. La Figura 1 muestra el sistema descrito anteriormente.

Además, es bien sabido que los sistemas físicos pueden describirse mediante su lagrangiano, y a partir de éste se pueden obtener ecuaciones diferenciales de movimiento de segundo orden que describen dichos sistemas dinámicos. En la mayoría de los casos, no se puede obtener la solución exacta para estas ecuaciones de movimiento de Lagrange, y esto lleva a emplear un enfoque numérico alternativo para resolver dichas ecuaciones. El interés de este trabajo es obtener analíticamente las ecuaciones de movimiento de un péndulo de resorte doble con extensión de resorte dependiente del tiempo. Por tanto, se utiliza la formulación lagrangiana de la mecánica para obtener las ecuaciones de movimiento del sistema. Las ecuaciones de Lagrange resultantes se resuelven de forma aproximada y numérica utilizando Python.

Obtención de las ecuaciones de movimiento de Lagrange

La figura 1 muestra el sistema dinámico que queremos estudiar:

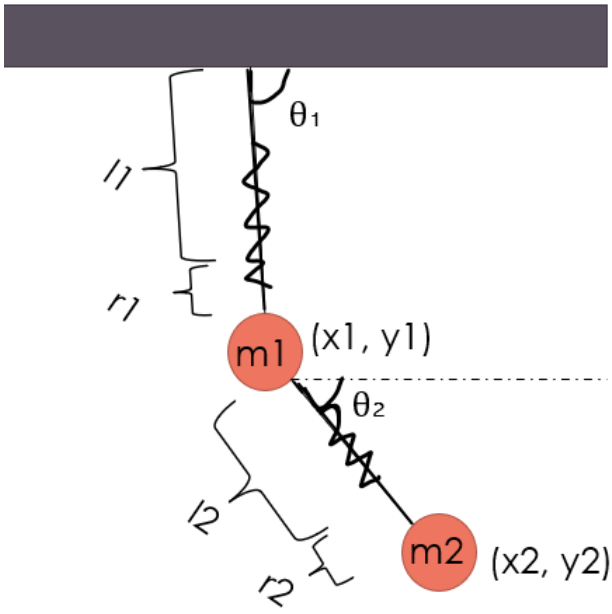

Figura 1. Diagrama del sistema.

Suponemos que las posiciones de las masas m_1 y m_2 en cualquier tiempo en el espacio son expresadas en coordenadas cartesianas como (x_1, y_1) y (x_2, y_2) respectivamente. Las longitudes naturales de los resortes son l_1 y l_2 y las longitudes de extensión son $r_1(t)$ y $r_2(t)$ respectivamente como se muestra en la figura 1.

Las coordenadas de posición de los péndulos son las siguientes:

$$x_1(t) = r_1(t)sen\theta_1(t) \tag{1}$$

$$y_1(t) = -r_1(t)cos\theta_1(t) \tag{2}$$

$$x_2(t) = r_1(t)sen\theta_1(t) + r_2(t)sen\theta_2(t) \tag{3}$$

$$y_2(t) = -r_1(t)cos\theta_1(t) - r_2(t)cos\theta_2(t) \tag{4}$$

Con estas coordenadas disponibles podemos escribir el Lagrangiano correspondiente a este sistema:

$$L = T - V \tag{5}$$

La energía cinética del sistema es entonces:

$$T = \frac{1}{2}m(\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2}m(\dot{x}_2^2 + \dot{y}_2^2) \tag{6}$$

Mientras que la energía potencial es:

$$V = \frac{1}{2}k(r_1(t) - l_1)^2 + mgy_1 + \frac{1}{2}k(r_2(t) - l_2)^2 + mgy_2 \tag{7}$$

Por lo tanto, escribiendo el Lagrangiano y desarrollando en términos de las coordendas generalizadas conseguimos la siguiente expresión:

$$\begin{aligned} L = {} & \frac{1}{2}m\left[(\dot{r}_1(t)sen\theta_1(t) + r_1(t)cos\theta_1(t)\dot{\theta}_1(t))^2 + (-\dot{r}_1(t)cos\theta_1(t) - r_1(t)sen\theta_1(t)\dot{\theta}_1(t))^2\right] \\ & + \frac{1}{2}m\left[(\dot{r}_1(t)sen\theta_1(t) + r_1(t)cos\theta_1(t)\dot{\theta}_1(t) + \dot{r}_2(t)sen\theta_2(t) + r_2(t)cos\theta_2(t)\dot{\theta}_2(t))^2 + (\dot{r}_1(t)cos\theta_1(t) + r_1(t)sen\theta_1(t)\dot{\theta}_1(t)\right. \\ & \left. - \dot{r}_2(t)cos\theta_2(t) + r_2(t)sen\theta_2(t)\dot{\theta}_2(t))^2\right] \\ & - \left[\frac{1}{2}k_1(l_1^2 - 2l_1r_1(t) + r_1^2(t)) + m_1g(-r_1(t)cos\theta_1(t) + \frac{1}{2}k_2(l_2^2 - 2l_2r_2(t) + r_2^2(t)) + m_2g(-r_1(t)cos\theta_1(t) - r_2(t)cos\theta_2(t))\right] \end{aligned} \tag{8}$$

Como se puede apreciar, el Lagrangiano por sí solo luce bastante complicado. Para resolverlo, nos encargamos primero de establecer las Ecuaciones de Euler-Lagrange para las θ y las r:

$$\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} = 0 \tag{9}$$

En total se tendrán cuatro ecuaciones de Lagrange, correspondientes a θ_1, θ_2, r_1, r_2. Por supuesto que éstas lucirán todavía más complicadas, por ejemplo mostramos la primera ecuación de Lagrange para θ_1:

$$m\left(2.0g\,\mathrm{r}_1(t)\cos(\theta_1(t)) + 2.0g\cos(\theta_1(t)) - 2.0\,\mathrm{r}_1{}^2(t)\frac{d^2}{dt^2}\theta_1(t) - 1.0\,\mathrm{r}_1(t)\,\mathrm{r}_2(t)\sin(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.0\,\mathrm{r}_1(t)\,\mathrm{r}_2(t)\cos\right.$$
$$(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 1.0\,\mathrm{r}_1(t)\sin(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\,\mathrm{r}_1(t)\sin(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\mathrm{r}_2(t) - 2.0\,\mathrm{r}_1(t)\cos$$
$$(\theta_1(t)-\theta_2(t))\frac{d}{dt}\theta_2(t)\frac{d}{dt}\mathrm{r}_2(t) - 1.0\,\mathrm{r}_1(t)\cos(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 4.0\,\mathrm{r}_1(t)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\mathrm{r}_1(t) - 4.0\,\mathrm{r}_1(t)\frac{d^2}{dt^2}\theta_1(t) - 1.0\,\mathrm{r}_2(t)\sin$$
$$(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.0\,\mathrm{r}_2(t)\cos(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 1.0\sin(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\sin(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\mathrm{r}_2(t) - 2.0\cos$$
$$\left.(\theta_1(t)-\theta_2(t))\frac{d}{dt}\theta_2(t)\frac{d}{dt}\mathrm{r}_2(t) - 1.0\cos(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 4.0\frac{d}{dt}\theta_1(t)\frac{d}{dt}\mathrm{r}_1(t) - 2.0\frac{d^2}{dt^2}\theta_1(t)\right)$$

Figura 2. Ecuación de lagrange para θ_1.

.
Si resolvemos para $\frac{d^2q}{dt^2}$ entonces obtenemos dos ecuaciones para cada q. Definiendo Vq como $\frac{dq}{dt}$ se tiene lo siguiente:

$$\frac{dq}{dt} = V_q$$

$$\frac{dV_q}{dt} = Arbitrario$$

Definiendo las siguientes derivadas:

$$\omega_1 \equiv \frac{d\theta_1}{dt} \qquad \omega_2 \equiv \frac{d\theta_2}{dt} \tag{10}$$
$$v_1 \equiv \frac{r_1}{dt} \qquad v_2 \equiv \frac{r_2}{dt} \tag{11}$$

Se resuelven las ecuaciones de Lagrange para las segundas derivadas en cada coordenada. Mostramos cómo se vería una de las soluciones escritas:

$$m\left(2.0g\,\mathrm{r}_1(t)\cos(\theta_1(t)) + 2.0g\cos(\theta_1(t)) - 2.0\,\mathrm{r}_1{}^2(t)\frac{d^2}{dt^2}\theta_1(t) - 1.0\,\mathrm{r}_1(t)\,\mathrm{r}_2(t)\sin(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.0\,\mathrm{r}_1(t)\,\mathrm{r}_2(t)\cos\right.$$
$$(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 1.0\,\mathrm{r}_1(t)\sin(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\,\mathrm{r}_1(t)\sin(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\mathrm{r}_2(t) - 2.0\,\mathrm{r}_1(t)\cos$$
$$(\theta_1(t)-\theta_2(t))\frac{d}{dt}\theta_2(t)\frac{d}{dt}\mathrm{r}_2(t) - 1.0\,\mathrm{r}_1(t)\cos(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 4.0\,\mathrm{r}_1(t)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\mathrm{r}_1(t) - 4.0\,\mathrm{r}_1(t)\frac{d^2}{dt^2}\theta_1(t) - 1.0\,\mathrm{r}_2(t)\sin$$
$$(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.0\,\mathrm{r}_2(t)\cos(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 1.0\sin(\theta_1(t)-\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\sin(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\mathrm{r}_2(t) - 2.0\cos$$
$$\left.(\theta_1(t)-\theta_2(t))\frac{d}{dt}\theta_2(t)\frac{d}{dt}\mathrm{r}_2(t) - 1.0\cos(\theta_1(t)-\theta_2(t))\frac{d^2}{dt^2}\theta_2(t) - 4.0\frac{d}{dt}\theta_1(t)\frac{d}{dt}\mathrm{r}_1(t) - 2.0\frac{d^2}{dt^2}\theta_1(t)\right)$$

Figura 3. Solución para $\frac{d\omega_1}{dt}$

De esta forma se tienen cuatro ecuaciones diferenciales ordinarias de segundo orden acopladas y con las definiciones proporcionadas previamente se tienen 8 ecuaciones diferenciales ordinarias de primer orden acopladas. Las primeras cuatro son:

$$\dot{\theta}_1(t) = \omega_1 \qquad \dot{r}_1(t) = v_1 \tag{12}$$

$$\dot{\theta}_2(t) = \omega_2 \qquad \dot{r}_2(t) = v_2 \tag{13}$$

Apoyándonos de la librería Sympy de Python la cual realiza cálculo simbólico, como el mostrado en la Figura 3, logramos escribir las otras cuatro ecuaciones diferenciales:

$$\dot{\omega_1} = \frac{k_2(l_2 - r_2(t))sen(\theta_1(t) - \theta_2(t)) - m_1 g sen\theta_1(t) - 2m_1 v_1 \omega_1}{m_1 r_1(t)} \tag{14}$$

$$\dot{v_1} = \frac{k_1(l_1 - r_1(t)) - k_2(l_2 - r_2(t))cos(\theta_1(t) - \theta_2(t)) + m_1 g cos\theta_1(t) + m_1 r_1(t)\omega_1}{m_1} \tag{15}$$

$$\dot{\omega_2} = \frac{-2m_1 v_2 \omega_2 - k_1(l_1 - r_1(t))sen(\theta_1 - \theta_2)}{m_1 r_2(t)} \tag{16}$$

$$\dot{v_2} = \frac{k_2(m_1 + m_2)(l_2 - r_2(t)) + m_1 m_1 r_2(t)\omega_2^2 - m_2 k_1(l_1 - r_1(t)cos(\theta_1 - \theta_2))}{m_1 m_2} \tag{17}$$

Solución numérica

Con las ecuaciones diferenciales bien establecidas, procedemos a escribir nuestro código, el cuál fue elaborado en Python.

```
import matplotlib.animation as animation
from matplotlib.animation import PillowWriter
import timeit
import os
import matplotlib.pyplot as plt
from numpy import sqrt, sin, cos, arange, pi, append, array, floor
from pylab import plot, xlabel, ylabel, title, show, axhline, savefig, subplots_adjust,\
        figure, xlim, rcParams, rc, rc_context, subplot, tight_layout, axvline, xlim, ylim, scatter
```

```
#Aquí definimos el tiempo de inicio
inicio = timeit.default_timer()
```

```
#Definimos una función para calcular las deriva en una posición determinada
# eta = (theta1, omega1, r1, v1, theta2, omega2, r2, v2)
def f(eta):
    theta1 = eta[0]
    omega1 = eta[1]
    r1 = eta[2]
    v1 = eta[3]
    theta2 = eta[4]
    omega2 = eta[5]
    r2 = eta[6]
    v2 = eta[7]
    f_theta1 = omega1
    f_r1 = v1
    f_theta2 = omega2
    f_r2 = v2

    f_omega1 = (k2*(l2-r2)*sin(theta1-theta2) - 2*m1*v1*omega1\
                - m1*g*sin(theta1)) / (m1*r1)

    f_v1 = (m1*g*cos(theta1) + k1*(l1-r1) - k2*(l2-r2)*cos(theta1-theta2)\
            + m1*r1*omega1**2) / m1

    f_omega2 = (-k1*(l1-r1)*sin(theta1-theta2) - 2*m1*v2*omega2) / (m1*r2)

    f_v2 = (k2*(m1+m2)*(l2-r2) + m1*m2*r2*omega2**2 -m2*k1*(l1-r1)*cos(theta1-theta2))/(m1*m2)

    return(array([f_theta1,f_omega1, f_r1, f_v1, \
                  f_theta2,f_omega2, f_r2, f_v2],float))
```

Figura 4. Establecemos las ecuaciones diferenciales que obtuvimos a partir de las ecuaciones de Euler-Lagrange

A continuación, definimos una función que permite establecer los valores de los ángulos iniciales y las compresiones de los resortes como parámetros en un arreglo de salidas de ángulos y velocidades de las dos masas involucradas. Así mismo, se establece el intervalo del tiempo en el cuál se estará trabajando y los saltos de tiempo considerados.

```
# definimos una función que toma ángulos iniciales y compresiones de resorte como
# parámetros y arreglo de salidas de ángulos y velocidades de ambas masas
def doblependulo(theta1_initial_deg, theta2_initial_deg, r1_initial, r2_initial):
    omega1_initial_deg = 0.0                    # Velocidad angular incial
    omega2_initial_deg = 0.0                    # Velocidad angular inicial
    theta1_initial = theta1_initial_deg*pi/180  # convierte el ángulo inicial a grados
    omega1_initial = omega1_initial_deg*pi/180  # convierte la velocidad angular inicial en grados
    theta2_initial = theta2_initial_deg*pi/180  # convierte el ángulo inicial a grados
    omega2_initial = omega2_initial_deg*pi/180  # convierte la velocidad angular inicial en gradoss
    v1_initial = 0.0                            # Valor incial de v_1 (r1 punto)
    v2_initial = 0.0                            # Valor inciial de v_2 (r2 punto)

    # Establecemos el intervalo que ocuparemos
    a = 0.0                                 # Inicio
    b = 80.0                                # Final
    dt = 0.003                              # Tiempo de salto
    t_points = arange(a,b,dt)               # matriz de tiempos


    # Condiciones iniciales eta = (theta1, omega1, r1, v1, theta2, omega2, r2, v2)
    eta = array([theta1_initial, omega1_initial, r1_initial, v1_initial, \
                 theta2_initial, omega2_initial, r2_initial, v2_initial],float)
```

Figura 5. Valores iniciales de las velocidades y compresiones de los resortes. Se establecen el intervalo de tiempo y las divisiones del mismo.

A continuación, se crean listas vacías para ir actualizandolos mediante los valores que son de interés, para finalmente aplicar Runge-Kutta de orden 4.

```
    # Aquí creamos unos conjuntos vacíos para actualizar con los valores que nos interesan, luego aplicamos Runge-Kutta orden 4
    theta1_points = []
    omega1_points = []
    r1_points = []
    v1_points = []
    theta2_points = []
    omega2_points = []
    r2_points = []
    v2_points = []
    energy_points = []
    for t in t_points:

        theta1_points.append(eta[0])
        omega1_points.append(eta[1])
        r1_points.append(eta[2])
        v1_points.append(eta[3])
        theta2_points.append(eta[4])
        omega2_points.append(eta[5])
        r2_points.append(eta[6])
        v2_points.append(eta[7])
        E_1 = 0.5*m1*(eta[3]**2 + eta[2]**2 * eta[1]**2) + 0.5*k1*(l1-eta[2])**2 \
                  - m1*g*eta[2]*cos(eta[0])
        E_2 = 0.5*m2*(eta[3]**2 + eta[7]**2) + m2*(-sin(eta[0]-eta[4])*eta[2]*eta[7]*eta[1] \
                      + eta[2]**2 * eta[1]**2 + cos(eta[0]-eta[4])*eta[2]*eta[6]*eta[1]*eta[5] \
                      + 0.5*eta[6]**2 * eta[5]**2 + eta[3]*(cos(eta[0]-eta[4])*eta[7] \
                      + sin(eta[0]-eta[4])*eta[6]*eta[5])) + 0.5*k2*(l2-eta[6])**2 \
                      - m2*g*(cos(eta[0])*eta[2] + cos(eta[4])*eta[6])
        E_net = E_1 + E_2
        energy_points.append(E_net)

        kutta1 = dt*f(eta)
        kutta2 = dt*f(eta + 0.5*kutta1)
        kutta3 = dt*f(eta + 0.5*kutta2)
        kutta4 = dt*f(eta + kutta3)
        eta += (kutta1 + 2*kutta2 + 2*kutta3 + kutta4)/6

    return(theta1_points, omega1_points, r1_points, v1_points, \
              theta2_points, omega2_points, r2_points, v2_points, t_points, energy_points)
```

Figura 6. Aplicación de Runge-Kutta a orden 4.

El usuario puede ingresar las condiciones de frontera, es decir los valores de las masas, las longitudes de los resortes y las constantes de Hook respectivamente.

Posteriormente se llama a la función previamente definida y se almacena la matriz de datos. Luego se crean listas de valores tanto para x como para y y para la masa 1 y la masa 2.

```
# llama a la función y almacena las matrices de datos
theta1_points, omega1_points, r1_points, v1_points, theta2_points, omega2_points,\
        r2_points, v2_points, t_points, energy_points = doblependulo(170,105, l1*1.65, l2*1.95)
```

```
# creamos listas de valores x e y para la masa 1 y la masa 2
x1_points = []
y1_points = []
x2_points = []
y2_points = []
for i in range(len(t_points)):
    x1_new = r1_points[i]*sin(theta1_points[i])
    y1_new = -r1_points[i]*cos(theta1_points[i])
    x1_points.append(x1_new)
    y1_points.append(y1_new)

    x2_new = r1_points[i]*sin(theta1_points[i]) + r2_points[i]*sin(theta2_points[i])
    y2_new = -r1_points[i]*cos(theta1_points[i]) - r2_points[i]*cos(theta2_points[i])
    x2_points.append(x2_new)
    y2_points.append(y2_new)
```

Finalmente, se procede a hacer la animación correspondiente y se grafica la energía del sistema contra el tiempo.

```
# Animaciones en camino... =D
# Opciones de estilo ¡estilooo ;D!
rcParams.update({'font.size': 18})
rc('axes', linewidth=2)
with rc_context({'axes.edgecolor':'white', 'xtick.color':'white', \
                 'ytick.color':'white', 'figure.facecolor':'darkslategrey',\
                 'axes.facecolor':'darkslategrey','axes.labelcolor':'white',\
                 'axes.titlecolor':'white'}):
    # Gráfica de energía vs tiempo
    fig_E = figure(figsize=(10,7))
    ax_energy = subplot(1,1,1)
    title("Energía vs. Tiempo")
    xlabel("Tiempo (s)")
    ylabel("Energía (J)")
    ax_energy.plot(t_points,energy_points,color='coral',lw=1.4)
    output_directory = "./elasticDoublePendulumm/"
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)
    savefig(os.path.join(output_directory, "175_185_elasticDoublePend_energy_kutta003.png"))


    # Tiempo de ejecución del código
    stop = timeit.default_timer()
    print('Time: ', stop - inicio)

    # Configuramos la parte del péndulo
    fig = figure(figsize=(10,10))

    # Animación del péndulo
    ax_pend = subplot(1,1,1, aspect='equal')
    # get rid of axis ticks
    ax_pend.tick_params(axis='both', colors="darkslategrey")
```

```
### Aquí ya tenemos la animación como tal ###
# creamos una lista para ingresar imágenes para cada paso de tiempo
ims = []
index = 0
# Solo muestra los primeros 80 segundos aproximadamente en el gif.
while index <= len(t_points)-1:
    ln1, = ax_pend.plot([0,r1_points[index]*sin(theta1_points[index])],\
                                [0,-r1_points[index]*cos(theta1_points[index])],\
                        color='k',lw=3,zorder=99)
    bob1, = ax_pend.plot(r1_points[index]*sin(theta1_points[index]),\
                                -r1_points[index]*cos(theta1_points[index]),'o',\
                                markersize=22,color="m",zorder=100)

    ln2, = ax_pend.plot([r1_points[index]*sin(theta1_points[index]),\
                        r1_points[index]*sin(theta1_points[index])+\
                        r2_points[index]*sin(theta2_points[index])],\
                        [-r1_points[index]*cos(theta1_points[index]),\
                            -r1_points[index]*cos(theta1_points[index])\
                            -r2_points[index]*cos(theta2_points[index])], color='k',lw=3,zorder=99)
    bob2, = ax_pend.plot(r1_points[index]*sin(theta1_points[index])+\
                        r2_points[index]*sin(theta2_points[index]),\
                        -r1_points[index]*cos(theta1_points[index])\
                        -r2_points[index]*cos(theta2_points[index]),'o',\
                                markersize=22,color="coral",zorder=100)

    if index > 200:
        trail1, = ax_pend.plot(x1_points[index-140:index],y1_points[index-140:index], \
                            color="lime",lw=0.8,zorder=20)

        trail2, = ax_pend.plot(x2_points[index-190:index],y2_points[index-190:index], \
                            color="cyan",lw=0.8,zorder=20)

    else:
        trail1, = ax_pend.plot(x1_points[:index],y1_points[:index], \
                            color="lime",lw=0.8,zorder=20)

        trail2, = ax_pend.plot(x2_points[:index],y2_points[:index], \
                               color="cyan",lw=0.8,zorder=20)
    # Agrega frames a la lista ims
    ims.append([ln1, bob1, ln2, bob2, trail1, trail2])

    # Sólo se muestra cada 6 frames
    index += 6
# Guardamos la animación y listo

ani = animation.ArtistAnimation(fig, ims, interval=100)
writervideo = PillowWriter(fps=60)
ani.save('./elasticDoublePendulumm/170_105_elasticDoublePend.gif', writer=writervideo)
```

A continuación se muestra la gráfica de la energía del sistema vs el tiempo. Esta gráfica está sujeta a las condiciones proporcionadas anteriormente.

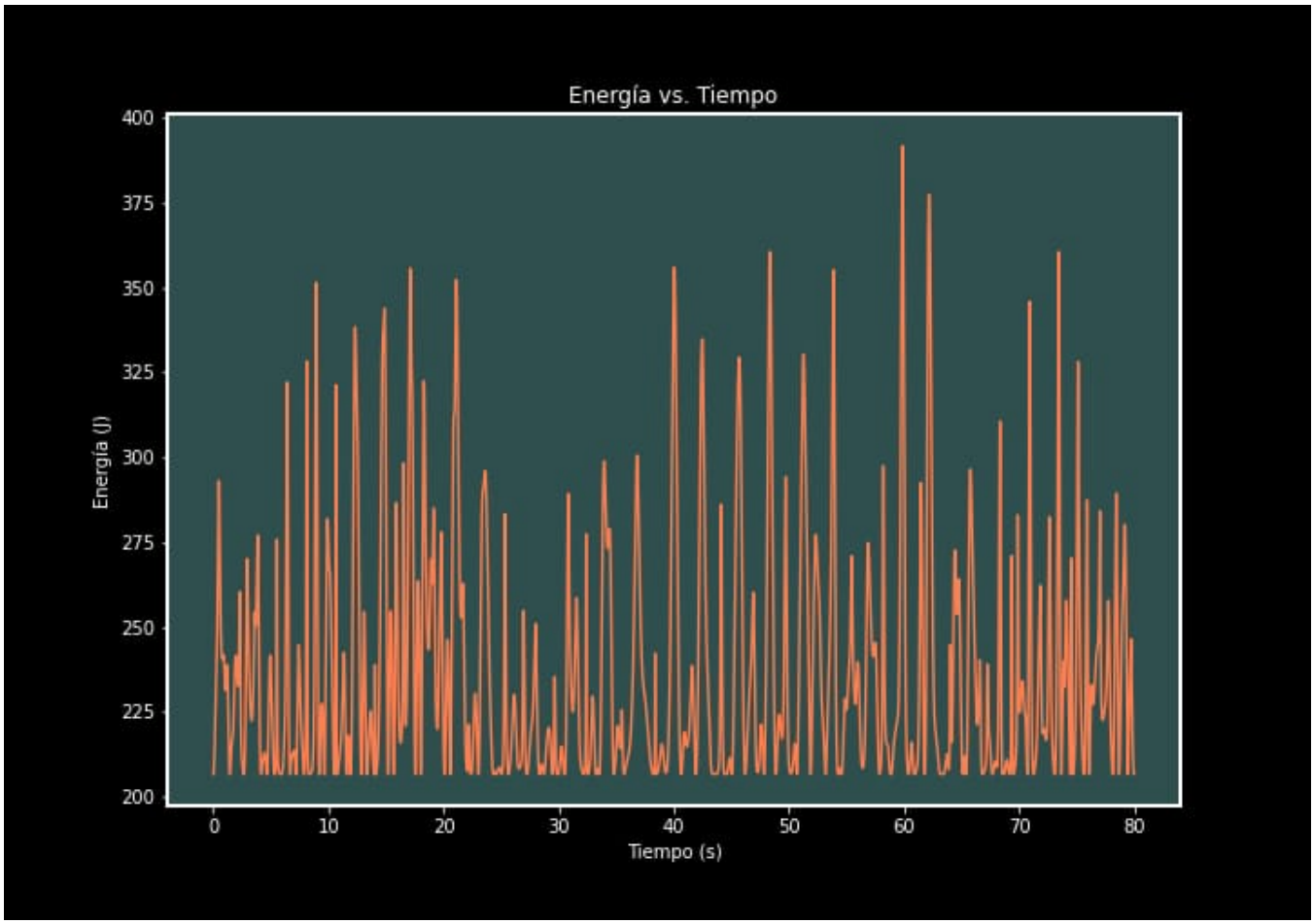

Figura 7. Energía del sistema vs tiempo.

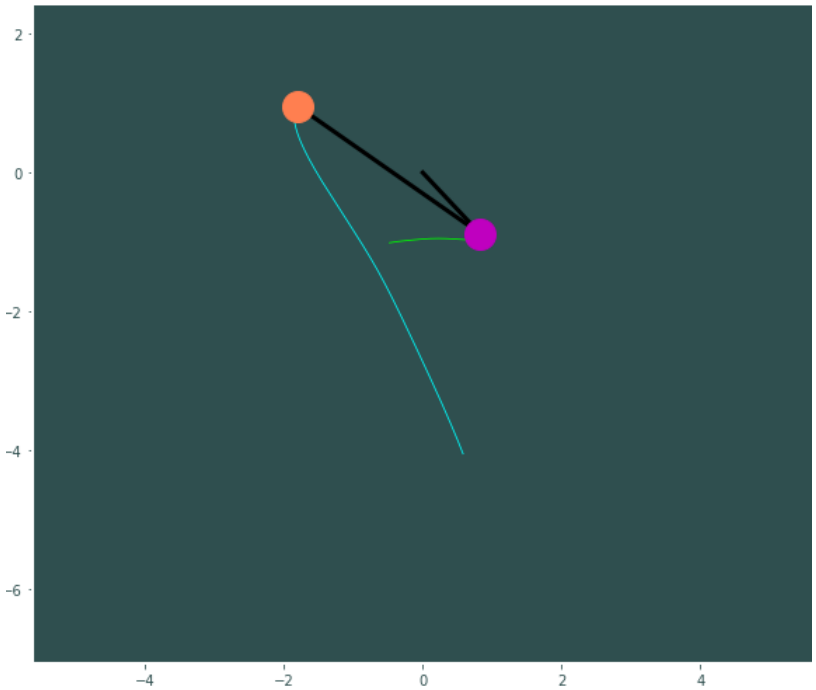

Figura 8. Una animación del sistema es obtenida acorde a las condiciones iniciales establecidas por el usuario

Bibliografía

Rañada, A. F. (1986). Movimiento caótico. Investigación y ciencia, 114, 12-23

RAMIREZ, R. D. M. (2018). Implementación de osciladores caóticos en sistemas embebidos y aplicaciones. Ensenada, Baja California, México:[sn].

Santos Burguete, C., Simarro Grande, J. P., y Fuertes Marrón, D. (2018). Física del caos. In Física del caos en la predicción meteorológica (pp. 49–65). Agencia Estatal de Meteorología.

Martin Rojas, D. M. (2023). Simulación del Péndulo Doble como Herramienta para la Enseñanza del Caos. Góndola, Enseñanza y Aprendizaje de las Ciencias, 18(Especial), 376–386. https://doi.org/10.14483/23464712.21395

Nenuwe, N. O. (2019). Application of Lagrange equations to 2D double spring-pendulum in generalized coordinates. Ruhuna Journal of Science, 10(2), 120. https://doi.org/10.4038/rjs.v10i2.78