FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
NATIONAL RESEARCH UNIVERSITY ITMO
ITMO UNIVERSITY

FACULTY OF CONTROL SYSTEMS AND ROBOTICS

PROGRAM:
ROBOTICS AND ARTIFICIAL INTELLIGENCE

LABORATORY REPORT
SIMULATION OF ROBOTIC SYSTEMS
PRACTICAL WORK №2

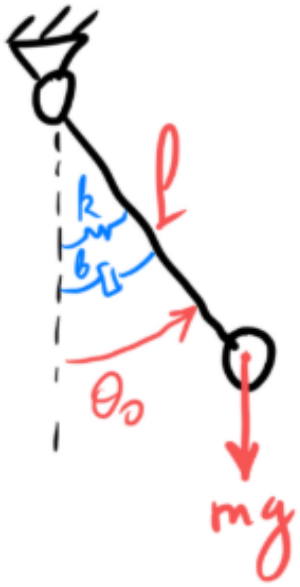
STUDENT
Zongo Sabane

PROFESSOR
Rakshin Egor Aleksandrovich

Academic Year: 2025

STATEMENT

1. Compose the ODE from the mass spring damper system



m, kg	k, Nm/rad	b, Nm*s/rad	l, m	theta 0, rad
1	5.8	0.02	0.6	-1.319509896

2. Try to analytically solve the ODE you composed. If you can't solve it, describe the reason and why?

3. Compare results of analytical solutions (if it exists) with numerical solutions, discuss and conclude.

SOLVING

1. Ordinary Differential Equation (ODE) of the System

The system described is a mass spring damper system, with the following parameters:

Mass of the body: m = 1 kg
Rod length: l = 0.6 m
spring stiffness: k = 5.8 Nm/rad
Damper viscosity: b = 0.02 Nm*s/rad
Initial angle: theta 0 = −1.319509896 rad.

lagrangian analysis: L =T - V

- Kinetic Energy T

$$T = \frac{1}{2} m \cdot v^2 \Leftrightarrow T = \frac{1}{2} m \cdot l^2 \cdot \dot{\theta}^2, \quad v = l \cdot \dot{\theta}$$

- Potential energy V

$$V = V_g + V_K$$

  - potential energy of the body Vg

$$V_g = mgl(1 - \cos\theta)$$

  - Potential energy of the spring Vk

$$V_k = \frac{1}{2} k \cdot \theta^2$$

Total potential energy: V=Vg+Vk

$$V = mgl(1 - \cos\theta) + \frac{1}{2} k \cdot \theta^2$$

$$\text{lagrangian: } L = T - V = \frac{1}{2} m \cdot l^2 \cdot \dot{\theta}^2 - \left[mgl(1 - \cos\theta) + \frac{1}{2} k \cdot \theta^2 \right]$$

Apply Euler–Lagrange equation (ELE):

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial L}{\partial \dot{\theta}}\right)-\frac{\partial L}{\partial \theta}=Q$$

where Q is an external force acting on the system, which is the damping force in this case

$$\frac{\partial \mathrm{L}}{\partial \dot{\theta}}=m\cdot l^2\dot{\theta}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right)=m\cdot l^2\ddot{\theta}$$

$$\frac{\partial L}{\partial \theta}=-mgl\,\sin\theta-k\theta$$

$$Q=-b\dot{\theta}$$

$substitute$:

$$m\cdot l^2\ddot{\theta}+\mathrm{mgl}\,\sin\theta+\mathrm{k}\theta=-\mathrm{b}\dot{\theta}\;\Leftrightarrow\; m\cdot l^2\ddot{\theta}+\mathrm{b}\dot{\theta}+\mathrm{mgl}\,\sin\theta+\mathrm{k}\theta=0$$

$$\Leftrightarrow\ddot{\theta}+\frac{\mathrm{b}}{m\cdot l^2}\dot{\theta}+\frac{\mathrm{g}}{\mathrm{l}}\sin\theta+\frac{\mathrm{k}}{m\cdot l^2}\theta=0\ \ (\mathrm{ODE})$$

Insert numerical values:

$$\frac{\mathrm{b}}{\mathrm{m}\cdot \mathrm{l}^2}=\frac{0.02}{0.36}=0.0555=C_1$$

$$\frac{g}{l}=\frac{9.8}{0.6}=16.3333=C_2$$

$$\frac{k}{m\cdot l^2}=\frac{5.8}{0.36}=16.1111=C_3$$

$$\ddot{\theta}+\mathrm{C}_1\dot{\theta}+\mathrm{C}_2\sin\theta+\mathrm{C}_3\theta=0$$

$$\ddot{\theta}+0.0555\dot{\theta}+16.3333\sin\theta+16.1111\theta=0\ \text{(EDO)}$$

2. This ODE can’t be solved analitycally

The presence of the nonlinear term sinθ makes the differential equation nonlinear. In general, nonlinear second-order ODEs do not admit closed-form analytical solutions. Only under the small angle approximation (sinθ≈θ) does the equation become linear and solvable analytically. However, the given initial condition theta_0= -1.319509896 rad is too large for this approximation to be valid. Therefore, no exact analytical solution exists for this problem in its full form.

3. Numerical solutions of the ODE

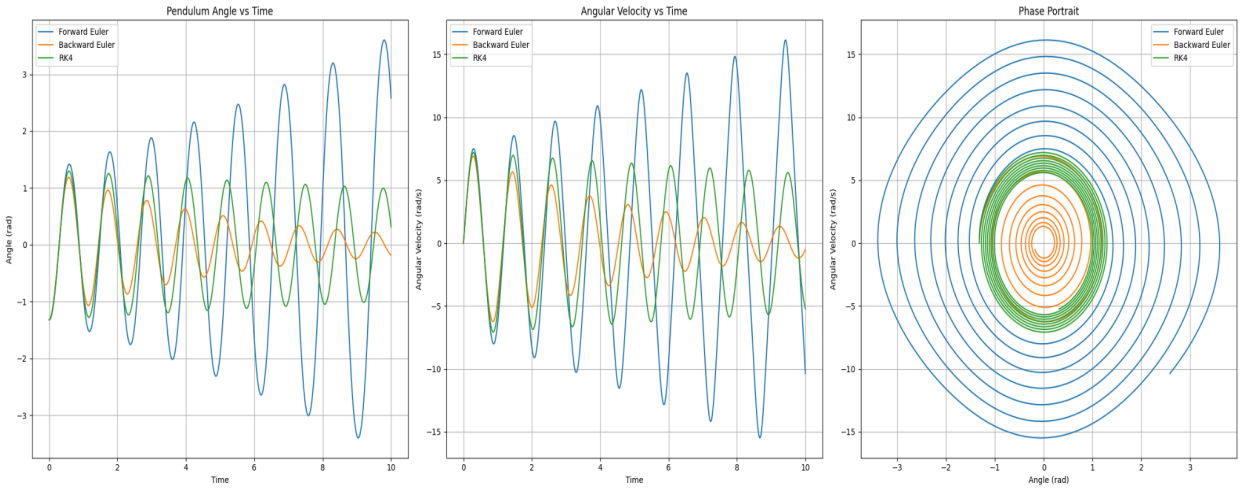
Python script and plot:

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

def pendulum_dynamics(x):

    # System Parameters
    m = 1.0
    l = 0.6
    k = 5.8
    b = 0.02
    g = 9.8
    c1 = b / (m * l**2)
    c2 = g / l
    c3 = k / (m * l**2)

    theta, thetadot = x
    dthetadot = -c1 * thetadot - c2 * np.sin(theta) - c3 * theta

    return np.array([thetadot, dthetadot])

# Initial conditions
theta0 = -1.319509896
thetadot0 = 0.0
x0 = [theta0, thetadot0]

def forward_euler(fun, x0, Tf, h):
    """
    Explicit Euler integration method
    """
```

```
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        x_hist[:, k + 1] = x_hist[:, k] + h * fun(x_hist[:, k])

    return x_hist, t

def backward_euler(fun, x0, Tf, h, tol=1e-8, max_iter=100):
    """
    Implicit Euler integration method using fixed-point iteration
    """
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        x_hist[:, k + 1] = x_hist[:, k]  # Initial guess

        for i in range(max_iter):
            x_next = x_hist[:, k] + h * fun(x_hist[:, k + 1])
            error = np.linalg.norm(x_next - x_hist[:, k + 1])
            x_hist[:, k + 1] = x_next

            if error < tol:
                break

    return x_hist, t

def runge_kutta4(fun, x0, Tf, h):
    """
    4th order Runge-Kutta integration method
    """
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        k1 = fun(x_hist[:, k])
        k2 = fun(x_hist[:, k] + 0.5 * h * k1)
        k3 = fun(x_hist[:, k] + 0.5 * h * k2)
        k4 = fun(x_hist[:, k] + h * k3)
```

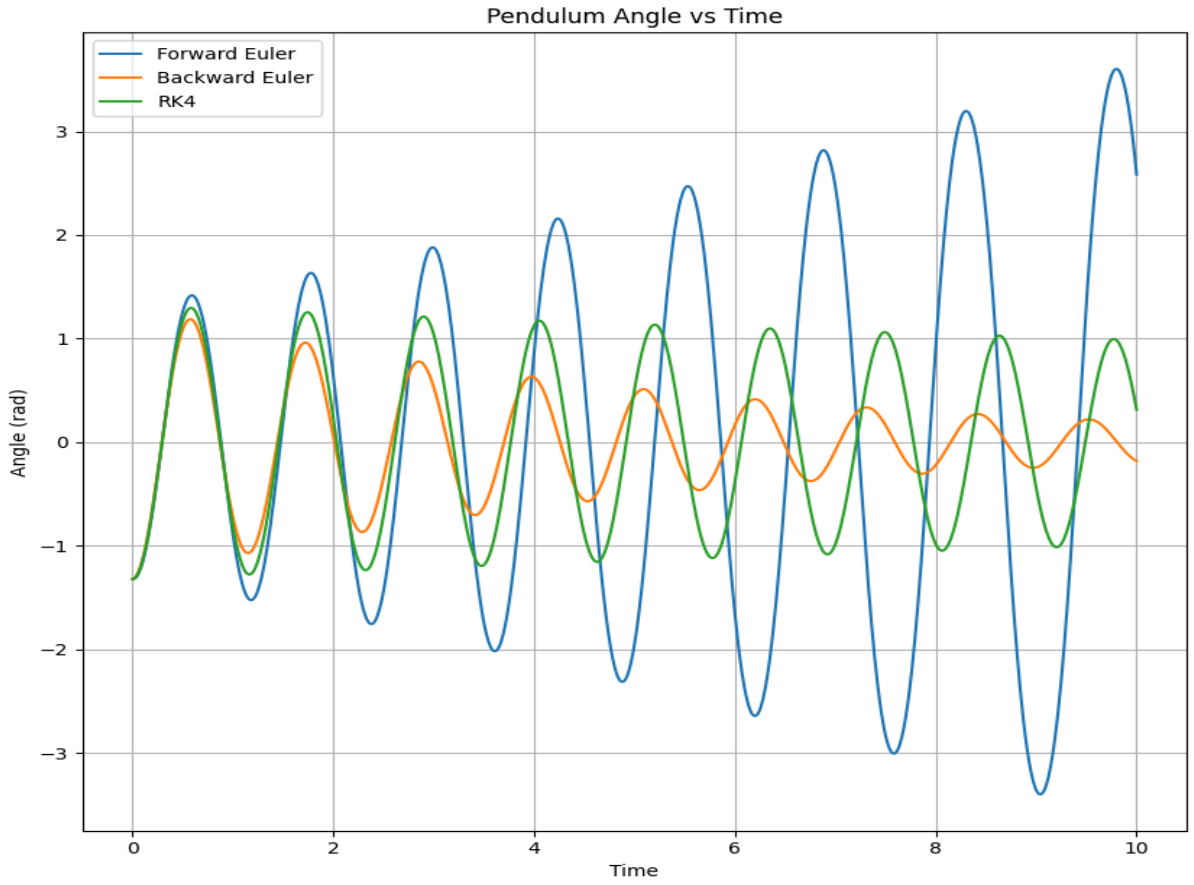
```
        x_hist[:, k + 1] = x_hist[:, k] + (h / 6.0) * (k1 + 2*k2 + 2*k3
+ k4)

    return x_hist, t

# Test all integrators
Tf = 10.0
h = 0.01

# Forward Euler
x_fe, t_fe = forward_euler(pendulum_dynamics, x0, Tf, h)

# Backward Euler
x_be, t_be = backward_euler(pendulum_dynamics, x0, Tf, h)

# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(pendulum_dynamics, x0, Tf, h)

# Plot results
plt.figure(figsize=(24, 8))

plt.subplot(1, 3, 1)
plt.plot(t_fe, x_fe[0, :], label='Forward Euler')
plt.plot(t_be, x_be[0, :], label='Backward Euler')
plt.plot(t_rk4, x_rk4[0, :], label='RK4')
plt.xlabel('Time')
plt.ylabel('Angle (rad)')
plt.legend()
plt.title('Pendulum Angle vs Time')
plt.grid(True)

plt.subplot(1, 3, 2)
plt.plot(t_fe, x_fe[1, :], label='Forward Euler')
plt.plot(t_be, x_be[1, :], label='Backward Euler')
plt.plot(t_rk4, x_rk4[1, :], label='RK4')
plt.xlabel('Time')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Angular Velocity vs Time')
plt.grid(True)

plt.subplot(1, 3, 3)
```

```
plt.plot(x_fe[0, :], x_fe[1, :], label='Forward Euler')
plt.plot(x_be[0, :], x_be[1, :], label='Backward Euler')
plt.plot(x_rk4[0, :], x_rk4[1, :], label='RK4')
plt.xlabel('Angle (rad)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Phase Portrait')
plt.grid(True)

plt.tight_layout()
plt.savefig('509109_Zongo_Saban_task2_plot.png', dpi=150)
plt.show()
```



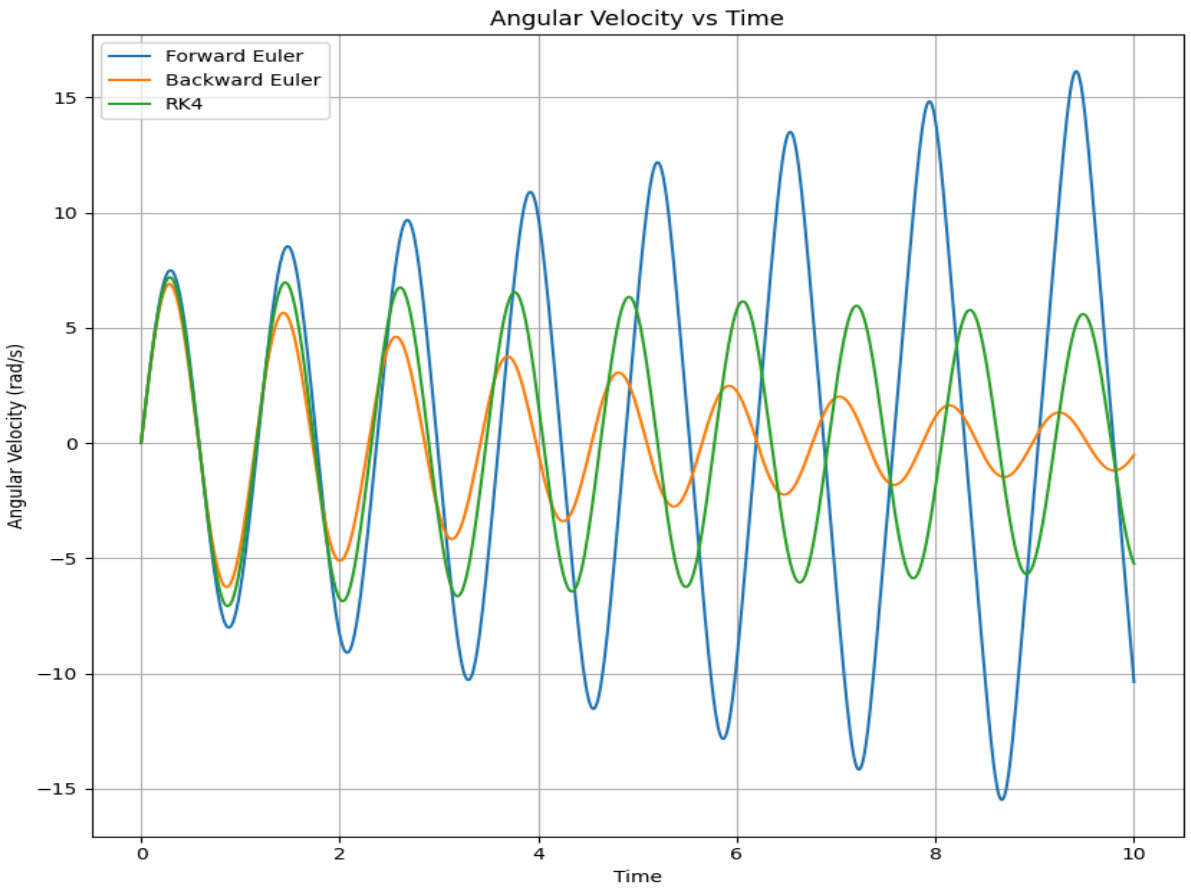
4. Observations

Angle vs time:



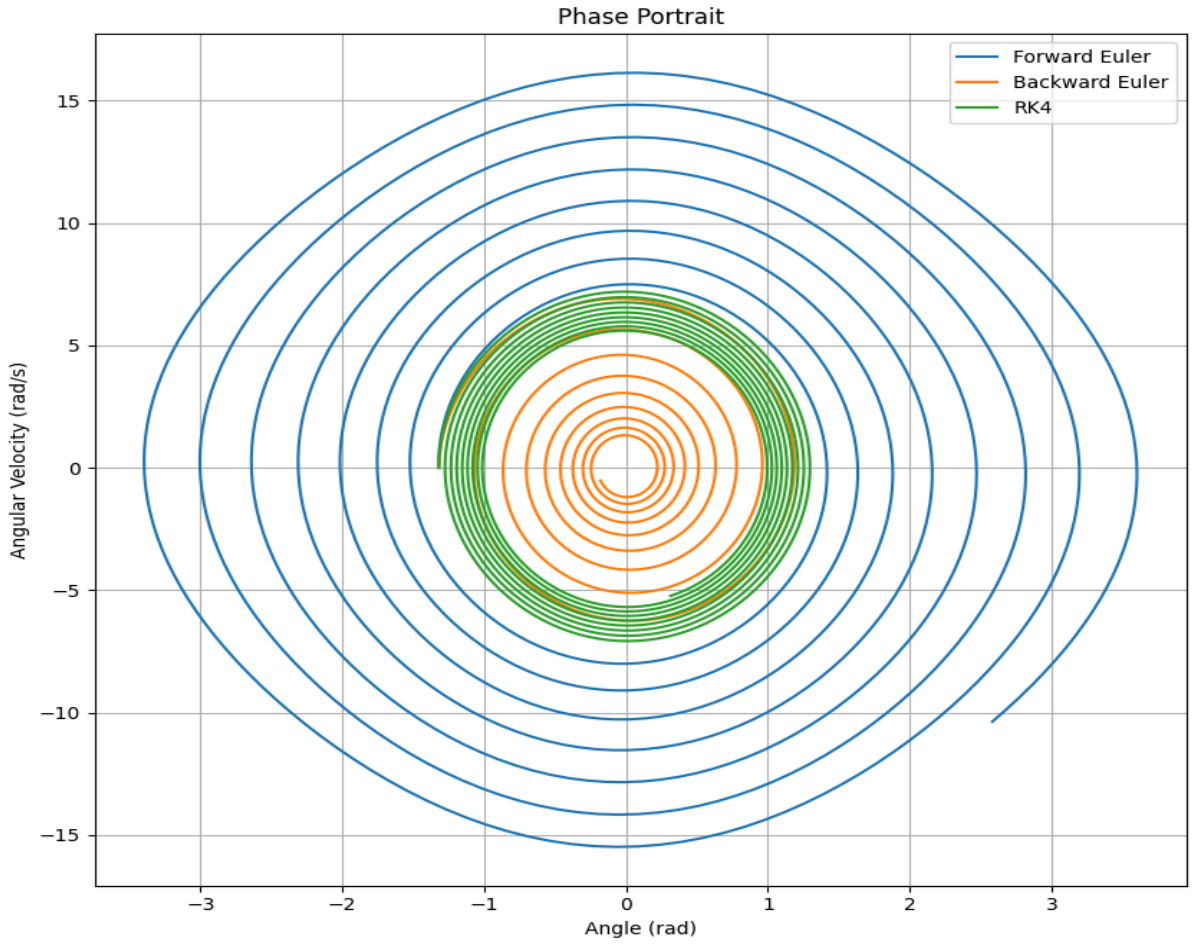
The three methods are qualitatively consistent, but:
Forward Euler shows slight instability/ error accumulation (amplitude does not decay properly).
Backward Euler is over-damped (numerically dissipative behavior).
RK4 is the most accurate and stable, with consistent physical damping.

Velocity vs time:



Same observations: **RK4** faithfully captures the dynamics, while **Forward Euler** overestimates the energy, **and Backward** Euler underestimates it.

Phase portrait:



RK4 traces a smooth spiral towards the origin (expected behavior of a nonlinear damped oscillator).
Forward Euler tends to spiral outwards (non-physical energy gain >> instability).
B**ackward Euler** converges too quickly towards the origin (excessive energy loss).

5. Conclusion

The system is nonlinear due to the sin(θ) term, so no analytical solution exists for this case because theta_0 is too large.

RK4 is clearly superior for this nonlinear system.
Forward Euler is simple but unstable even for small time steps.
Backward Euler is unconditionally stable but too dissipative for oscillating systems.