Universidad de Sonora

División de Ciencias Exactas y Naturales

Física Computacional I

# Práctica #10: Animaciones en matplotlib.

*Integrante:*
Alejandro Guirado García

2 de mayo de 2016

# 1. Objetivos

- En ésta actividad nos centraremos en animaciones apoyados en la biblioteca Matplotlib de Python.

# 2. Descripción de la actividad

Se pide adaptar cualquiera de los códigos mencionados en la actividad para reproducir animaciones del movimiento en el fase y el movimiento físico del péndulo. Se realizó con la biblioteca de Matplotlib junto con screencast programa sugerido por el profesor.
**Características del péndulo.**

- Largo= 1 metro.
- No se considera fricción
- Gravedad se considera con un valor de 9.8.

# 3. Códigos

## Código inicial

Es el código que se nos proporcionó primeramente, el cual modificamos para obtemer un péndulo simple.

```
General Numerical Solver for the 1D Time-Dependent Schrodinger's equation.

adapted from code at http://matplotlib.sourceforge.net/examples/animation/double_pe

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

author: Jake Vanderplas
email: vanderplas@astro.washington.edu
website: http://jakevdp.github.com
license: BSD
Please feel free to use and modify this, but keep the above information. Thanks!
"""
```

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

class DoublePendulum:
    """Double Pendulum Class

    init_state is [theta1, omega1, theta2, omega2] in degrees,
    where theta1, omega1 is the angular position and velocity of the first
    pendulum arm, and theta2, omega2 is that of the second pendulum arm
    """
    def __init__(self,
                 init_state = [120, 0, -20, 0],
                 L1=1.0,  # length of pendulum 1 in m
                 L2=1.0,  # length of pendulum 2 in m
                 M1=1.0,  # mass of pendulum 1 in kg
                 M2=1.0,  # mass of pendulum 2 in kg
                 G=9.8,  # acceleration due to gravity, in m/s^2
                 origin=(0, 0)):
        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):
        """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., 0.0, -20., 0.0])
dt = 1./30 # 30 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

# choose the interval based on dt and the time to animate one step
from time import time
t0 = time()
animate(0)
t1 = time()
interval = 1000 * dt - (t1 - t0)

ani = animation.FuncAnimation(fig, animate, frames=300,
                              interval=interval, blit=True, init_func=init)

# save the animation as an mp4.  This requires ffmpeg or mencoder to be
# installed.  The extra_args ensure that the x264 codec is used, so that
# the video can be embedded in html5.  You may need to adjust this for
# your system: for more information, see
# http://matplotlib.sourceforge.net/api/animation_api.html
#ani.save('double_pendulum.mp4', fps=30, extra_args=['-vcodec', 'libx264'])

plt.show()
```

El código que nos permite obtener el movimiento del péndulo simpĺe, el cuál realmente es el mismo código solamente modificando la longitu del segundo segmente del péndulo doble, el código utilizado es el siguiente:

```
#-*- coding: utf-8 -*-

from numpy import sin, cos
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import matplotlib.animation as animation

#Movimiento Fisico del Pendulo

class DoublePendulum:
"""Double Pendulum Class

init_state is [theta1, omega1, theta2, omega2] in degrees,
where theta1, omega1 is the angular position and velocity of the first
pendulum arm, and theta2, omega2 is that of the second pendulum arm
```

```
"""
def __init__(self,
init_state = [ 120 ,0, 0, 0],
L1=1.0,  # length of pendulum 1 in m
L2=0.0,  # largo
M1=100,  # mass of pendulum 1 in kg
M2=1.0,  # mass of pendulum 2 in kg
G=9.8,  # acceleration due to gravity, in m/s^2
origin=(0, 0)):
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):
"""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])])
y = np.cumsum([self.origin[1],
-L1 * cos(self.state[0])])
return (x, y)


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

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

return self.state


#------------------------------------------------------------
# set up initial state and global variables
pendulum = DoublePendulum([90, 15, 0., 0.0]) #theta1, omega1, theta2, omega2
dt = 1./30 # 30 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)) #tamao ejes
ax.grid()

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



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

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

line.set_data(*pendulum.position())
time_text.set_text('tiempo = %.1f' % pendulum.time_elapsed)
return line, time_text



# choose the interval based on dt and the time to animate one step
from time import time
t0 = time()
animate(0)
t1 = time()
interval = 1000 * dt - (t1 - t0)

ani = animation.FuncAnimation(fig, animate, frames=80,
interval=interval, blit=True, init_func=init)

ani.save('pendulo0', writer='ffmpeg')


plt.show()

##Espacio fase del mal
def pend(y, t, b, c):
theta, omega = y
dydt = [omega, -b*omega - c*np.sin(theta)]
return dydt

b = 0.0 #friccin
g = 9.8 #gravedad
l = 4.5 #longitud de la cuerda
c = g/l


y0 = [(90*np.pi)/180, 15.0]
```

```
t = np.linspace(0, 20, 100)

from scipy.integrate import odeint
sol = odeint(pend, y0, t, args=(b, c))




#Crear archivo
np.savetxt("90.txt", sol)

file = open("90.txt","r")
print file.read()

import matplotlib.pyplot as plt
import numpy as np


data = np.loadtxt('90.txt')

x1=data[:,0] #velocidad angular
y1=data[:,1] #posicin angular



x11 = x1.astype(np.float) #velocidad
y11 = y1.astype(np.float) #posicin


##Espacio fase del mal
def pend(y, t, b, c):
theta, omega = y
dydt = [omega, -b*omega - c*np.sin(theta)]
return dydt

b = 0.0 #friccin
g = 9.8 #gravedad
l = 4.5 #longitud de la cuerda
c = g/l
```

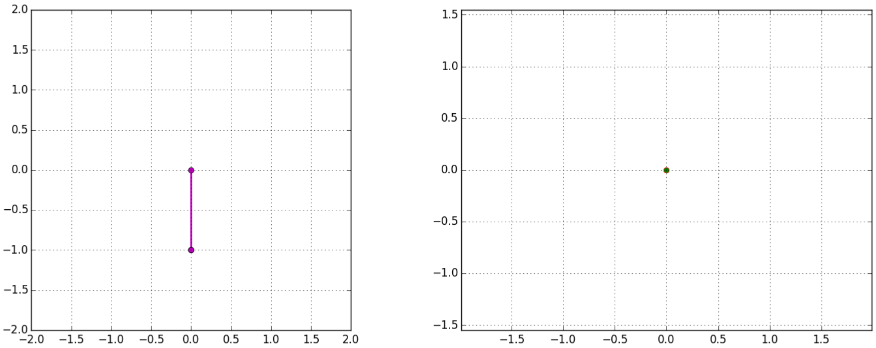
```
y0 = [(90*np.pi)/180, 15.0]

t = np.linspace(0, 20, 100)

from scipy.integrate import odeint
sol = odeint(pend, y0, t, args=(b, c))




#Crear archivo
np.savetxt("90.txt", sol)

file = open("90.txt","r")
print file.read()

import matplotlib.pyplot as plt
import numpy as np


data = np.loadtxt('90.txt')

x1=data[:,0] #velocidad angular
y1=data[:,1] #posicin angular



x11 = x1.astype(np.float) #velocidad
y11 = y1.astype(np.float) #posicin


#animacion espacio fase

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import matplotlib.animation as animation
```

```
class SubplotAnimation(animation.TimedAnimation):
def __init__(self):
fig = plt.figure()
#posicion de los cuadros de animacion
ax1 = fig.add_subplot(1, 1, 1) #plano xy
#ax2 = fig.add_subplot(2, 2, 2) #plano yz
#ax3 = fig.add_subplot(2, 2, 4) #plano xz


#funcin a graficar
self.t = np.linspace(0, 80, 300)
self.x = x11 #np.cos((np.pi/2) * self.t / 10.) #funcion eje x
self.y = y11 #np.sin((np.pi/2) * self.t / 10.) #funcin eje y
self.z = 5 * self.t


#caracteristicas animacion eje xy
ax1.set_xlabel('Posicion angular')
ax1.set_ylabel('Velocidad angular')
self.line1 = Line2D([], [], color='pink')
self.line1a = Line2D([], [], color='magenta', linewidth=2)
self.line1e = Line2D(
[], [], color='magenta', marker='o', markeredgecolor='b')
ax1.add_line(self.line1)
ax1.add_line(self.line1a)
ax1.add_line(self.line1e)
ax1.set_xlim(-4, 4)#tamano eje x
ax1.set_ylim(-8, 8)#tamao eje y
#ax1.set_aspect('equal', 'datalim')


animation.TimedAnimation.__init__(self, fig, interval=115, blit=115)

def _draw_frame(self, framedata):
i = framedata
head = i - 1
head_len = 10
head_slice = (self.t > self.t[i] - 1.0) & (self.t < self.t[i])
```

```
self.line1.set_data(self.x[:i], self.y[:i])
self.line1a.set_data(self.x[head_slice], self.y[head_slice])
self.line1e.set_data(self.x[head], self.y[head])

self._drawn_artists = [self.line1, self.line1a, self.line1e,
#self.line2, self.line2a, self.line2e,
#self.line3, self.line3a, self.line3e
]

def new_frame_seq(self):
return iter(range(self.t.size))

def _init_draw(self):
lines = [self.line1, self.line1a, self.line1e,
#self.line2, self.line2a, self.line2e,
#self.line3, self.line3a, self.line3e
]
for l in lines:
l.set_data([], [])

ani = SubplotAnimation()
#ani.save('penduloOF', writer='ffmpeg')


plt.show()
```

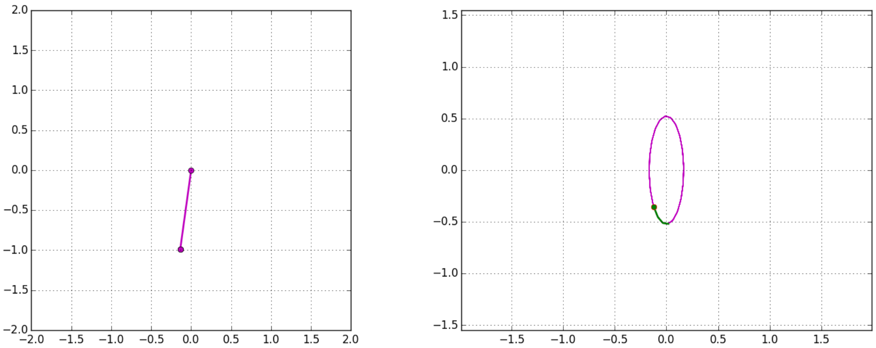
# 4. Resultados

A continuación se mostrarán una serie de imagénes sobre los resultados obtenidos.



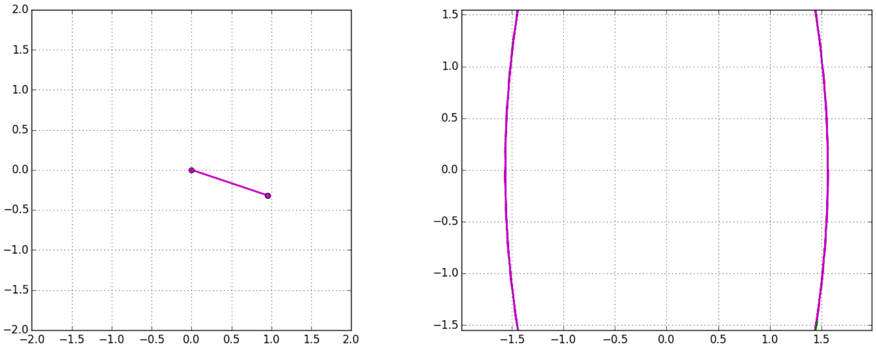

(a) Péndulo (b) Espacio fase.

Figura 1: $\theta_0 = 0 \quad v_0 = 0$.



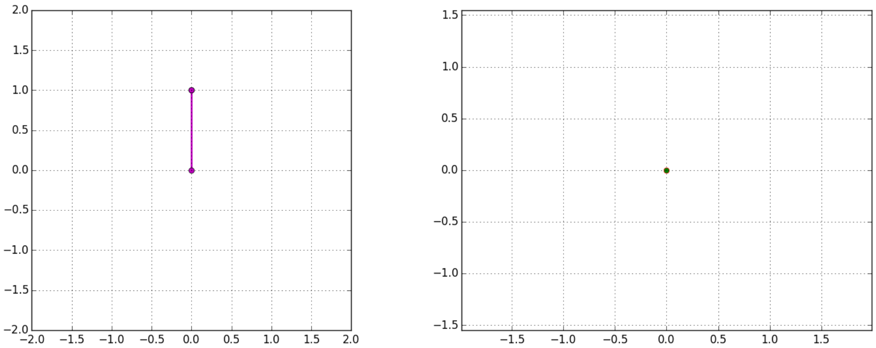

(a) Péndulo (b) Espacio fase.

Figura 2: $\theta_0 = 30 \quad v_0 = 0$.



(a) Péndulo (b) Espacio fase.

Figura 3: $\theta_0 = 90 \quad v_0 = 0.$



(a) Péndulo (b) Espacio fase.

Figura 4: $\theta_0 = 180 \quad v_0 = 0.$

# Referencias

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

[2] Lizárraga, C. *Actividad 10 (2016-1)*. Recuperado en Abril de 2016 de `http://computacional1.pbworks.com/w/page/107247876/Actividad%2010%20%282016-1%29)`