2-chaos

November 14, 2020

1 Caos en sistemas contínuos

[1]:
```
from numpy import sin, cos
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import matplotlib.animation as animation
from IPython.display import YouTubeVideo
#%pylab inline
```

[2]:
```
# El código siguiente recarga (reloads) las rutinas externas cada vez que el␣
↪código cambia (es útil para "debuggear" código externo)

%load_ext autoreload
%autoreload 2
```

[3]:
```
from utils import normalizeRads, normalizeAngle
```

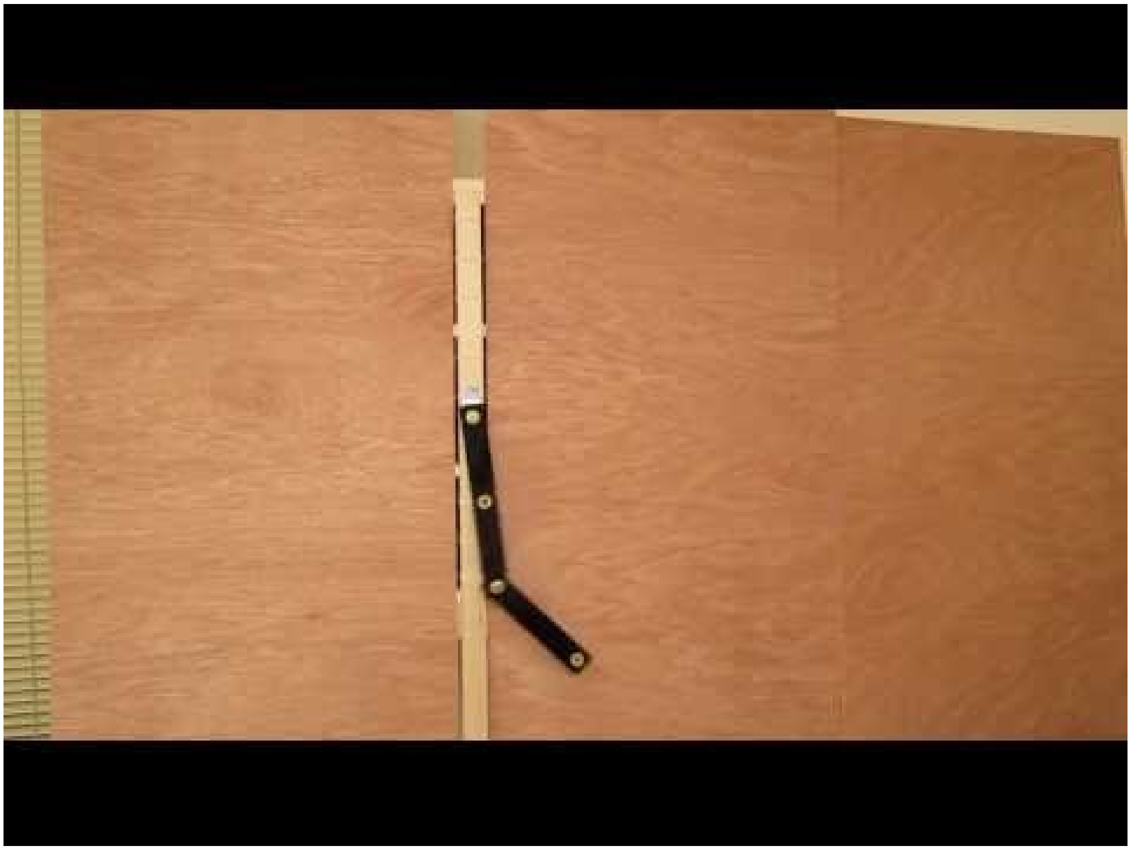
Bibliografía de soporte

- Una buena referencia para como deducir las ecuaciones de movimiento es: **Single and Double Pendulum** de *Gabriela González*.

1.0.1 La vida real

[4]:
```
YouTubeVideo("AwT0k09w-jw")
```

[4]:



1.1 El péndulo simple

Imagen cortesía de Wikipedia

1.1.1 Ecuación de movimiento

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

Análisis dimensional Un análisis poderoso (no numérico o computacional) que es bueno introducir aquí, es el *análisis dimensional*. No lo vamos a explotar en su totalidad, pero lo usaremos para comprobar la validez de las ecuaciones de movimiento.

Si las ecuaciones son correctas, ambos lados de la ecuación deben de tener las mismas unidades. Las unidades fundamentales son **longitud**, $[L]$, **masa**, $[M]$ y **tiempo** $[T]$.

Empezando por el lado izquierdo, g es la aceleración de la gravedad y tiene unidades de m/s^2 que en unidades fundamentales es:

$$[g] = \left[\frac{L}{T^2}\right]$$

Así mismo, la longitud del péndulo, l, tiene unidades de $m \to$

$$[l] = [L]$$

y θ es adimensional. Por lo tanto el **rhs** tiene unidades de

$$[rhs] = [T^{-2}]$$

Por su parte el **lhs**, θ como vimos es adimensional, pero está derivada respecto al tiempo, entonces,

$$\left[\frac{d}{dt^2}\right] = \left[\frac{1}{T^2}\right]$$

con lo cual se establece la igualdad de unidades, dándonos así un indicador de que al menos *no están tan mal*.

Nota El análisis dimensional es muy poderoso aunque con limitaciones. Usalo siempre que puedas para verificar tu trabajo.

1.1.2 Energía

La energía total, E, se descompone en energía cinética, K y energía potencial U.

$$E = K + U$$

La energía potencial depende de la posición en el campo gravitacional

$$U = -mgy$$

$$U = -mgl \sin\theta$$

Y la energía cinética

$$K = \frac{1}{2}mv^2$$

$$K = \frac{1}{2}m\left[\dot{x}^2 + \dot{y}^2\right]$$

$$K = \frac{1}{2}ml\dot{\theta}^2$$

Así,

$$E = \frac{1}{2}ml\dot{\theta}^2 - mgl\sin\theta$$

Ejercicio Revisa las ecuaciones con análisis dimensional y verifica que sean correctas

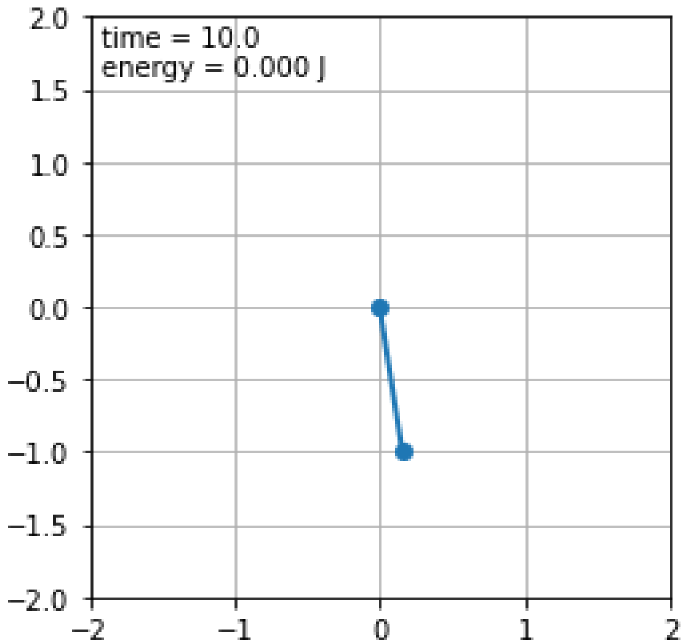
1.1.3 Animación

[5]:
```
from pendulo import Pendulo
from IPython.display import HTML
```

[6]:
```
pendulo = Pendulo(estado_inicial=[np.pi/2, 0])
fps = 30
```

[7]:
```
HTML(pendulo.animar(1./fps).to_html5_video())
```

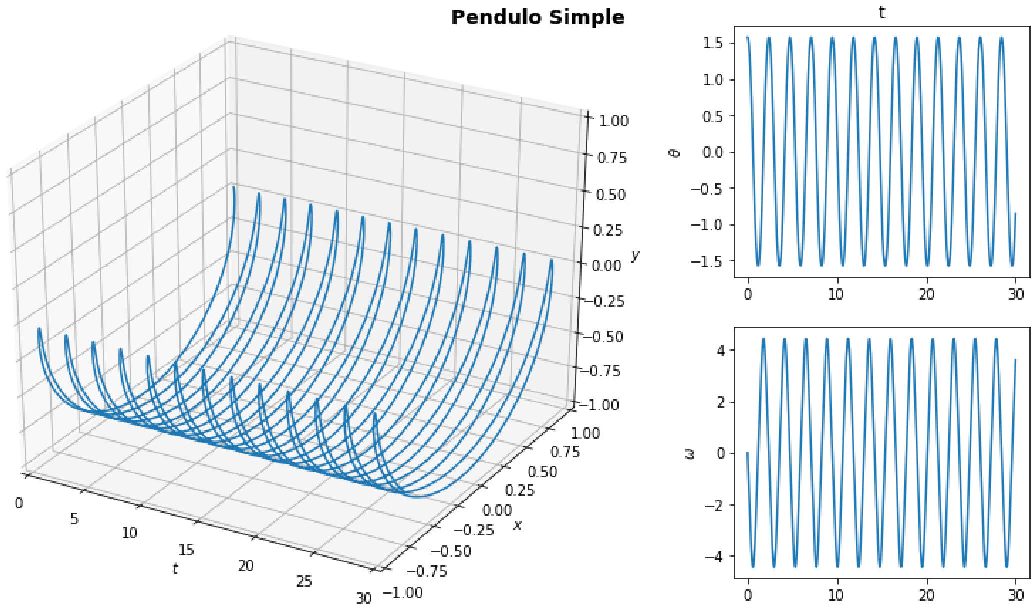
[7]: <IPython.core.display.HTML object>



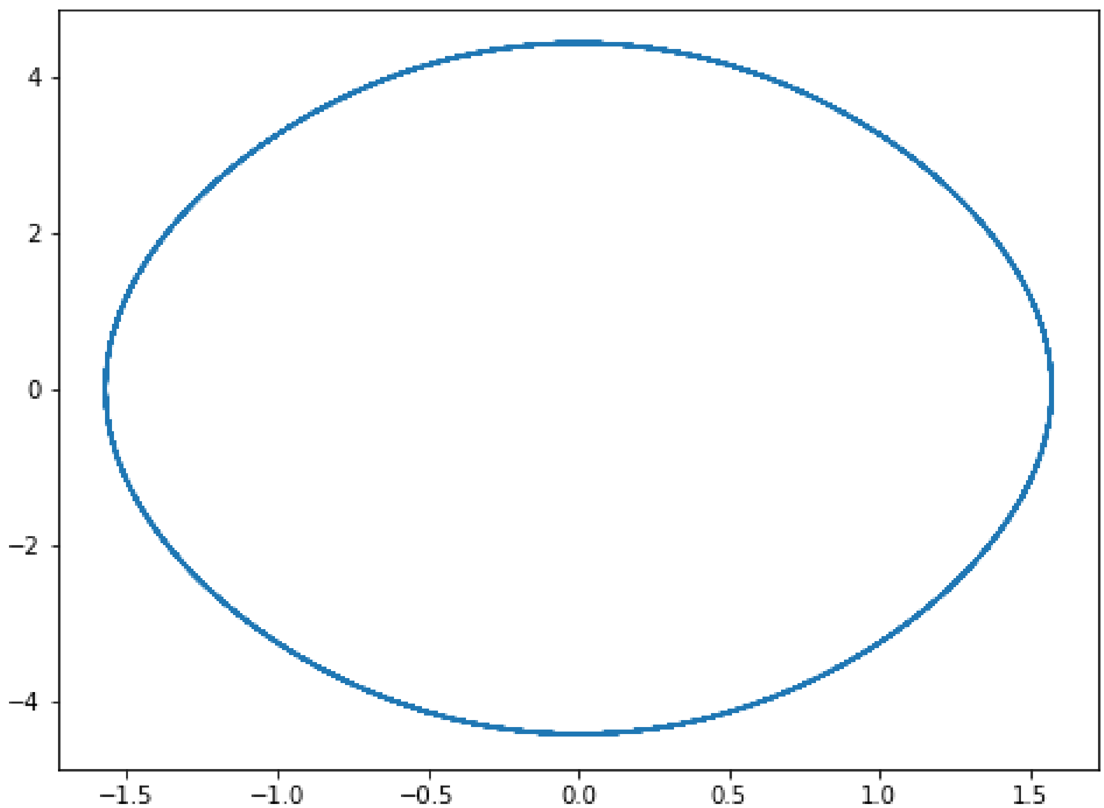
1.1.4 Análisis Gráfico

[8]:
```
pendulo.integrate()
```

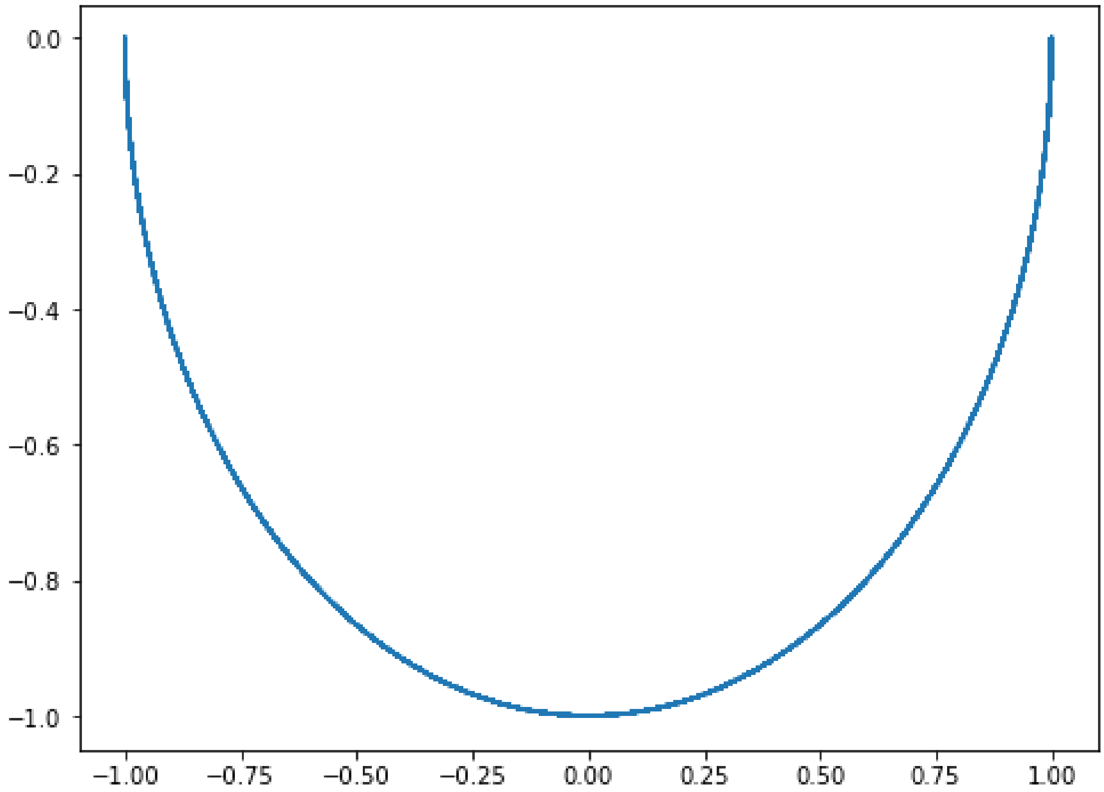
[9]:
```
pendulo.plot()
```



[10]:
```
pendulo.phase_space()
```



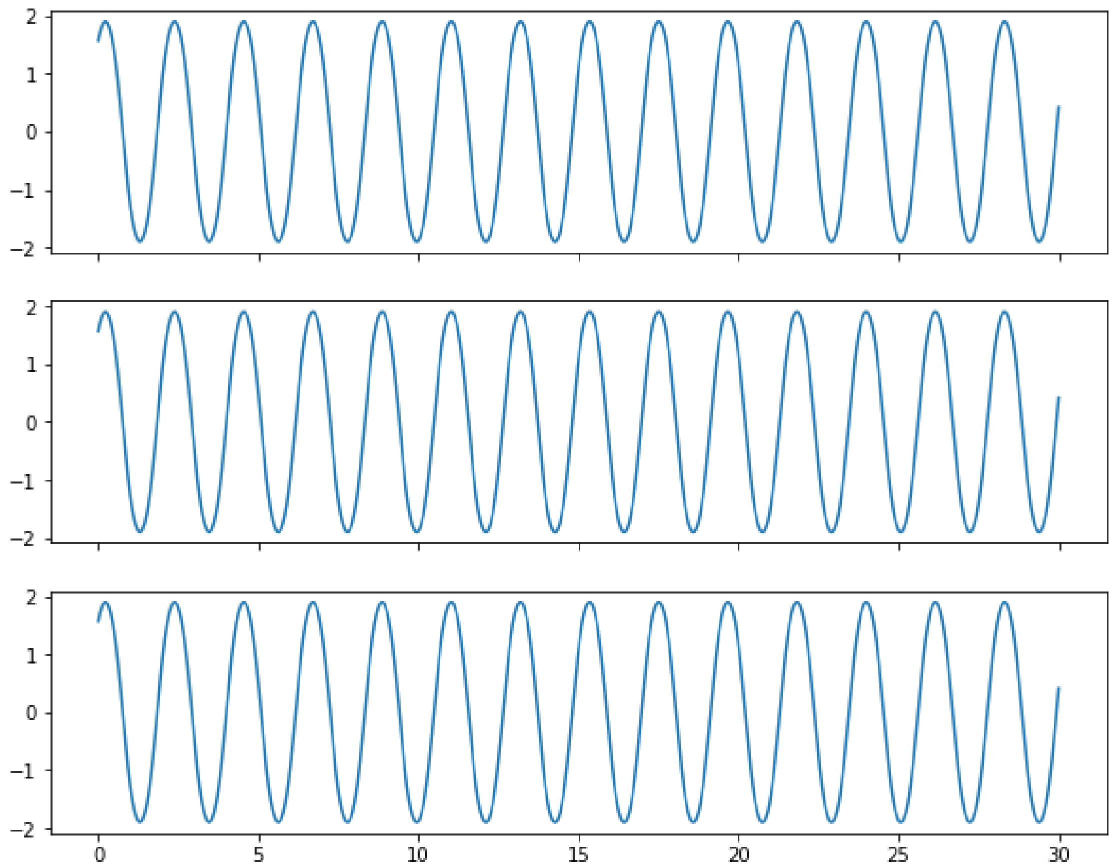
[11]:
```
pendulo.xy_snapshot()
```



1.1.5 Sensibilidad a condiciones iniciales

[12]:
```
pendulo1 = Pendulo([np.pi/2, 3.], longitud= 0.7)
pendulo1.integrate()
pendulo2 = Pendulo([np.pi/2, 3.0001], longitud= 0.7)
pendulo2.integrate()
pendulo3 = Pendulo([np.pi/2, 3.0002], longitud= 0.7)
pendulo3.integrate()
```

[13]:
```
fig, ax = plt.subplots(3,1, figsize=(10,8), sharex = True)

ax[0].plot(pendulo1.tau, pendulo1.theta())
ax[1].plot(pendulo2.tau, pendulo2.theta())
ax[2].plot(pendulo3.tau, pendulo3.theta())

plt.show()
```



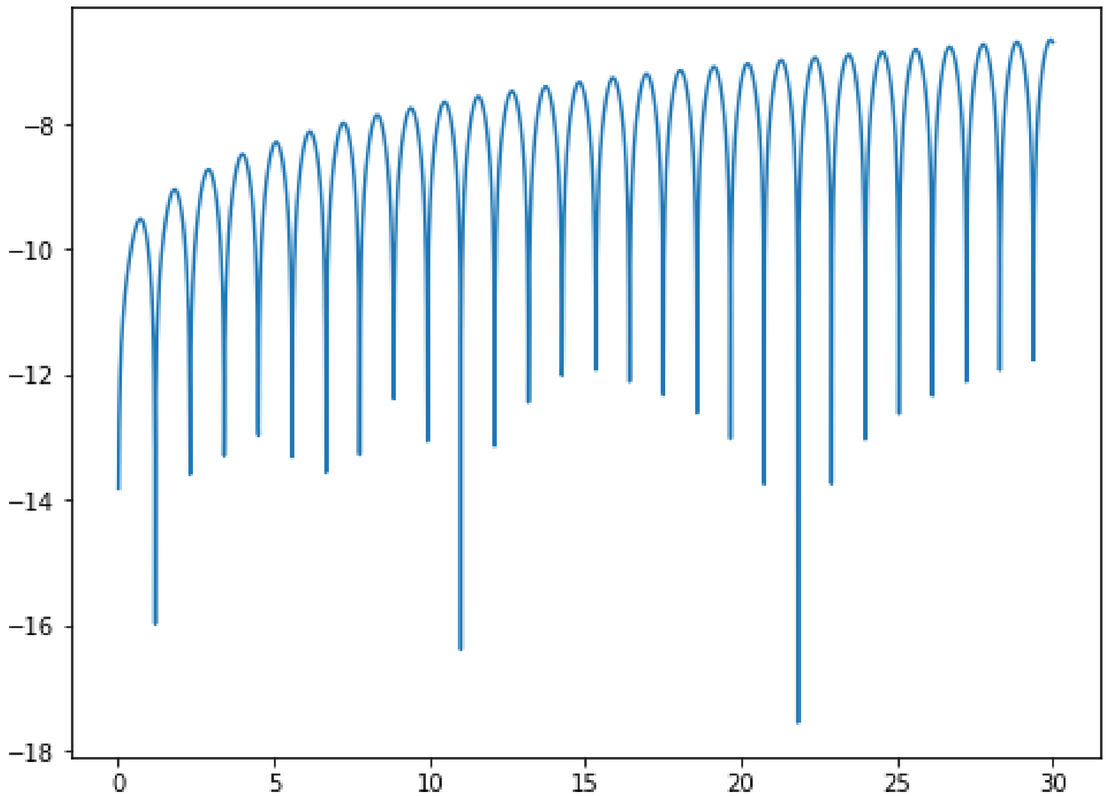
Evolución en el tiempo de la diferencia entre los ángulos

[18]:
```
delta_theta = abs(pendulo1.theta() - pendulo2.theta())
delta_theta
```

[18]:
```
array([0.00000000e+00, 1.00033697e-06, 2.00072148e-06, …,
       1.26328498e-03, 1.25331589e-03, 1.24169943e-03])
```

[19]:
```
plt.figure(1, figsize=(8,6))
plt.plot(pendulo1.tau, np.log(delta_theta))
plt.show()
```

```
<ipython-input-19-511469f95b97>:2: RuntimeWarning: divide by zero encountered in
log
  plt.plot(pendulo1.tau, np.log(delta_theta))
```



1.2 El péndulo doble

Imagen cortesía de Wikipedia

1.2.1 Ecuaciones de movimiento

$$(m_1 + m_2)L_1\ddot{\theta_1} + m_2L_2\ddot{\theta_2}\cos(\theta_2 - \theta_1) = m_2L_2\dot{\theta_2}^2\sin(\theta_2 - \theta_1) - (m_1 + m_2)g\sin\theta_1$$

$$L_2\ddot{\theta_2} + L_1\ddot{\theta_1}\cos(\theta_2 - \theta_1) = -L_1\dot{\theta_1}^2\sin(\theta_2 - \theta_1) - g\sin\theta_2$$

Ejercicio Revisa las ecuaciones con análisis dimensional y verifica que sean correctas

Ejercicio Intenten escribir como una ecuación de primer grado.

Nota El truco se puede ver aquí

1.2.2 Energía

La energía potencial es:

$$V = -m_1gy_1 - m_2gy_2$$

$$V = -(m_1 + m_2)gL_1 \cos\theta_1 - m_2 g L_2 \cos\theta_2$$

La energía cinética es:

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

$$K = \frac{1}{2}m_1 L_1^2 \dot{\theta_1}^2 + \frac{1}{2}m_2 \left[L_1^2 \dot{\theta_1}^2 + L_2^2 \dot{\theta_2}^2 + 2L_1 L_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)\right]$$

Finalmente la energía total, E, es

$$E = K + V$$

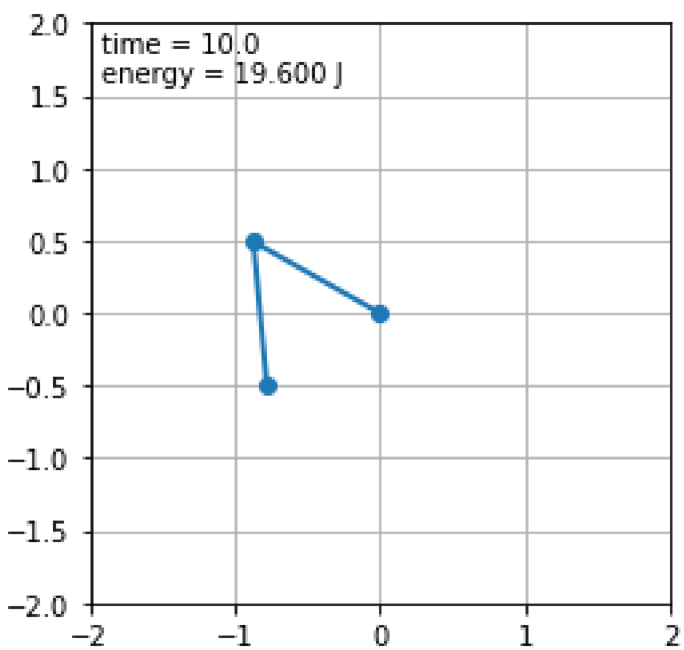
1.2.3 Animación

[20]:
```python
from pendulo_doble import PenduloDoble
```

[21]:
```python
d_pendulo = PenduloDoble()
fps = 30
```

[22]:
```python
HTML(d_pendulo.animar(1./fps).to_html5_video())
```

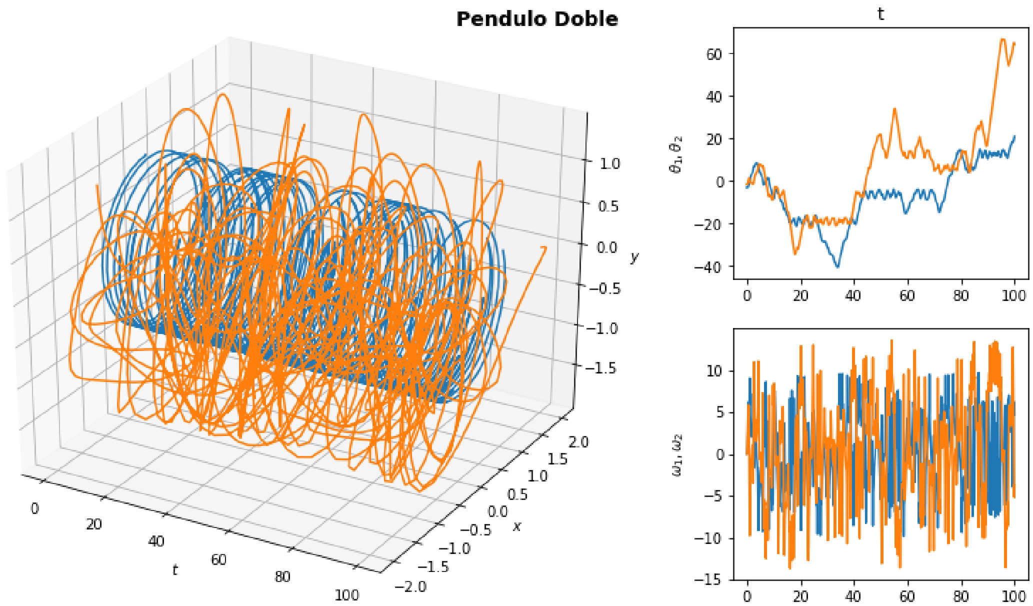
[22]: <IPython.core.display.HTML object>



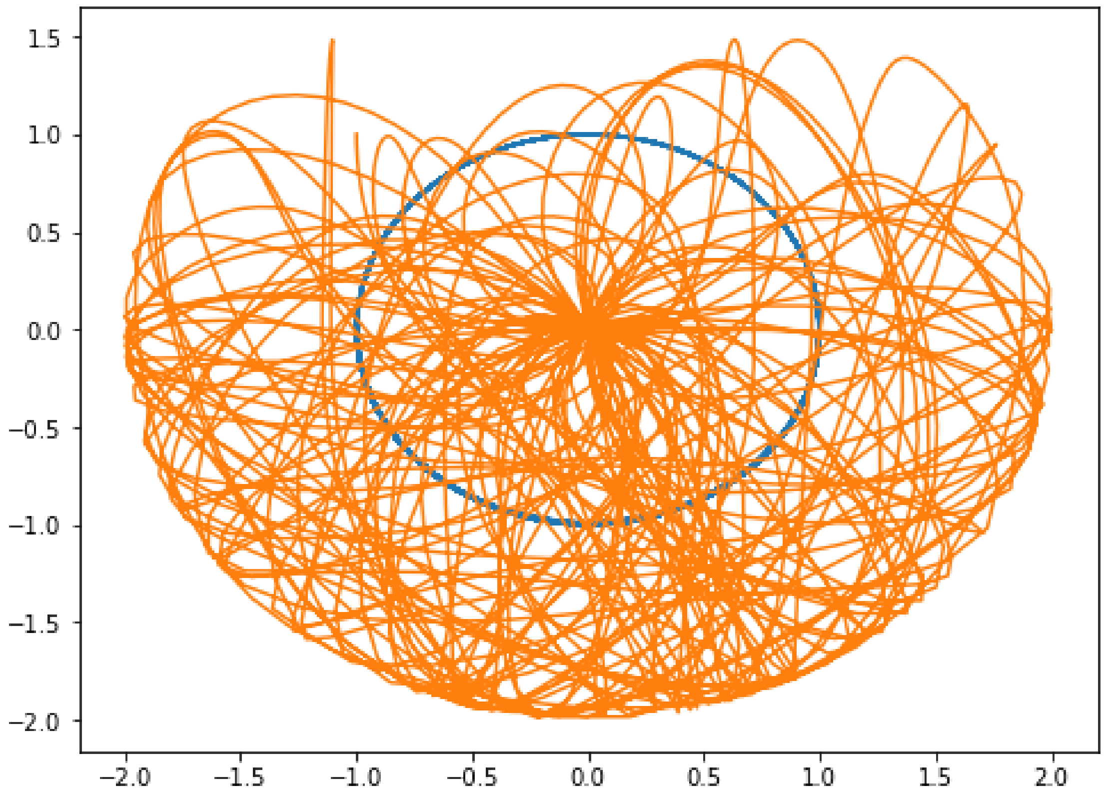
1.2.4 Análisis gráfico

[23]:
```
d_pendulo = PenduloDoble()
d_pendulo.integrate(t_f=100)
```

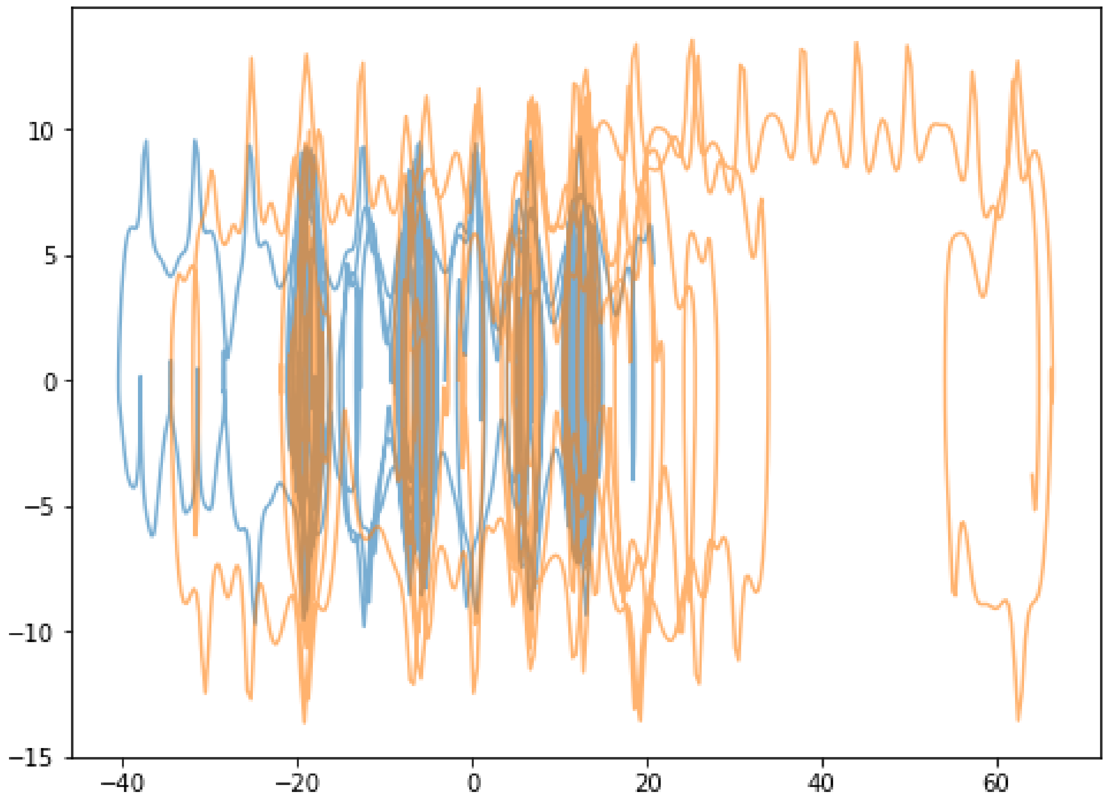
[24]:
```
d_pendulo.plot()
```



[25]:
```
d_pendulo.xy_snapshot()
```



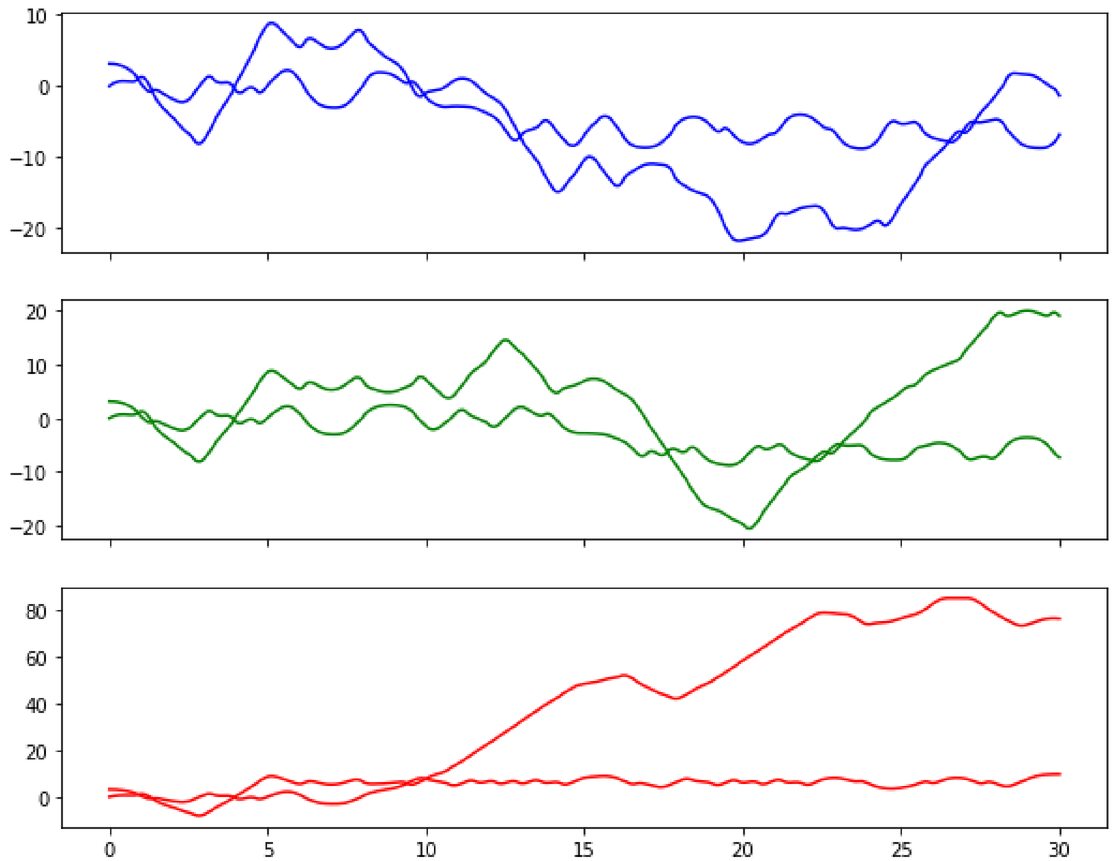
[26]:
```
d_pendulo.phase_space()
```



1.3 Sensibilidad a condiciones iniciales

[27]:
```
d_pendulo1 = PenduloDoble([np.pi, 0.,0., 3.], L1= 0.7, L2=0.7)
d_pendulo1.integrate()
d_pendulo2 = PenduloDoble([np.pi, 0.,0., 3.0001], L1= 0.7, L2=0.7)
d_pendulo2.integrate()
d_pendulo3 = PenduloDoble([np.pi, 0.,0., 3.0002], L1= 0.7, L2=0.7)
d_pendulo3.integrate()
```

[28]:
```
fig, ax = plt.subplots(3,1, figsize=(10,8), sharex = True)

ax[0].plot(d_pendulo1.tau, d_pendulo1.theta(), label="theta", color="blue")
ax[1].plot(d_pendulo2.tau, d_pendulo2.theta(), label="omega", color="green")
ax[2].plot(d_pendulo3.tau, d_pendulo3.theta(), color='red', label="Energia")

plt.show()
```

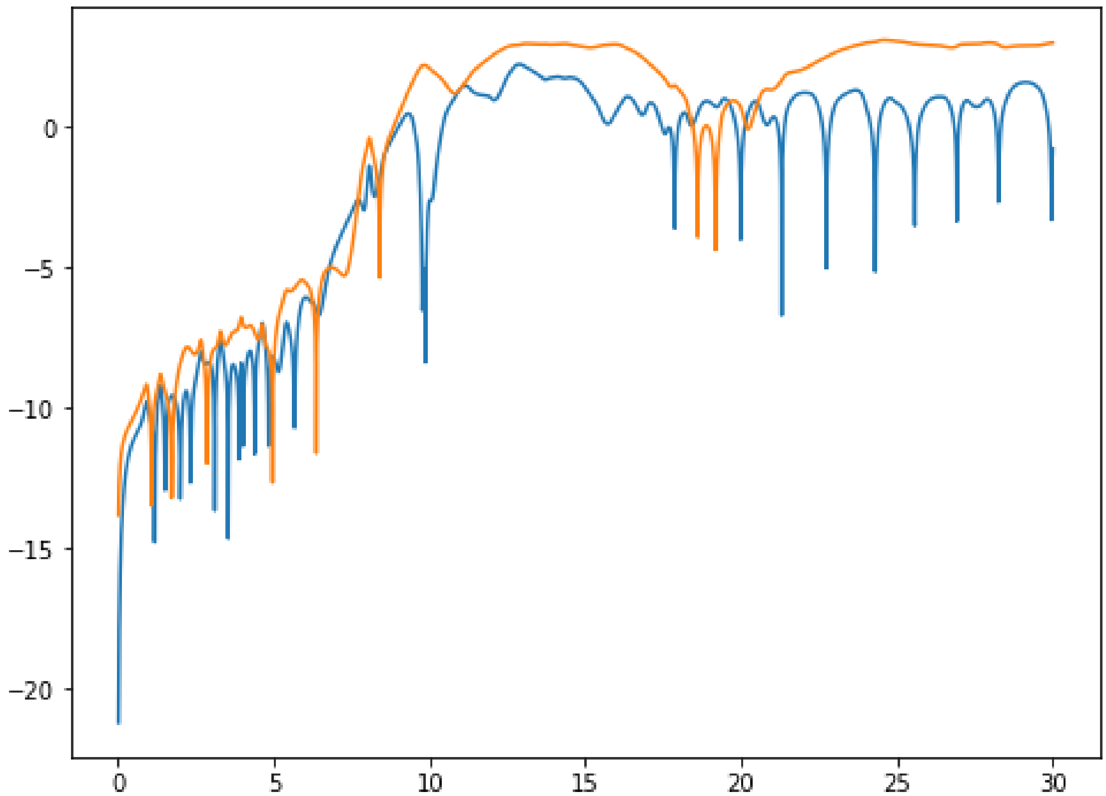


Evolución en el tiempo de la diferencia entre los ángulos

[29]:
```
delta_theta = abs(d_pendulo1.theta() - d_pendulo2.theta())
```

[30]:
```
plt.figure(1, figsize=(8,6))
plt.plot(d_pendulo1.tau, np.log(delta_theta))
plt.show()
```

```
<ipython-input-30-2ff8ffcb2a5e>:2: RuntimeWarning: divide by zero encountered in
log
  plt.plot(d_pendulo1.tau, np.log(delta_theta))
```

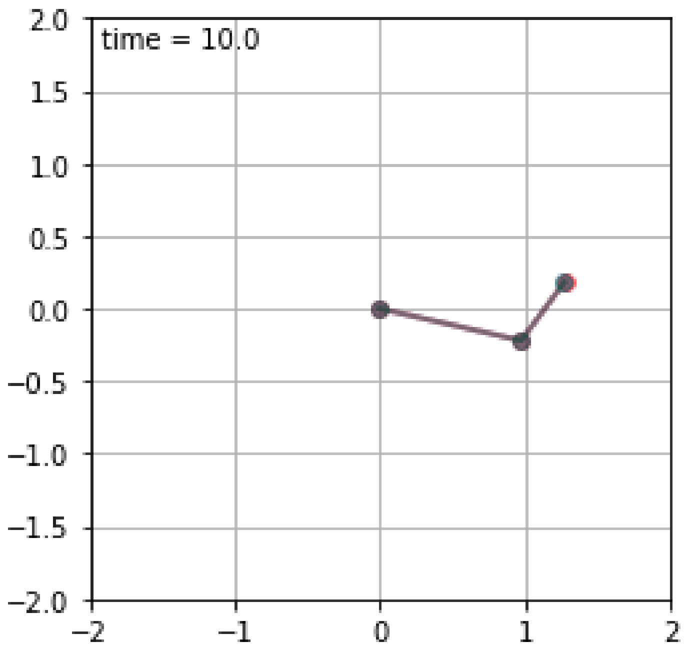


1.3.1 Sensibilidad ante las condiciones iniciales: Animación

[31]:
```
d_pendulo1 = PenduloDoble([4, 0.,3., 0.], M1=0.5, M2=1., L1= 1., L2=0.5)
d_pendulo2 = PenduloDoble([4, 0.,3.00000003, 0.], M1=0.5, M2=1., L1= 1., L2=0.5)
d_pendulo3 = PenduloDoble([4, 0.,3.00000004, 0.], M1=0.5, M2=1., L1= 1., L2=0.5)

dt = 1./30

fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal', autoscale_on=False,
                     xlim=(-2, 2), ylim=(-2, 2))

ax.grid()

line1, = ax.plot([], [], 'o-', lw=2, color="red", alpha=0.5)
line2, = ax.plot([], [], 'o-', lw=2, color="blue", alpha=0.3)
line3, = ax.plot([], [], 'o-', lw=2, color="green", alpha=0.3)
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"""
    line1.set_data([], [])
    line2.set_data([], [])
    line3.set_data([], [])
    time_text.set_text('')
#    energy_text.set_text('')
    return line1, line2, line3, time_text#, energy_text

def animate(i):
    """perform animation step"""
    global d_pendulo1, d_pendulo2, dt
    d_pendulo1.step(dt)
    d_pendulo2.step(dt)
    d_pendulo3.step(dt)

    line1.set_data(*d_pendulo1.posicion())
    line2.set_data(*d_pendulo2.posicion())
    line3.set_data(*d_pendulo3.posicion())
    time_text.set_text('time = %.1f' % d_pendulo1.time_elapsed)
    #energy_text.set_text('energy = %.3f J' % self.energia())
    return line1, line2, line3, time_text #, energy_text

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)

HTML(ani.to_html5_video())
```

[31]: <IPython.core.display.HTML object>



[32]:
```
%cat pendulo.py
```

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

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

import mpl_toolkits.mplot3d.axes3d as a3d

from numpy import sin, cos

from utils import normalizeRads, normalizeAngle
from scipy.integrate import odeint

class Pendulo:
    """
    Péndulo
    -----------------------

    El estado inicial está dado por [theta, omega] ambos en radianes.
    Theta y omega son la posición angular y la velocidad angular
respectivamente.
    """
    def __init__(self,
                 estado_inicial = [-np.pi/6, 0.],
```

```
                 masa = 1.0, # masa en kg
                 longitud = 1.0, # longitud en m
                 gravedad = 9.8, # aceleración de la gravedad, en m/s^2
                 origen=(0,0)):

        self.estado_inicial = np.asarray(estado_inicial, dtype="float")
        self.estado_inicial = normalizeRads(self.estado_inicial, decimals=8) #
normalizamos para estar entre [-pi, pi)

        self.params = (longitud, masa, gravedad)
        self.origen = origen
        self.time_elapsed = 0

        self.estado = self.estado_inicial

        self.trayectoria = self.estado


    def x(self):

        (longitud, masa, gravedad) = self.params

        return longitud*sin(self.theta())



    def y(self):

        (longitud, masa, gravedad) = self.params

        return -longitud*cos(self.theta())


    def theta(self):
        return self.trayectoria[:,0]

    def omega(self):
        return self.trayectoria[:,1]


    def dinamica(self, state, t):
        """ Ecuaciones de movimiento """

        (longitud, masa, gravedad) = self.params

        dydx = np.zeros_like(state)

        dydx[0] = state[1]
```

```
        dydx[1] = -(gravedad/longitud)*sin(state[0])

        return dydx


    def posicion(self):
        """ Calcula la posición x,y actual del péndulo. """
        (longitud, masa, gravedad) = self.params

        x = [self.origen[0], self.origen[0] + longitud*sin(self.estado[0])]
        y = [self.origen[1], self.origen[1] - longitud*cos(self.estado[0])]

        return (x,y)

    def energia(self):
        """ Calcula la energía mecánica """
        (longitud, masa, gravedad) = self.params

        U = - masa * gravedad * longitud * cos(self.estado[0])

        K = 0.5 * masa * longitud**2 * self.estado[1]**2

        return U + K


    def step(self, dt):
        """ Ejecuta un paso de tiempo, dt, y actualiza el estado. """
        self.estado = odeint(self.dinamica, self.estado, [0,dt])[1]

        self.time_elapsed += dt



    def integrate(self, num_steps=3000, t_i=0, t_f=30):
        """ Resuelve la dinámica en el delta de tiempo especificado """
        self.tau = np.linspace(t_i, t_f, num=num_steps)

        self.trayectoria = odeint(self.dinamica, self.estado_inicial, self.tau)


    def plot(self):
        """ Basado en http://debtechandstuff.blogspot.mx/2009/10/creating-video-
of-3d-graph-plotting.html """

        fig=plt.figure(figsize=(12,6))
        fig.suptitle("Pendulo Simple", fontsize=14, fontweight='bold')
```

```
        ax = a3d.Axes3D(fig,rect=[0,0,0.6,1])
        ax.set_autoscale_on(False)
        ax.set_xlim3d((0,30))
        ax.set_ylim3d((-1,1))
        ax.set_zlim3d((-1,1))
        ax.set_xlabel(r'$t$')
        ax.set_ylabel(r'$x$')
        ax.set_zlabel(r'$y$')
        ax.plot3D(self.tau, self.x(), self.y())

        fig.subplots_adjust(left=0.66,bottom=0.05,top=0.95)

        bx = fig.add_subplot(211)
        bx.set_autoscale_on(True)
        bx.set_ylabel(r'$\theta$')
        bx.set_title('t')
        bx.plot(self.tau,self.theta())

        cx = fig.add_subplot(212)
        cx.set_autoscale_on(True)
        cx.set_ylabel(r'$\omega$')
        cx.plot(self.tau,self.omega())

        plt.show()

    def phase_space(self):
        plt.figure(1, figsize=(8,6))
        plt.plot(self.theta(), self.omega())

        plt.show()

    def xy_snapshot(self):
        plt.figure(1, figsize=(8,6))
        plt.plot(self.x(), self.y())

        plt.show()


    def animar(self, dt):
        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"""
            self.step(dt)

            line.set_data(*self.posicion())
            time_text.set_text('time = %.1f' % self.time_elapsed)
            energy_text.set_text('energy = %.3f J' % self.energia())
            return line, time_text, energy_text

        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)

        return ani
```

[38]:

```
%cat pendulo_doble.py
```

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

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

import mpl_toolkits.mplot3d.axes3d as a3d

from numpy import sin, cos

from utils import normalizeRads, normalizeAngle
from scipy.integrate import odeint

class PenduloDoble:
    """
    PenduloDoble
    ------------------------------
```

```
    Basado en el código de https://jakevdp.github.io/blog/2012/08/18/matplotlib-
animation-tutorial/

    El estado inicial está dado por [theta1, omega1, theta2, omega2] en
radianes,
    theta1 y omega1 son la posición y la velocidad angular de la primera masa y
    theta2 y omega2 son la posición y la velocidad angular de la segunda masa.
    """

    def __init__(self,
                 estado_inicial =  [-np.pi, -0.0, -np.pi/2, 0.0],
                 L1=1.0,  # longitud del primer brazo en m
                 L2=1.0,  # longitud del primer brazo en m
                 M1=1.0,  # masa del primer péndulo en kg
                 M2=1.0,  # masa del segundo péndulo en kg
                 G=9.8,  # aceleración de la gravedad en m/s^2
                 origen=(0, 0)):
        self.estado_inicial = np.asarray(estado_inicial, dtype="float")
        self.estado_inicial = normalizeRads(self.estado_inicial, decimals=8) #
normalizamos para estar entre [-pi, pi)


        self.params = (L1, L2, M1, M2, G)
        self.origen = origen
        self.time_elapsed = 0

        self.estado = self.estado_inicial

        self.trayectoria = self.estado


    def x1(self):
        (L1, L2, M1, M2, G) = self.params

        return L1 * sin(self.theta()[:,0])

    def x2(self):
        (L1, L2, M1, M2, G) = self.params

        return L1 * sin(self.theta()[:,0]) + L2 * sin(self.theta()[:,1])

    def y1(self):
        (L1, L2, M1, M2, G) = self.params

        return -L1 * cos(self.theta()[:,0])

    def y2(self):
        (L1, L2, M1, M2, G) = self.params
```

```
        return -L1 * cos(self.theta()[:,0]) -L2 * cos(self.theta()[:,1])


    def theta(self):
        return self.trayectoria[:, [0,2]]

    def omega(self):
        return self.trayectoria[:, [1,3]]

    def dinamica(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 path(self):
        """ Devuelve el path x,y actual de los brazos del péndulo. """
        (L1, L2, M1, M2, G) = self.params

        x = np.cumsum([L1 * sin(self.trayectoria[:,0]),
                       L2 * sin(self.trayectoria[:,2])], axis=0)
        y = np.cumsum([-L1 * cos(self.trayectoria[:,0]),
                       -L2 * cos(self.trayectoria[:,2])], axis=0)
        return (x, y)


    def posicion(self):
        """ Calcula la posición x,y actual de los brazos del péndulo. """
```

```
        (L1, L2, M1, M2, G) = self.params

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

    def energia(self):
        """ Calcula la energía mecánica """
        (L1, L2, M1, M2, G) = self.params

        x = np.cumsum([L1 * sin(self.estado[0]),
                       L2 * sin(self.estado[2])])
        y = np.cumsum([-L1 * cos(self.estado[0]),
                       -L2 * cos(self.estado[2])])
        vx = np.cumsum([L1 * self.estado[1] * cos(self.estado[0]),
                        L2 * self.estado[3] * cos(self.estado[2])])
        vy = np.cumsum([L1 * self.estado[1] * sin(self.estado[0]),
                        L2 * self.estado[3] * sin(self.estado[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 step(self, dt):
        """ Ejecuta un paso de tiempo, dt, y actualiza el estado. """
        self.estado = odeint(self.dinamica, self.estado, [0, dt])[1]

        self.time_elapsed += dt


    def integrate(self, num_steps=3000, t_i=0, t_f=30):
        """ Resuelve la dinámica en el delta de tiempo especificado """
        self.tau = np.linspace(t_i, t_f, num=num_steps)

        self.trayectoria = odeint(self.dinamica, self.estado_inicial, self.tau)

    def plot(self):
        """ Basado en http://debtechandstuff.blogspot.mx/2009/10/creating-video-
of-3d-graph-plotting.html """

        fig=plt.figure(figsize=(12,6))
        fig.suptitle("Pendulo Doble", fontsize=14, fontweight='bold')
```

```
    ax = a3d.Axes3D(fig,rect=[0,0,0.6,1])
    ax.set_autoscale_on(True)
    ax.set_xlabel(r'$t$')
    ax.set_ylabel(r'$x$')
    ax.set_zlabel(r'$y$')
    ax.plot3D(self.tau, self.x1(), self.y1())
    ax.plot3D(self.tau, self.x2(), self.y2())

    fig.subplots_adjust(left=0.66,bottom=0.05,top=0.95)

    bx = fig.add_subplot(211)
    bx.set_autoscale_on(True)
    bx.set_ylabel(r'$\theta_1, \theta_2$')
    bx.set_title('t')
    bx.plot(self.tau,self.theta())

    cx = fig.add_subplot(212)
    cx.set_autoscale_on(True)
    cx.set_ylabel(r'$\omega_1, \omega_2$')
    cx.plot(self.tau,self.omega())

    plt.show()

def xy_snapshot(self):
    plt.figure(1, figsize=(8,6))
    plt.plot(self.x1(), self.y1())
    plt.plot(self.x2(), self.y2())

    plt.show()

def phase_space(self):
    plt.figure(1, figsize=(8,6))
    plt.plot(self.theta(), self.omega(), alpha=0.6)

    plt.show()

def animar(self, dt):
    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"""
            self.step(dt)

            line.set_data(*self.posicion())
            time_text.set_text('time = %.1f' % self.time_elapsed)
            energy_text.set_text('energy = %.3f J' % self.energia())
            return line, time_text, energy_text

        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)

        return ani
```

[36]:

```
%cat utils.py
```

```
import numpy as np

def normalizeRads(angle_in_rads, decimals=4):
    return np.around(np.arctan2(np.sin(angle_in_rads), np.cos(angle_in_rads)),
decimals=decimals)

def normalizeAngle(angle_in_degrees, decimals=4):
    angle_in_rads = angle_in_degrees * np.pi/180.
    normalized_rads = normalizeRads(angle_in_rads, decimals)
    return normalized_rads*180/np.pi
```

[39]:

```
%cat video_tag.py
```

```
from tempfile import NamedTemporaryFile
from IPython.display import HTML
import matplotlib.pyplot as plt

VIDEO_TAG = """<video controls>
```

```
 <source src="data:video/x-m4v;base64,{0}" type="video/mp4">
 Tu navegador no soporta este formato de video.
</video>"""

def anim_to_html(anim):
    if not hasattr(anim, '_encoded_video'):
        with NamedTemporaryFile(suffix='.mp4') as f:
            anim.save(f.name, writer='avconv', fps=20, extra_args=['-vcodec',
'libx264'])
            video = open(f.name, "rb").read()
        anim._encoded_video = video.encode("base64")

    return VIDEO_TAG.format(anim._encoded_video)


def display_animation(anim):
    plt.close(anim._fig)
    return HTML(anim_to_html(anim))
```