Actividad 10: Animación del péndulo y su espacio fase

Valenzuela Chaparro Hugo de Jesús

8 de mayo de 2016

1. Periodo péndulo

Sabemos, anteriormente, que para encontrar el periodo del péndulo para oscilaciones arbitrarias debemos evaluar la integral:

$T = 4\sqrt{\frac{\ell}{2g}} \int_0^{\theta_0} \frac{1}{\sqrt{\cos\theta - \cos\theta_0}} d\theta$

Y con la aproximación de ángulos pequeños se reducía a:

$T_0 = \frac{2\pi}{\omega} = 2\pi\sqrt{\frac{\ell}{g}}$

En este caso haremos una animación de un péndulo con cuerda de 1 m de longitud, moviéndose a distintos ángulos iniciales, sin amortiguamiento; además veremos como avanza en el tiempo en el espacio fase.

2. Códigos y gráficas

2.1. Código

El código que se utilizó fue el siguiente, lo único que se hizo para cada caso fue cambiar el ángulo inicial θ_0:

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

class DoublePendulum:
    def __init__(self,
                 init_state = [0, 0, 0, 0],
                 L1 = 1.0,  # Longitud del pendulo 1
                 L2 = 0.0,  # Longitud del pendulo 2
```

```
            M1 = 1.0,  # Masa del pendulo 1
            M2 = 1.0,  # Masa del pendulo 2
            G = 9.8,   # Aceleracion por la gravedad
            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):
    (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 * 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):
    (M1, M2, L1, L2, G) = self.params

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

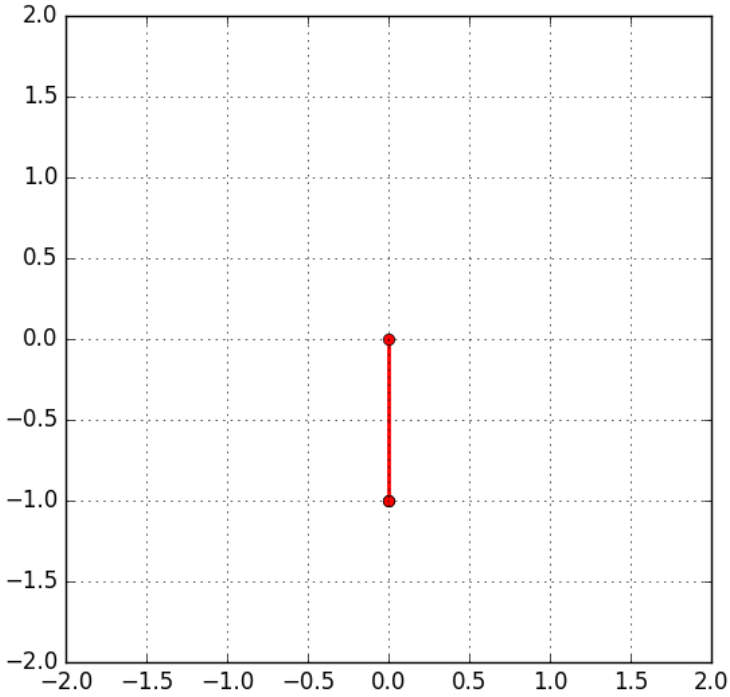
```
        cos_delta = cos(state[2] - state[0])
        sin_delta = sin(state[2] - state[0])

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

#Condiciones iniciales pendulo
theta0= 45 #angulo de donde se suelta el pendulo
omega0= 0 #velocidad angular inicial
pendulum = DoublePendulum([theta0, omega0, 0.0, 0.0])
g = 9.81
l = 1 #longitud del pendulo
c = g/l

X_f1 =np.array([(theta0/180.0)*np.pi,(omega0/180.0)*np.pi])
t = np.linspace(0,30,500)

b=0
#Ecuacion pendulo sin friccion
def pend(y, t, b, c):
    theta, omega = y
    dy_dt = [omega,-c*np.sin(theta)]
    return dy_dt

#Trayectoria
y0 = X_f1
X = odeint(pend, y0, t, args=(b,c))


dt = 1./60.
fig = plt.figure()
```

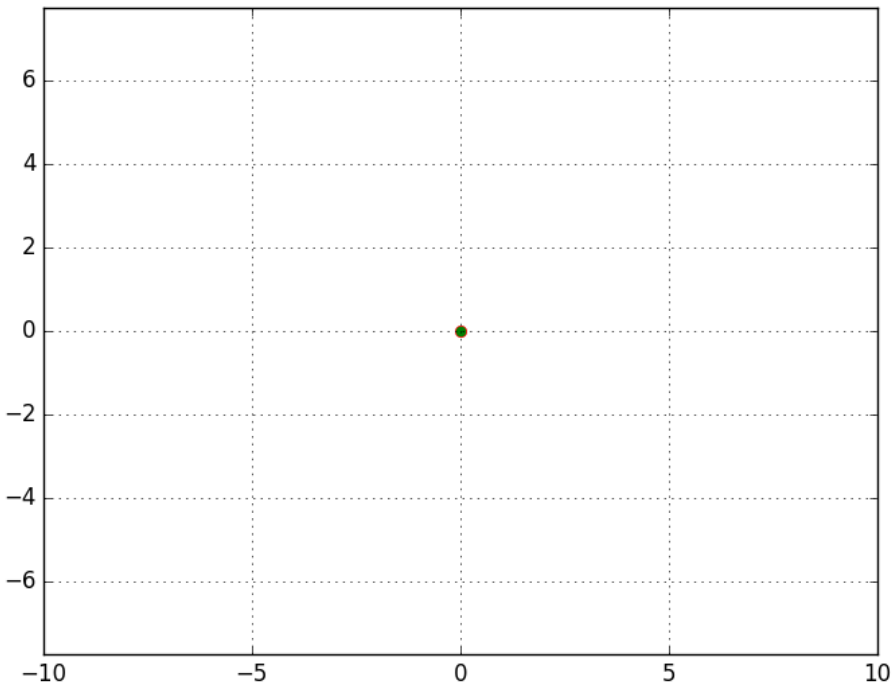
```
ax = fig.add_subplot(111, aspect='equal', autoscale_on=False,
                     xlim=(-2, 2), ylim=(-2, 2))
ax.grid()

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

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
    pendulum.step(dt)
    line.set_data(*pendulum.position())
    return line, time_text, energy_text

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

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


plt.show()

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='y')
        self.line1a = Line2D([], [], color='g', linewidth=2)
        self.line1e = Line2D(
            [], [], color='g', 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.grid()
        ax1.set_aspect('equal', 'datalim')

        an.TimedAnimation.__init__(self, fig, interval=50, blit=True)

    def _draw_frame(self, framedata):
        i = framedata
        head = i - 1
        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])

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

    def _init_draw(self):
        lines = [self.line1, self.line1a, self.line1e]
        for l in lines:
            l.set_data([], [])


ani = SubplotAnimation()
plt.show()
```

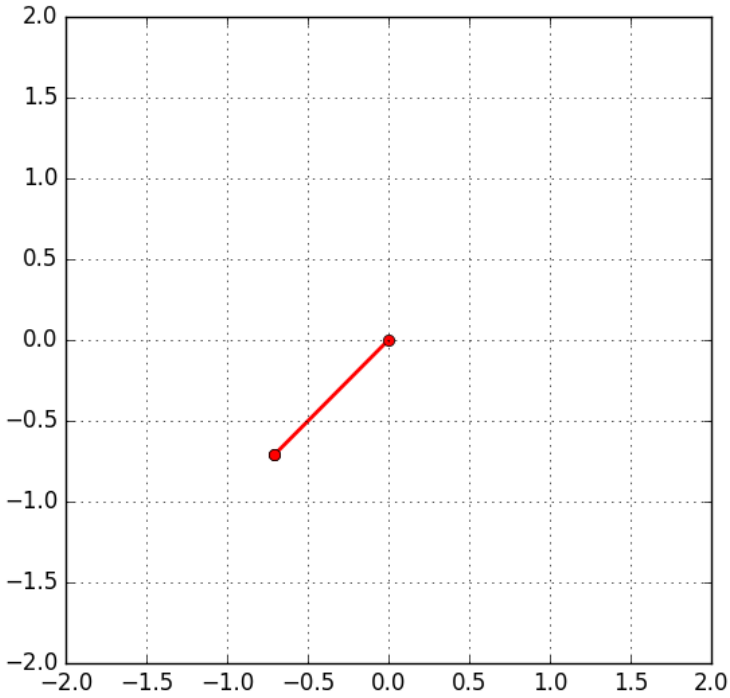
2.2. $\theta_0 = 0$

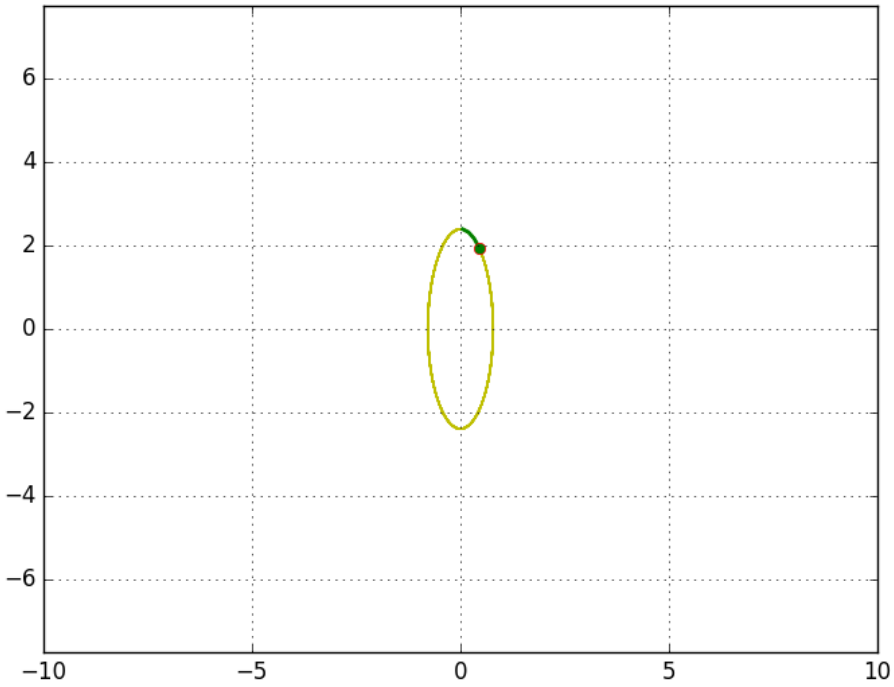
Posición de equilibrio


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

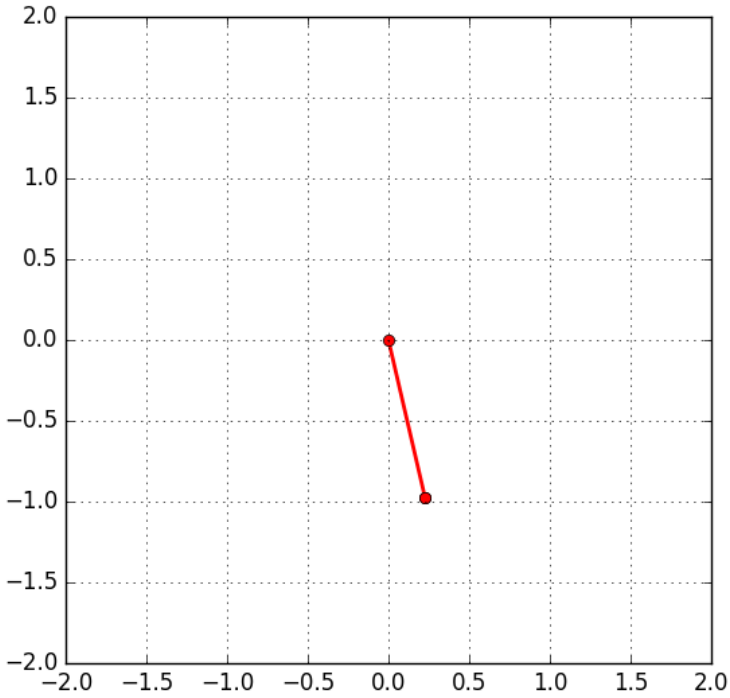

6
4
2
0
−2
−4
−6
−10
−5
0
5
10

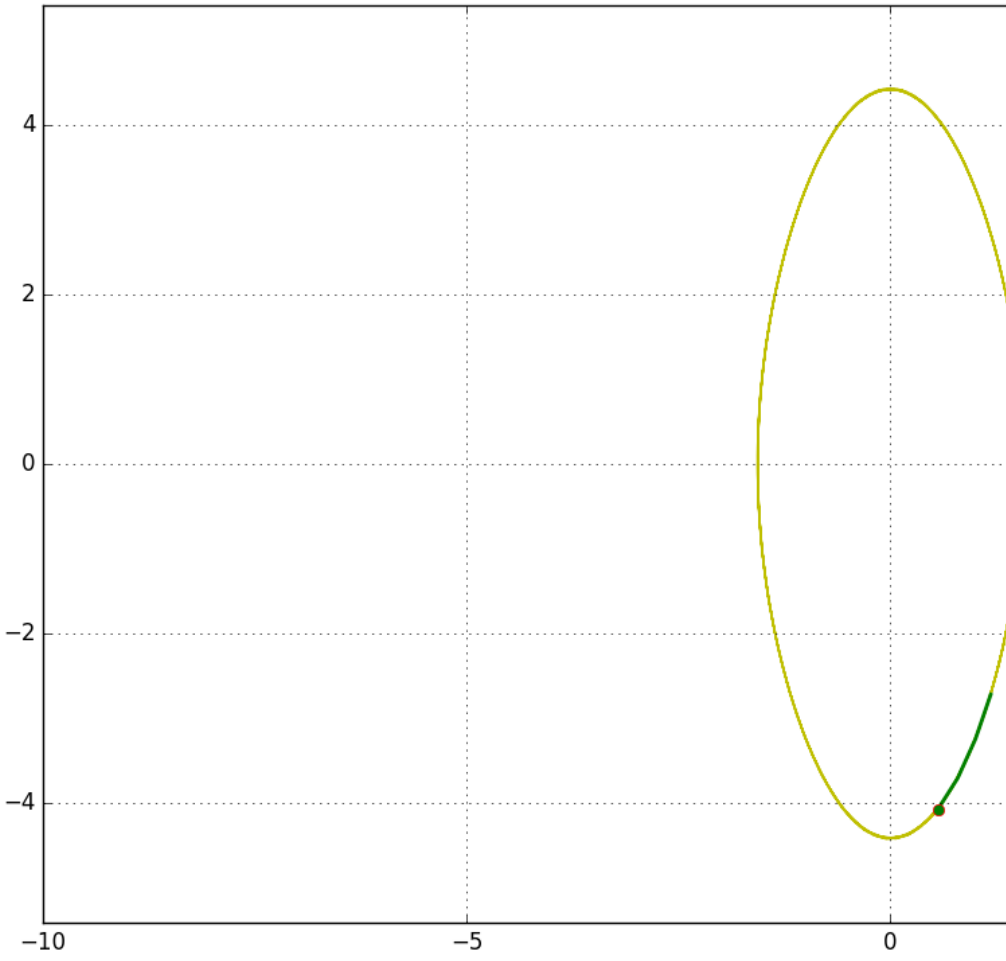
2.3. $\theta_0 = 45°$



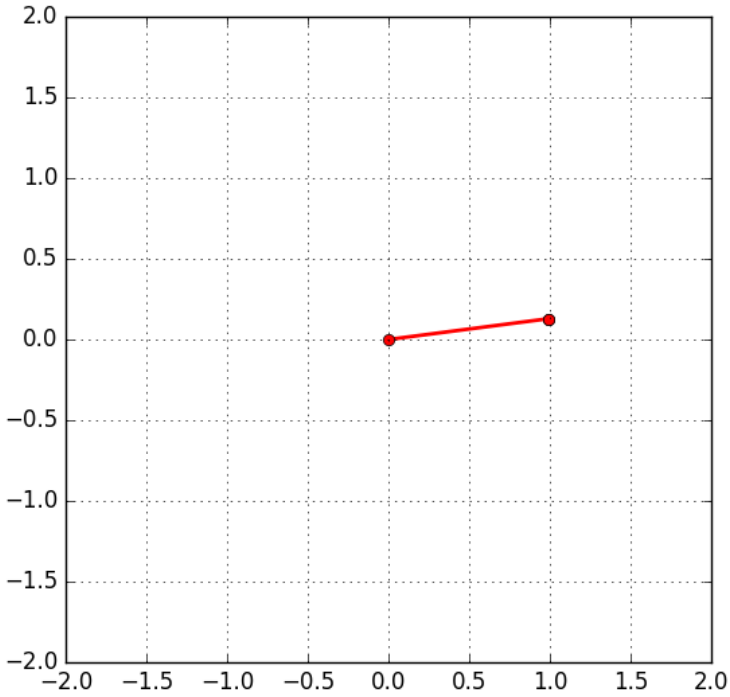

6
4
2
0
−2
−4
−6
−10
−5
0
5
10

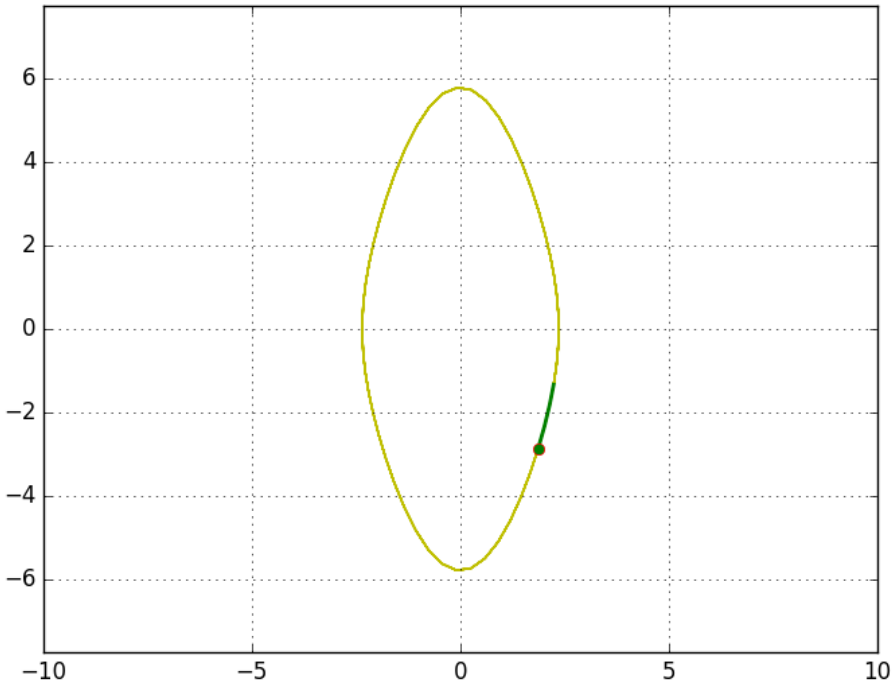
2.4. $\theta_0 = 90°$



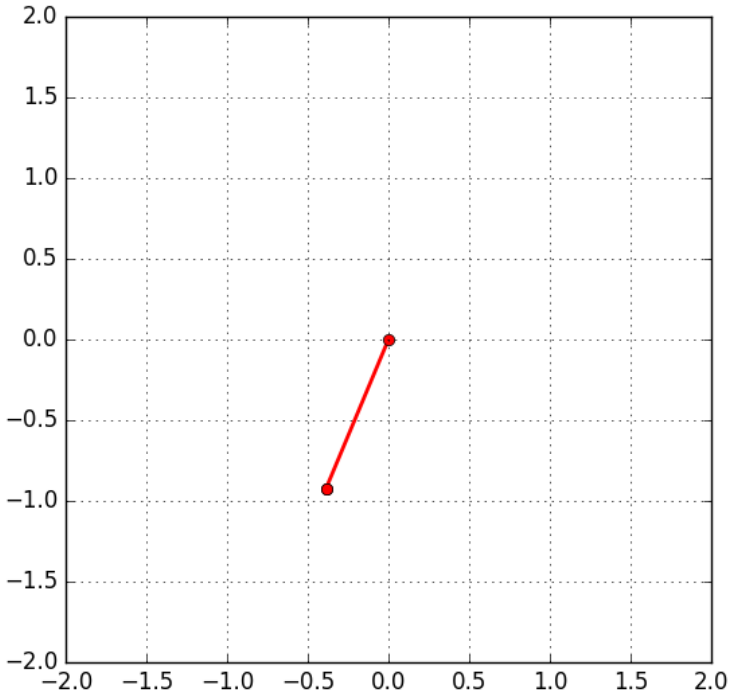

4
2
0
−2
−4
−10
−5
0

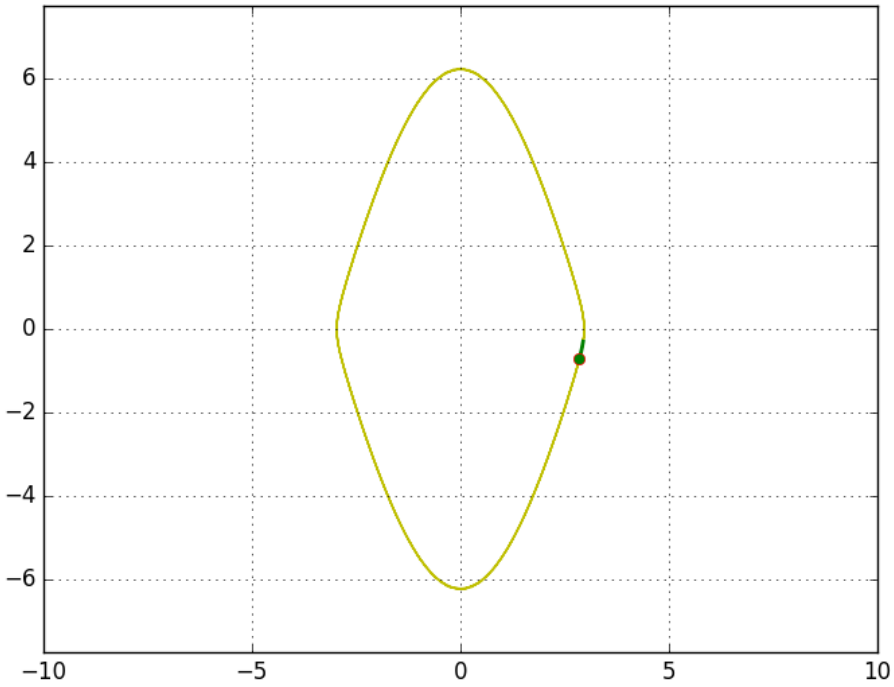
3. $\theta_0 = 135^\circ$



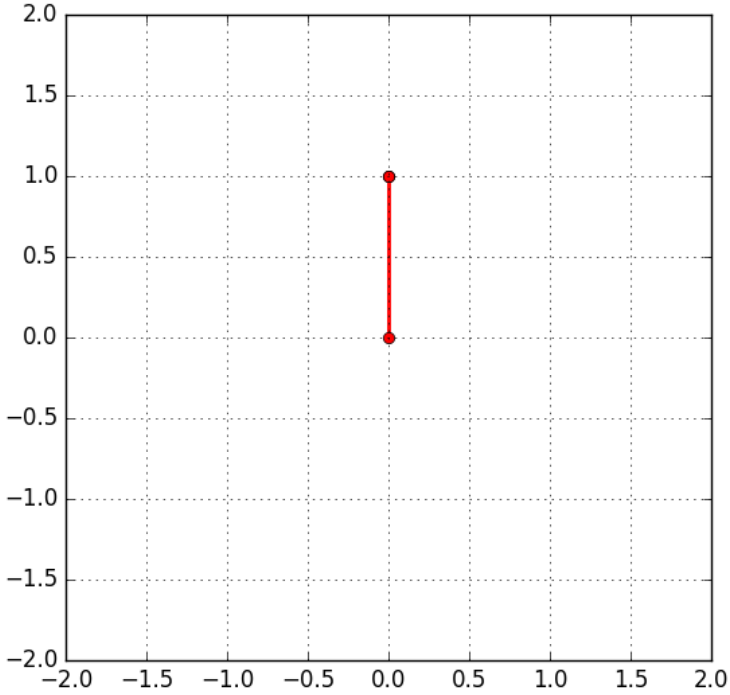

6
4
2
0
−2
−4
−6
−10
−5
0
5
10

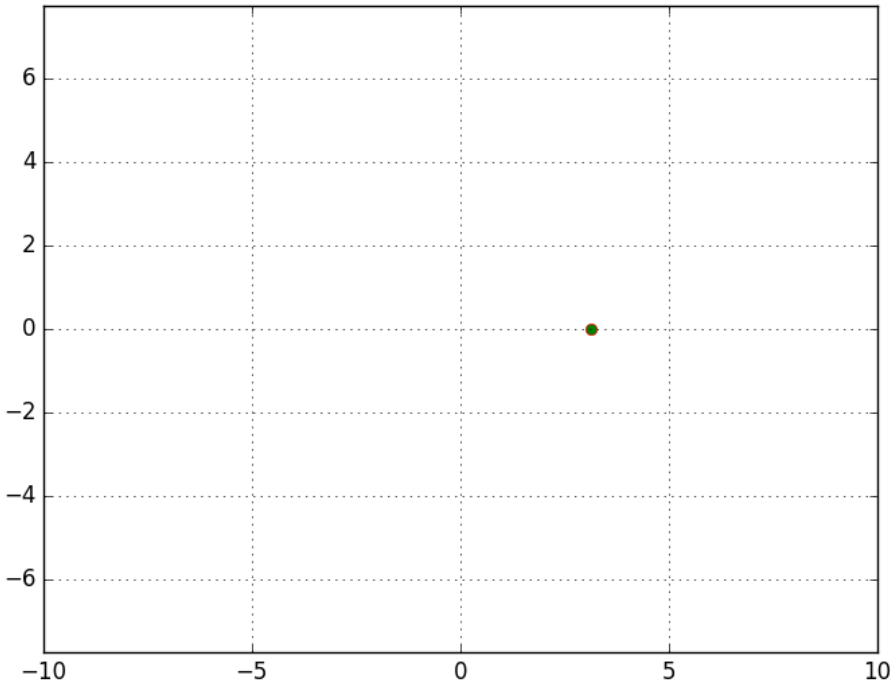
4. $\theta_0 = 170^\circ$



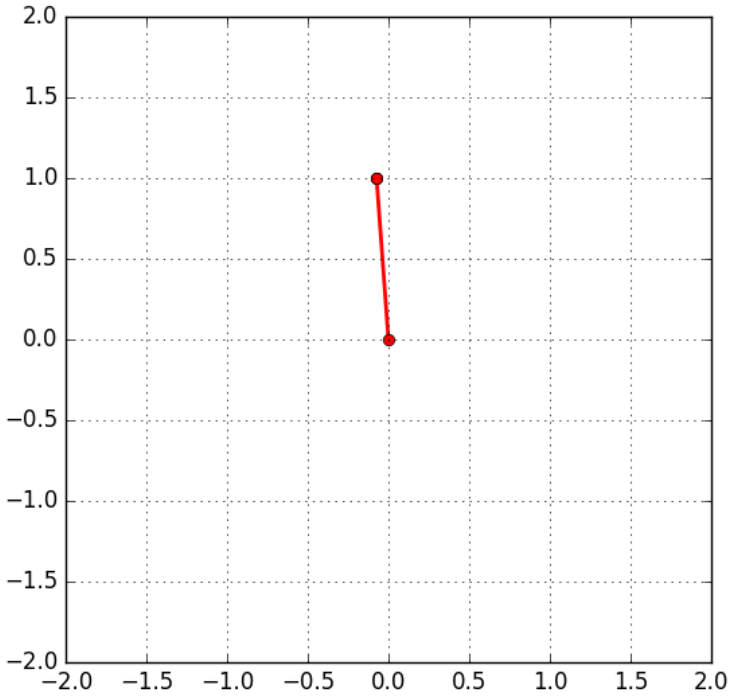

6
4
2
0
−2
−4
−6
−10
−5
0
5
10

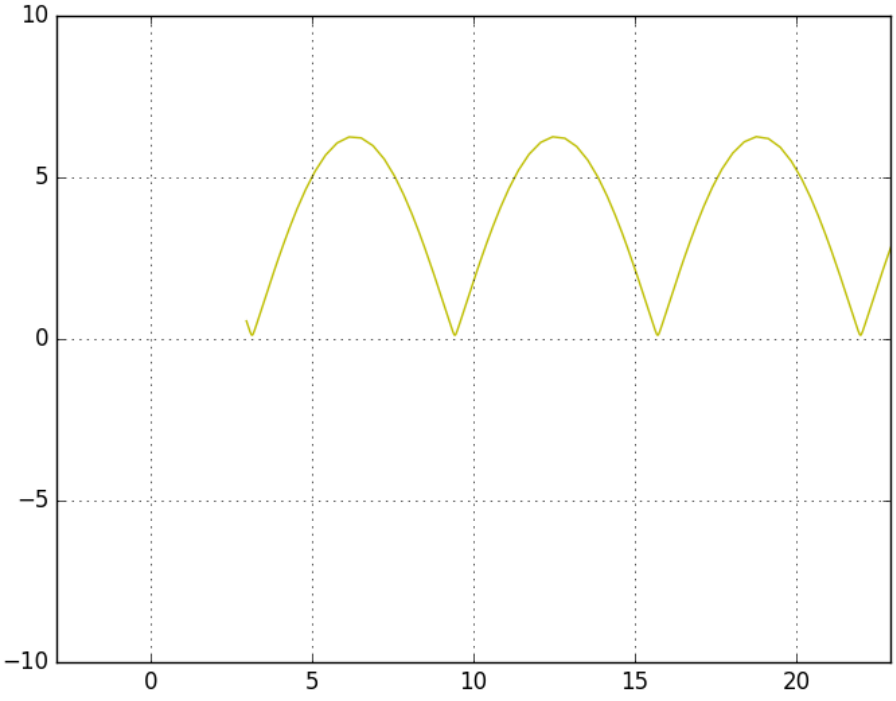
5. $\theta_0 = 180^\circ$



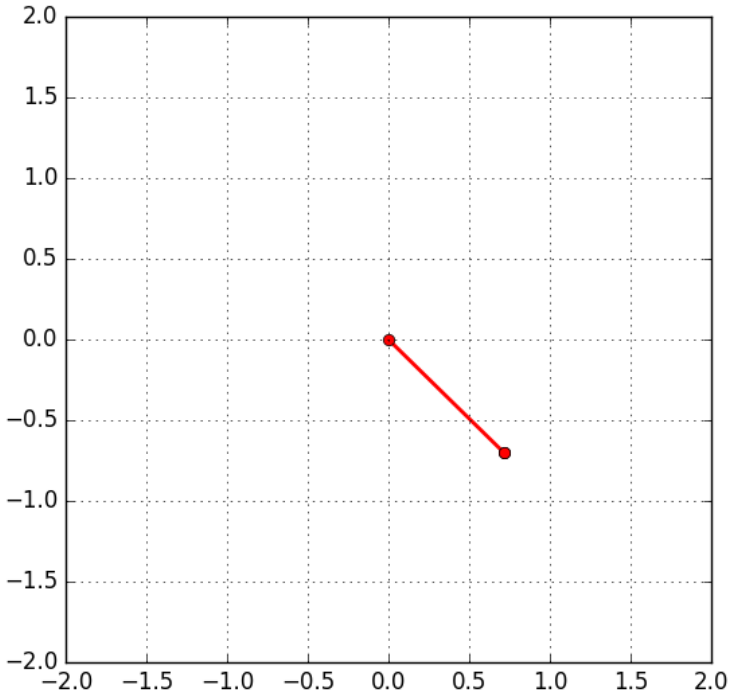

6
4
2
0
−2
−4
−6
−10
−5
0
5
10

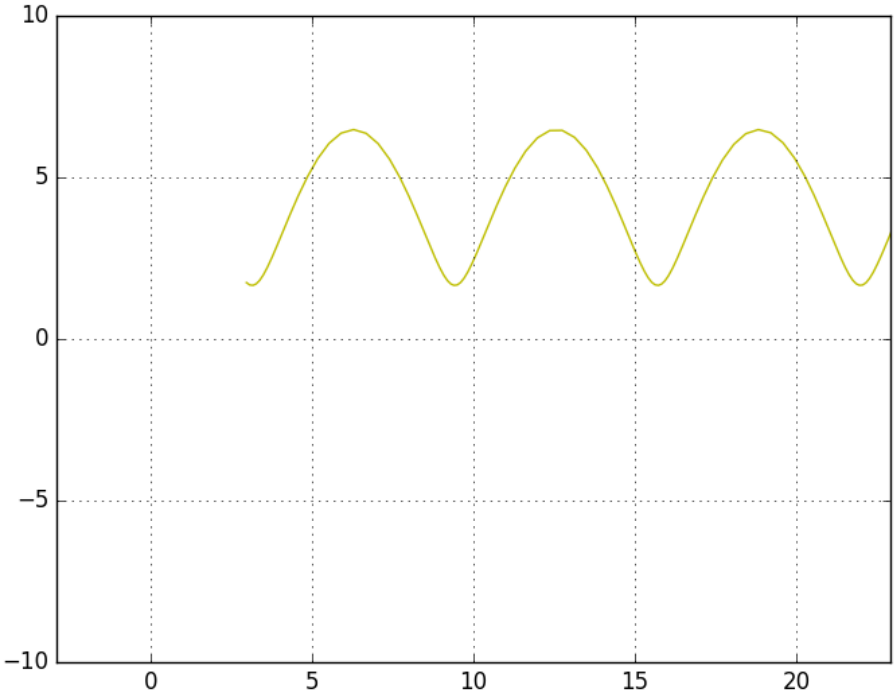
6. $\theta_0 = 170°$ **y la mínima energía para una oscilación completa (con** $\omega_0 = 32\frac{grados}{s}$**)**




10
5
0
−5
−10
0
5
10
15
20

7. $\theta_0 = 170°$ **y suficiente energía para una oscilación completa (con** $\omega_0 = 100\frac{grados}{s}$**)**




10
5
0
−5
−10
0
5
10
15
20