UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO
FACULTAD DE INGENIERÍA MECANICÁ ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS
(Año del Bicentenario, de la consolidación de nuestra Independencia, y de la conmemoración de la heroica batalla de Junín y Ayacucho)

TITULO: CODIGO PYTHON PENDULO SIMPLE

NOMBRE: FABRICIO MAYTA GUZMÁN

NUMERO DE ORDEN: 22

CURSO: FISICA I

DOCENTE: Dr. CARCAUSTO QUISPE CARLOS

SEMESTRE: II **GRUPO:** “A”

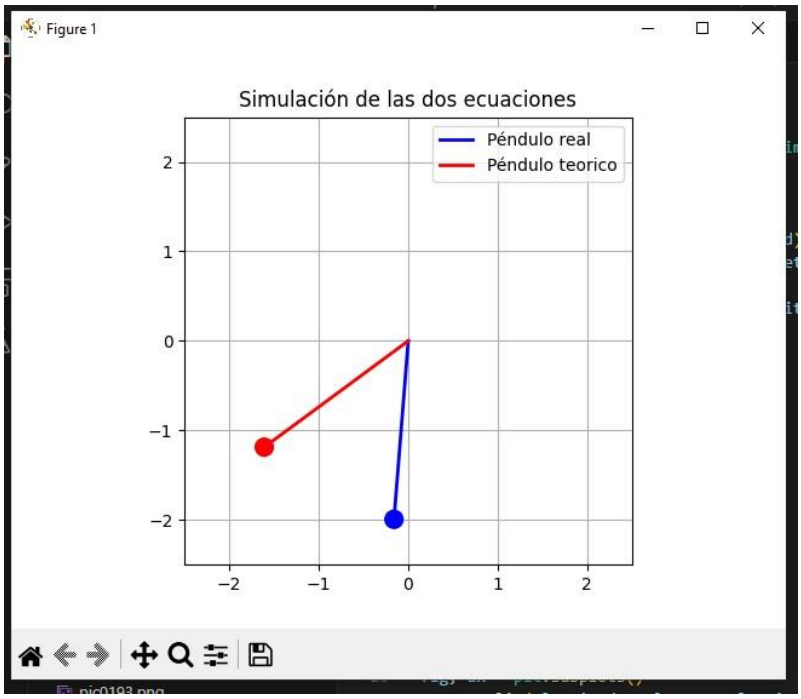
FECHA: 24/ 09/2024

PUNO – PERÚ

2024

Código en Python (Visual Studio Code)

Código de la simulación (1)

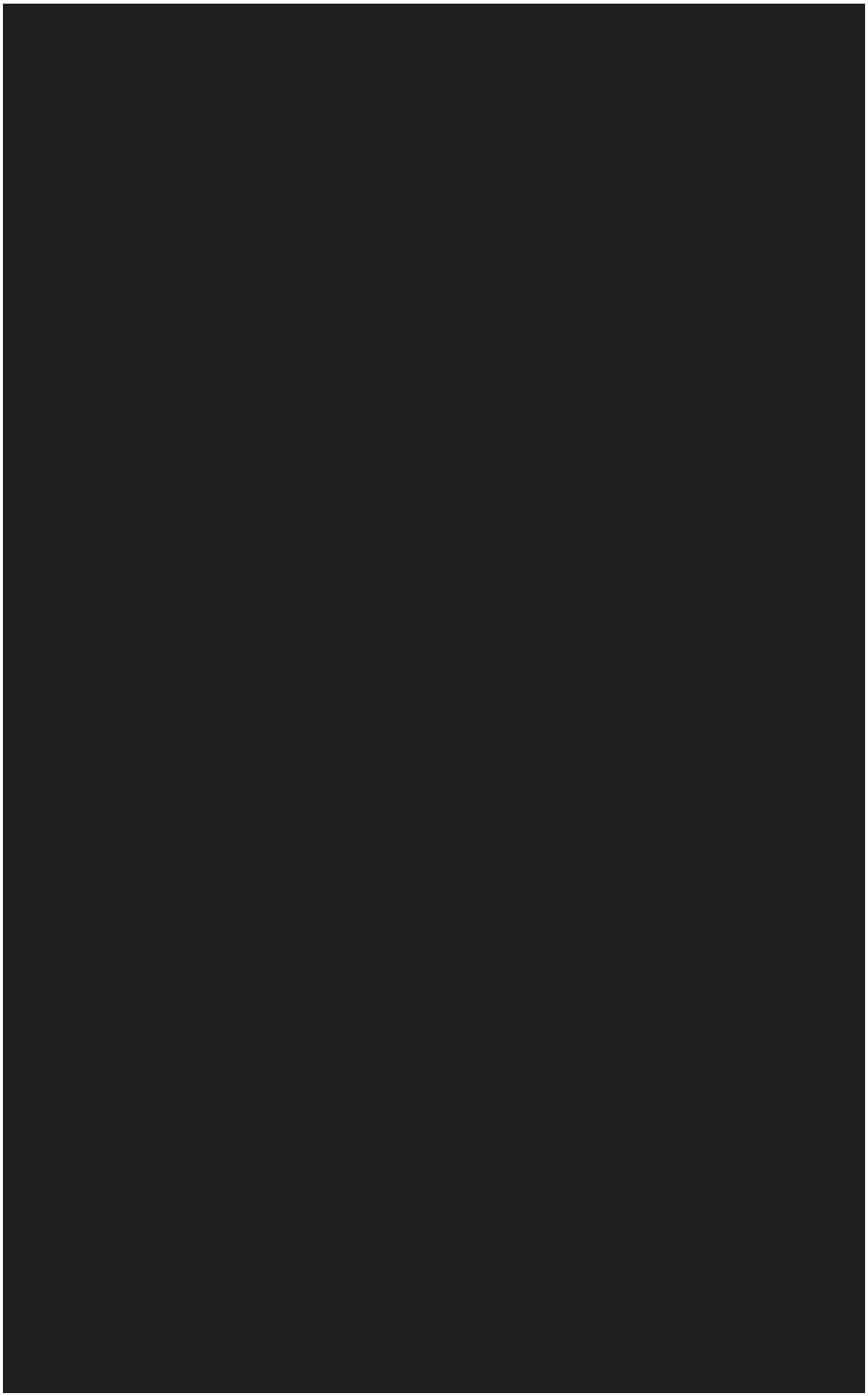


```
import numpy as np import matplotlib.pyplot as
plt from matplotlib.animation import
FuncAnimation

g = 9.81

def MoviientoReal(theta, omega, longitud):
return - (g / longitud) * np.sin(theta)
 def MoviientoTeorico(theta, omega,
longitud):
    return - (g / longitud) * theta
 theta0_Real =
np.radians(70)  omega0_Real
= 0.0   longitud_Real = 2.0
 theta0_Teorico =
np.radians(70)
omega0_Teorico = 0.0
longitud_Teorico = 2.0
 t_max = 10   fps = 60  t =
np.linspace(0, t_max, t_max * fps)
#grafico fig, ax = plt.subplots() ax.set_xlim(-
longitud_Real - 0.5, longitud_Real + 0.5)
ax.set_ylim(-longitud_Real - 0.5, longitud_Real + 0.5)
ax.set_aspect('equal')
 cuerda_Real, = ax.plot([], [], lw=2, label='Péndulo real', color='b')
cuerda_Teorico, = ax.plot([], [], lw=2, label='Péndulo teorico', color='r')

bola_radio = 0.1   bola_Real = plt.Circle((0, 0),
bola_radio, color='b') bola_Teorico = plt.Circle((0, 0),
bola_radio, color='r')
 ax.add_patch(bola_Real)
ax.add_patch(bola_Teorico)
 def
init():
    cuerda_Real.set_data([], [])
cuerda_Teorico.set_data([], [])     bola_Real.center = (0, -
longitud_Real)       bola_Teorico.center = (0, -
longitud_Teorico)       return cuerda_Real, cuerda_Teorico,
bola_Real, bola_Teorico
 def runge_kutta(theta, omega, longitud, dt,
ecuacion):
    k1_theta = omega     k1_omega =
ecuacion(theta, omega, longitud)
```



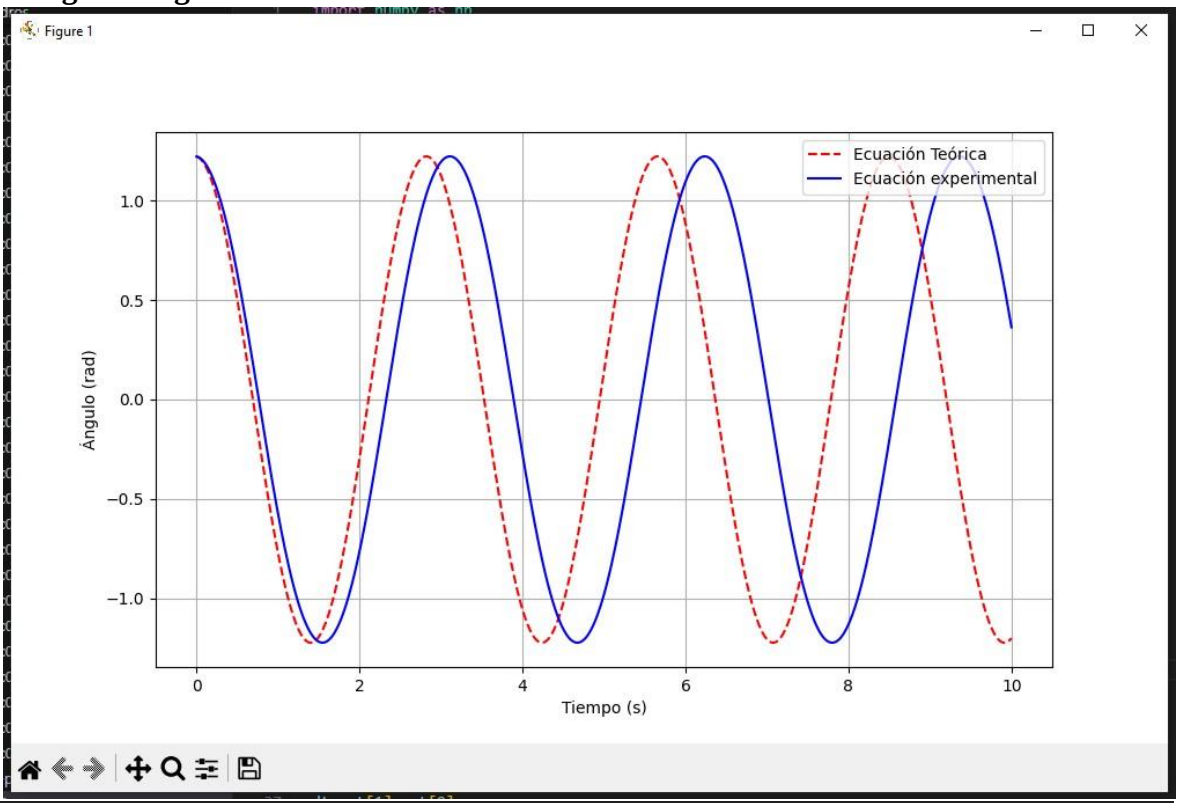
```
    k2_theta = omega + 0.5 * dt * k1_omega     k2_omega =
ecuacion(theta + 0.5 * dt * k1_theta, k2_theta, longitud)
    k3_theta = omega + 0.5 * dt * k2_omega     k3_omega =
ecuacion(theta + 0.5 * dt * k2_theta, k3_theta, longitud)
    k4_theta = omega + dt * k3_omega     k4_omega =
ecuacion(theta + dt * k3_theta, k4_theta, longitud)
    theta_next = theta + (dt / 6) * (k1_theta + 2 * k2_theta + 2 *
k3_theta
+ k4_theta)     omega_next = omega + (dt / 6) * (k1_omega + 2 * k2_omega +
2 * k3_omega
+ k4_omega)          return
theta_next, omega_next

# actualización  def
update(frame):
    global theta0_Real, omega0_Real, theta0_Teorico, omega0_Teorico
        dt = t[1]
- t[0]
    theta0_Real, omega0_Real = runge_kutta(theta0_Real,
omega0_Real, longitud_Real, dt, MoviientoReal)     theta0_Teorico,
omega0_Teorico = runge_kutta(theta0_Teorico, omega0_Teorico,
longitud_Teorico, dt, MoviientoTeorico)
        x_Real = longitud_Real *
np.sin(theta0_Real)     y_Real = -longitud_Real
* np.cos(theta0_Real)

    x_Teorico = longitud_Teorico * np.sin(theta0_Teorico)
y_Teorico = -longitud_Teorico * np.cos(theta0_Teorico)
        cuerda_Real.set_data([0, x_Real], [0,
y_Real])     bola_Real.center = (x_Real, y_Real)
        cuerda_Teorico.set_data([0, x_Teorico], [0,
y_Teorico])     bola_Teorico.center = (x_Teorico,
y_Teorico)

    return cuerda_Real, cuerda_Teorico, bola_Real, bola_Teorico
 ani = FuncAnimation(fig, update, frames=len(t), init_func=init,
blit=True, interval=1000/fps)
 plt.title('Simulación de las dos
ecuaciones') plt.legend() plt.grid()
plt.show()
```

Código de la grafica




```
import numpy as np import
matplotlib.pyplot as plt
 g = 9.81
longitud = 2.0
 def movimiento_real(theta, omega,
longitud):     return - (g / longitud) *
np.sin(theta)
 def movimiento_teorico(theta, omega,
longitud):
    return - (g / longitud) * theta
 def runge_kutta(theta, omega, funcion_movimiento, longitud,
dt):
    k1_theta = omega
    k1_omega = funcion_movimiento(theta, omega, longitud)
     k2_theta = omega + 0.5 * dt *
k1_omega
    k2_omega  =  funcion_movimiento(theta  +  0.5  *  dt  *  k1_theta,  k2_theta,
longitud)

    k3_theta = omega + 0.5 * dt * k2_omega
    k3_omega  =  funcion_movimiento(theta  +  0.5  *  dt  *  k2_theta,  k3_theta,
longitud)

    k4_theta = omega + dt * k3_omega     k4_omega = funcion_movimiento(theta
+ dt * k3_theta, k4_theta, longitud)
```

```
    theta_siguiente = theta + (dt / 6) * (k1_theta + 2 * k2_theta + 2
* k3_theta + k4_theta)    omega_siguiente = omega + (dt / 6) * (k1_omega
+ 2 * k2_omega + 2 * k3_omega + k4_omega)
        return theta_siguiente,
omega_siguiente
 theta0 =
np.radians(70)
omega0 = 0.0
 t_max = 10   fps = 60   t =
np.linspace(0, t_max, t_max * fps)
dt = t[1] - t[0]
 theta_teorico = [] theta0_t, omega0_t = theta0, omega0 for _ in t:
theta0_t, omega0_t = runge_kutta(theta0_t, omega0_t, movimiento_teorico,
longitud, dt)     theta_teorico.append(theta0_t)
 theta_real = [] theta0_r, omega0_r = theta0, omega0 for _ in t:
theta0_r, omega0_r = runge_kutta(theta0_r, omega0_r, movimiento_real,
longitud, dt)     theta_real.append(theta0_r)
 theta_teorico =
np.array(theta_teorico) theta_real =
np.array(theta_real)

plt.figure(figsize=(10, 6)) plt.plot(t, theta_teorico,
label='Ecuación Teórica', color='red', linestyle='--') plt.plot(t,
theta_real, label='Ecuación experimental', color='blue',
linestyle='-') plt.xlabel('Tiempo (s)') plt.ylabel('Ángulo (rad)')
 plt.legend(loc='upper right', bbox_to_anchor=(1,
1))  plt.grid(True)

plt.show()
```