Revista Brasileira de Ensino de Física
https://doi.org/10.1590/1806-9126-RBEF-2025-0233

Supplementary Material to "Small oscillations study of a nonlinear circular oscillator"

I. ARDUINO PROGRAM

Listing 1: Arduino program adapted from https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;
int pushButton = 7;
unsigned long int time0;
int cntrl = 0;

void setup(void) {
  Serial.begin(115200);
  // make the pushbutton's pin (Start/Stop) an input:
  pinMode(pushButton, INPUT);
  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
  // set accelerometer range to +-8G
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  // set gyro range to +- 500 deg/s
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  // set filter bandwidth to 21 Hz
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  delay(100);
}

void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
//Print values only if Start/Stop pushbotton is in state = 0.
  if(buttonState == 1){
    time0 = micros();
    if (cntrl == 0){
     Serial.print("Temperature in Celsius: ");
     Serial.println(temp.temperature);                // print temperature
     cntrl = 1;                                       // print only once
   }
  }
  else{
    Serial.print((float)(micros()-time0)/1000000,3);  // print time with 3 decimal places
    Serial.print(' ');                                // print a separation of data in the same line
    Serial.println(g.gyro.z,3);                       // print the z compoment of angular velocity
    }                                                 // and skip a line
  delay(1);
 }
```

II. PYTHON CODE TO FIT EXPERIMENTAL DATA

Listing 2: Python code to optimize the non-linear diferential equation solved with SciPy integrator to fit experimental data.

```python
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import differential_evolution
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

# experiental data file name (substitute it if you want)
file_ = 'exp_data.txt'  # or 'exp_data.csv'
deg_rad = np.pi/180  #convert degrees to radians
# load the experimental data to a NumPy array
# Assuming file has two columns (time and angular speed) separated by commas/space
exp_data = np.loadtxt(file_, delimiter=' ')  # To CSV use delimiter=','
t_exp = exp_data[:,0]
dphidt_exp = exp_data[:, 1]*deg_rad

# ODE (1st order)
def model(t, y, eta, omega0, epsilon):
    phi,  dphidt = y
    d2phidt2 = -eta * dphidt - omega0**2 * phi - epsilon * phi**2
    return [dphidt, d2phidt2]

# Numerical solution for a given set of parameters theta = (eta, omega0, epsilon)
def solve_ode(theta, t):
    phi0, eta, omega0, epsilon = theta
    cond_ini = [phi0, dphidt_exp[0]]  # initial condition (phi0, dphidt0=exp_data in t0)
    sol = solve_ivp(model, [t[0], t[-1]], cond_ini, args=(eta, omega0, epsilon), t_eval=t)
    # Getting the results
    phi = sol.y[0]  # Position phi(t)
    dphidt = sol.y[1]  # angular velocity dphi/dt
    return phi, dphidt  # return phi(t) and dphi/dt

# Residual function for optimization
def residual(theta):
    phi_pred,dphidt_pred = solve_ode(theta, t_exp)
    return (dphidt_pred - dphidt_exp)

#Calculate teh covariance using least_squares jacobian and the optmized parameters
def calculate_covariance(result, residual, bounds):
    res_lsq = least_squares(residual, result.x, bounds=(np.array(bounds)[:,0],
                            np.array(bounds)[:,1]))
    J = res_lsq.jac
    residuals = res_lsq.fun
    dof = len(residuals) - len(result.x)  # Graus de liberdade
    covariance = np.linalg.inv(J.T @ J) * (residuals**2).sum() / dof

    return covariance

#intervals for the parameters to be optimized
phi0_min, phi0_max = 0.0014, 0.04    #radians
eta_min, eta_max = 0.35, 0.75         # s^-1
omega0_min, omega0_max = 10, 20       # rad/s
eps_min, eps_max = -26550,-177        # s^-2

# Initial gess and fit
bounds = [(phi0_min, phi0_max),(eta_min, eta_max),(omega0_min, omega0_max),(eps_min, eps_max)]
result = differential_evolution(lambda params: np.sum(residual(params)**2), bounds=bounds)
# To calculate uncertainties of the optimized parameters
covariance = calculate_covariance(result, residual, bounds)
uncertainties = np.sqrt(np.diag(covariance))

# Printing the optimized parameters

print("\nOptimized parameters with uncertainties:")
print(f"phi0 = {result.x[0]:.6f} +/- {uncertainties[0]:.6f}")
print(f"eta = {result.x[1]:.6f} +/- {uncertainties[1]:.6f}")
```

```
print(f"omega0 = {result.x[2]:.6f}  {uncertainties[2]:.6f}")
print(f"epsilon = {result.x[3]:.6f} +/- {uncertainties[3]:.6f}")
print("Residual sum of squares : ",result.fun)

phi,dphidt =solve_ode(result.x, t_exp)

# print the optimized result values of solved ODE equation of angular speed in a file

with open('result.dat', 'w') as f:
    for i in range(len(t_exp)):
        print(t_exp[i],dphidt_exp[i],dphidt[i], file=f)

# Plotting results
fig = plt.figure(figsize=(12, 8))
gs = GridSpec(2, 2, figure=fig)
# Add space between subplots
fig.subplots_adjust(hspace=0.5)  # Adjust this value as needed

# Graphic 1: Angular velocity vs Time
ax1 = fig.add_subplot(gs[0, 0])
ax1.plot(t_exp, phi, 'b',
linewidth=2, label=f'$\\eta = {result.x[1]:.3f}\pm{uncertainties[1]:.3f}$' r' ${\rm s^{-1}}$, ' '\n'
f'$\overline{{\\epsilon}} = {result.x[3]:.0f}\pm{uncertainties[3]:.0f}$' r' ${\rm s^{-2}}$')
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Position $\\varphi(t)$')
ax1.set_title('Non-linear dumped oscillator')
ax1.grid(True)
ax1.legend()

# Graphic 2: phase space (phi vs v)
ax2 = fig.add_subplot(gs[0, 1])
ax2.plot(phi, dphidt, 'r', linewidth=1)
ax2.set_xlabel('Position $\\varphi$')
ax2.set_ylabel('Angular Velocity $\\dot{\\varphi}$')
ax2.set_title('Phase Space $(\\varphi, \\dot{\\varphi})$')
ax2.grid(True)

# Grphic 3: Non-linearity behaviour
ax3 = fig.add_subplot(gs[1, 0])
ax3.plot(t_exp, result.x[3] * (phi)**2, 'g', linewidth=2, \
label='Non-linear term $\\epsilon \\varphi^2$')
ax3.set_xlabel('Time (s)')
ax3.set_ylabel('$\\epsilon \\varphi^2$')
ax3.set_title('Non-Linear term contribution to ODE')
ax3.grid(True)
ax3.legend()
# Grphic 4: phi term contribution
ax4 = fig.add_subplot(gs[1, 1])
ax4.plot(t_exp, result.x[2]**2 * phi, 'g', linewidth=2, \
label='linear term $\\omega_0^2 \\varphi$')
ax4.set_xlabel('Time (s)')
ax4.set_ylabel('$\\omega_0 \\varphi$')
ax4.set_title('Linear term contribution to ODE')
ax4.grid(True)
ax4.legend()
# To save only the plot area without extra whitespace, use bbox_inches='tight':
plt.savefig("numerical.png", bbox_inches='tight', dpi=300)
plt.tight_layout()
plt.show()
```