Actividad 10: Animaciones con Matplotlib

Martin Alejandro Paredes Sosa

Marzo, 2016

1. Introducción

La matemática de un péndulo simple es, en general, compleja. Hacer suposiciones que simplifican la descripción nos permite resolver analíticamente las ecuaciones de movimiento. El péndulo simple, es una idealización se un péndulo real, pero en un sistema aislado donde se asume:

- La cuerda tiene una masa despreciable, es rígida y se mantiene tensa.
- El péndulo se maneja como una masa puntual.
- El movimiento es en dos dimensiones trazando un arco.
- No pierde energía por fricción o resistencia al aire.
- El campo gravitacional es uniforme.
- El soporte no se mueve.

La ecuación diferencial que representa el movimiento de un péndulo simple es:

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\sin\theta = 0 \tag{1}$$

donde g es la aceleración de la gravedad, ℓ es la longitud del péndulo y θ es el angulo de desplazamiento [**?**].

En esta práctica se nos pidió realizar una animación de un péndulo utilizando la biblioteca de `Matplotlib`.Haciendo uso del código de Jake Vanderplas para el péndulo doble [**?**] y el código de Matplotlib de subplots[**?**], se crearon las animaciones del péndulo y su espacio fase.

2. Ejercicio y Resultados

Esta actividad consistió en realizar un código en python que nos permitiera construir una animación del péndulo simple. Se hizo uso de la librería *matplotlib.animation* para crear la animación y poder observar el movimiento del péndulo con diferentes condiciones iniciales.

Los códigos que se utilizaron fueron los siguientes:

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation as an
from matplotlib.lines import Line2D
from scipy.integrate import odeint

#Definiendo las constantes de la Ec. Diferencial
g = 9.81
l = 1.0
b = 0.0 #Pendulo simple por lo tanto sin friccion
c = g/l

#Codiciones Iniciales
X_f1 =np.array([(170.0/180.0)*np.pi,(0./180.0)*np.pi])
t = np.linspace(-0.0*np.pi,5.0*np.pi,500) #Para generar la solucion

#Definicion de la ecuacion diderencial del pendulo
def p (y, t, b, c):
    theta, omega = y
    dy_dt = [omega,-b*omega -c*np.sin(theta)]
    return dy_dt

#====================================
# Trayectoria
y0 = X_f1                               # Punto de Inicio
X = odeint(p, y0, t, args=(b,c))

#====================================
#Graficar

class SubplotAnimation(an.TimedAnimation):
    def __init__(self):
        fig = plt.figure()
        ax1 = fig.add_subplot(1, 1, 1)

        self.t = np.linspace(0, 80, 400)
        self.x = X[:,0]
        self.y = X[:,1]

        self.line1 = Line2D([], [], color='black')
        self.line1a = Line2D([], [], color='red', linewidth=2)
        self.line1e = Line2D(
            [], [], color='red', marker='o', markeredgecolor='r')
        ax1.add_line(self.line1)
        ax1.add_line(self.line1a)
        ax1.add_line(self.line1e)
        ax1.set_xlim(-10, 10)
        ax1.set_ylim(-10, 10)
        ax1.set_aspect('equal', 'datalim')
```

```
50
51          an.TimedAnimation.__init__(self, fig, interval=50, blit=True)
52
53      def _draw_frame(self, framedata):
54          i = framedata
55          head = i - 1
56          #head_len = 10
57          head_slice = (self.t > self.t[i] - 1.0) & (self.t < self.t[i])
58
59          self.line1.set_data(self.x[:i], self.y[:i])
60          self.line1a.set_data(self.x[head_slice], self.y[head_slice])
61          self.line1e.set_data(self.x[head], self.y[head])
62
63      def new_frame_seq(self):
64          return iter(range(self.t.size))
65
66      def _init_draw(self):
67          lines = [self.line1, self.line1a, self.line1e]
68          for l in lines:
69              l.set_data([], [])
70
71  ani = SubplotAnimation()
72  #ani.save('test_sub.mp4')
73  plt.show()
```

Listing 1: Código Animacion_Fase.py

```
1   from numpy import sin, cos
2   import numpy as np
3   import matplotlib.pyplot as plt
4   import scipy.integrate as integrate
5   import matplotlib.animation as animation
6
7   class DoublePendulum:
8       """Double Pendulum Class
9
10      init_state is [theta1, omega1, theta2, omega2] in degrees,
11      where theta1, omega1 is the angular position and velocity of the first
12      pendulum arm, and theta2, omega2 is that of the second pendulum arm
13      """
14      def __init__(self,
15                   init_state = [120, 0, 0, 0],
16                   L1=1.0,  # length of pendulum 1 in m
17                   L2=0.0,  # length of pendulum 2 in m
18                   M1=1.0,  # mass of pendulum 1 in kg
19                   M2=1.0,  # mass of pendulum 2 in kg
20                   G=9.8,  # acceleration due to gravity, in m/s^2
21                   origin=(0, 0)):
22          self.init_state = np.asarray(init_state, dtype='float')
23          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):
        """compute the current x,y positions of the pendulum arms"""
        (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):
        """compute the energy of the current state"""
        (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 * y[0] + M2 * y[1])
        K = 0.5 * (M1 * np.dot(vx, vx) + M2 * np.dot(vy, vy))

        return U + K

    def dstate_dt(self, state, t):
        """compute the derivative of the given state"""
        (M1, M2, L1, L2, 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):
        """execute one time step of length dt and update state"""
        self.state = integrate.odeint(self.dstate_dt, self.state, [0, dt])
    [1]
        self.time_elapsed += dt

#------------------------------------------------------------
# set up initial state and global variables
pendulum = DoublePendulum([180.0, -250.0, 0.0, 0.0])
dt = 1./60. # 60 fps

#------------------------------------------------------------
# set up figure and animation
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal', autoscale_on=False,
                     xlim=(-2, 2), ylim=(-2, 2))
ax.grid()

line, = ax.plot([], [], 'o-', lw=2)
time_text = ax.text(0.02, 0.95, '', transform=ax.transAxes)
energy_text = ax.text(0.02, 0.90, '', transform=ax.transAxes)

def init():
    #initialize animation
    line.set_data([], [])
    time_text.set_text('')
    energy_text.set_text('')
    return line, time_text, energy_text

def animate(i):
    #perform animation step
    global pendulum, dt
    pendulum.step(dt)

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

```
121
122 # choose the interval based on dt and the time to animate one step
123 from time import time
124 t0 = time()
125 animate(0)
126 t1 = time()
127 interval = 1000 * dt - (t1 - t0)
128
129 ani = animation.FuncAnimation(fig, animate, frames=300,
130                               interval=interval, blit=True, init_func=init
    )
131
132 plt.show()
```

Listing 2: Código Animacion_Pendulo.py

El resultado que se obtuvo fue el siguiente:

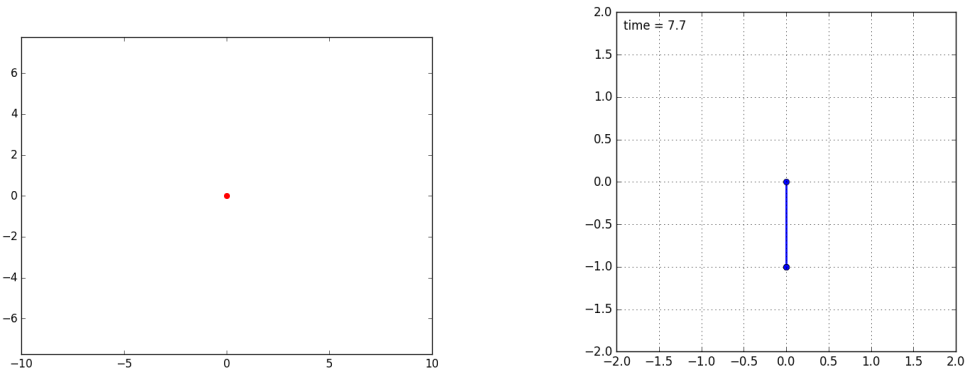

Figura 1: Imagen de Animación de péndulo con ángulo inicial 0.

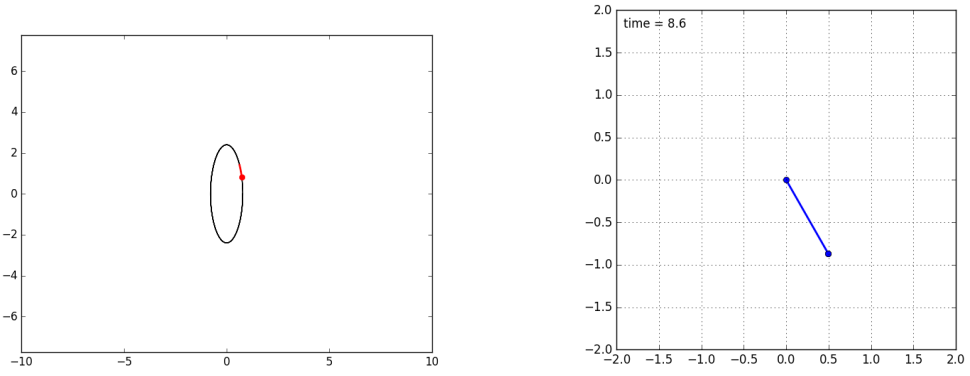

Figura 2: Imagen de Animación de péndulo con ángulo inicial 45.

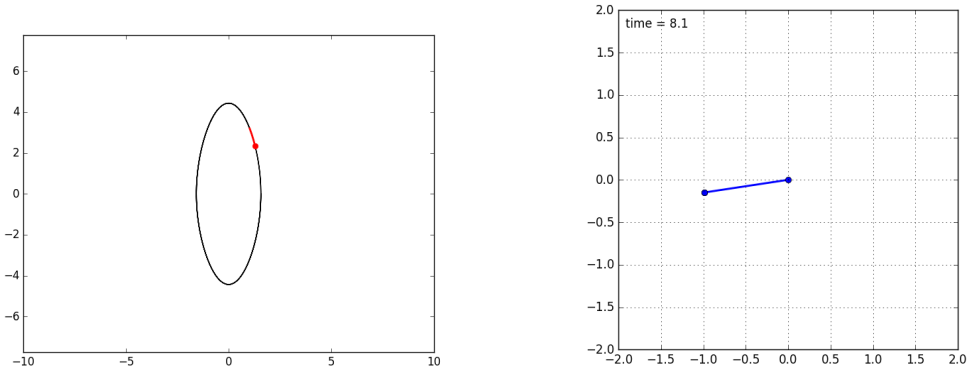

Figura 3: Imagen de Animación de péndulo con ángulo inicial 90.

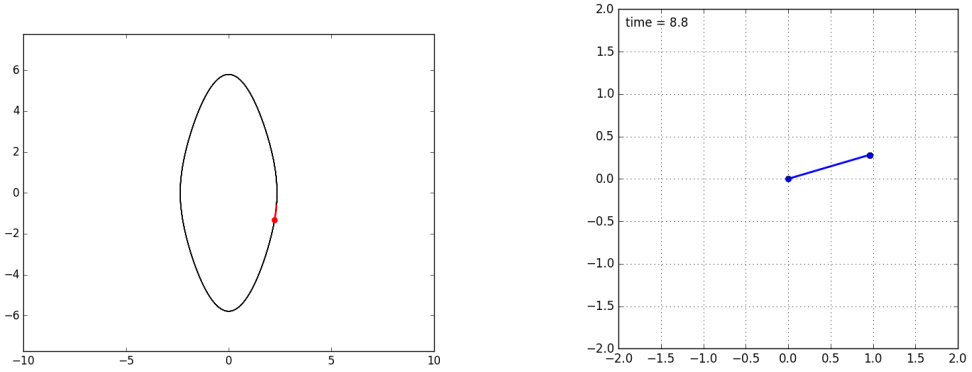

Figura 4: Imagen de Animación de péndulo con ángulo inicial 135.

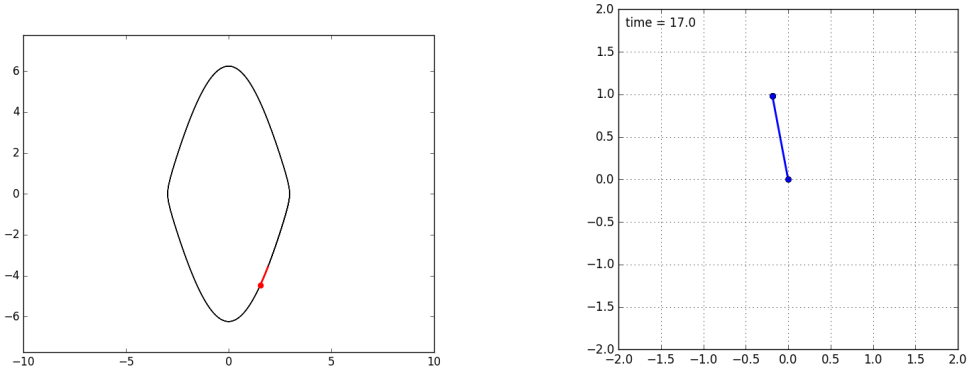

Figura 5: Imagen de Animación de péndulo con ángulo inicial 170.

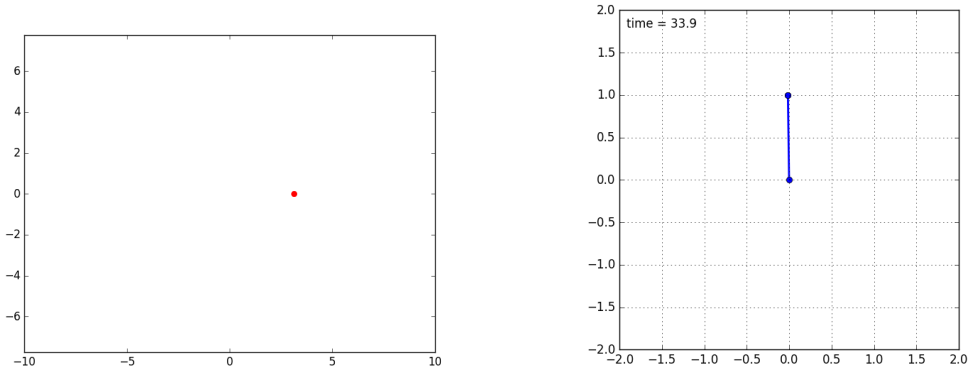

Figura 6: Imagen de Animación de péndulo con ángulo inicial 180.

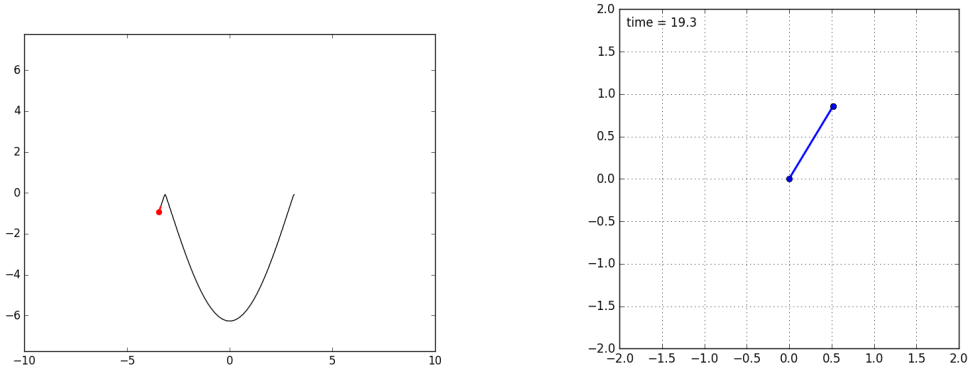

Figura 7: Imagen de Animación de péndulo con apenas energía para completar la vuelta.

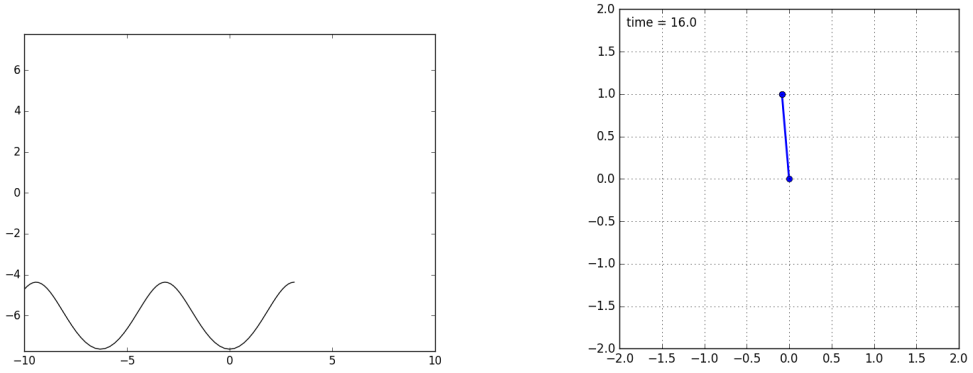

Figura 8: Imagen de Animación de péndulo con suficiente energía para completar la vuelta.

Se adjuntaron vídeos de las animaciones de cada uno de los casos.

Referencias

[1] Wikipedia,(2016) *Pendulum (mathematics)*. Recuperado de `https://en.wikipedia.org/wiki/Pendulum_%28mathematics%29`

[2] Vanderplas, J.(2012) *Matplotlib Animation Tutorial* Recuperado de `https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/`

[3] Matplotlib(2012) *animation example code: subplots.py* Recuperado de `http://matplotlib.org/examples/animation/subplots.html`

[4] Lizárraga, C. (2016) *Actividad 10 (2016-1)*. Recuperado de `http://computacional1.pbworks.com/w/page/107247876/Actividad%2010%20(2016-1)`