THE MINISTRY OF SCIENCE AND HIGHER EDUCATION
OF THE RUSSIAN FEDERATION

ITMO University
(ITMO)

Faculty of Control Systems and Robotics

# Report of 2nd lab
for the subject
“Simulation of Robotic Systems”

Student:	
Group R4136c	*Хюинь Тан Куонг*
Tutor:	
Инженер	*Ракшин Е.А.*

Saint Petersburg 2025

CONTENTS

1 PROBLEM

In this work, it is proposed to formulate an ODE based on the given schematic diagram and verify its solution using three methods: the explicit Euler method, the implicit Euler method, and the Runge–Kutta method. Additionally, the resulting graphs are to be plotted and compared with the analytical solution of the system.

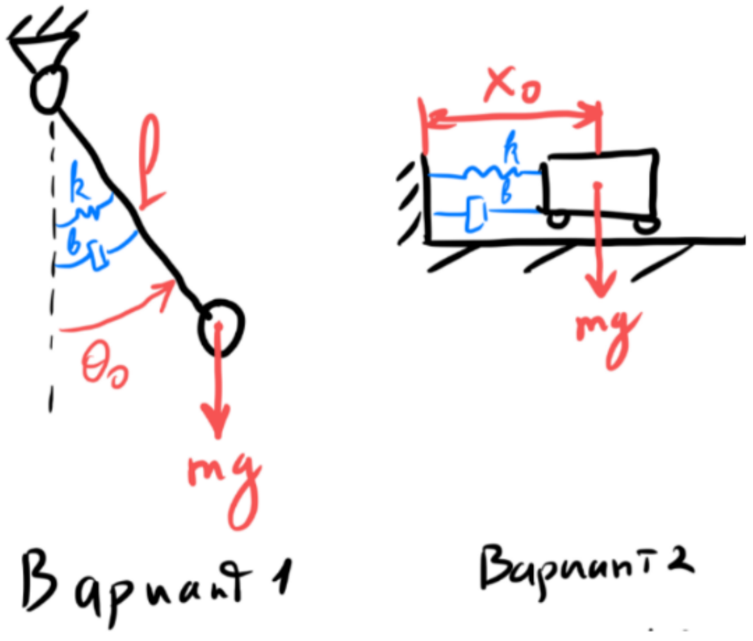

Figure 1 — Two schema from the task

My variant for the problem is shown below:

Variant	m, kg	k, N·m/rad	b, N·m·s/rad	l, m	θ_0, rad
1	1	17.2	0.005	0.96	-1.555136664

Table 1 — System parameters

2 SOLUTION

To write the equations of motion, we begin by deriving the Lagrangian for the system, L, as the difference between the kinetic and potential energy of the system

$$L = K - P$$

where K is the kinetic energy and P is the potential energy of the system.

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

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

The Lagrange's equations of motion:

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

where Q is the damping force in our case $Q = -b\dot{\theta}$

Let's compute these parts of the equation:
The partial derivative of L in respect to θ

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

The partial derivative of L in respect to $\dot{\theta}$

$$\frac{\partial L}{\partial \dot{\theta}} = ml^2\dot{\theta}$$

the full time derivative

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

Finally, the equations of motion for a pendulum-spring-damper system

$$ml^2\ddot{\theta} + mgl\sin\theta + k\theta = -b\dot{\theta}$$

or it can rewritten in the next form

$$\ddot{\theta} = -\frac{g}{l}\sin\theta - \frac{k}{ml^2}\theta - \frac{b}{ml^2}\dot{\theta}$$

This differential equation is nonlinear due to the part $\sin\theta$. The equation can be linearized ($\sin\theta \approx \theta$) only if the initial angle of the pendulum is small $\theta_0 < 5\,\text{deg}$. However, in my case, the initial angle $\theta_0 \approx -90\,\text{deg}$, so we can not do it. The analytical solution is unavailable, but we can solve it using numerical methods.

2.1 Simulation result

The simulation results show the behavior of the system in 10 second using 3 integrator methods: Forward Euler, Backward Euler and Runge-Kutta (RK4). As we can observe:

- When the time step is small (see 2), 3 methods is stable
- When the time step is medium (see 3), the Forward Euler tend to increase the amplitude of the motion. The Backward Euler decrease the amplitude of the motion. But the RK4 is still stable.
- When the time step is big (see 4), the Forward Euler become unstable.

2.1.1 Case study with small time step (dt=0.001)

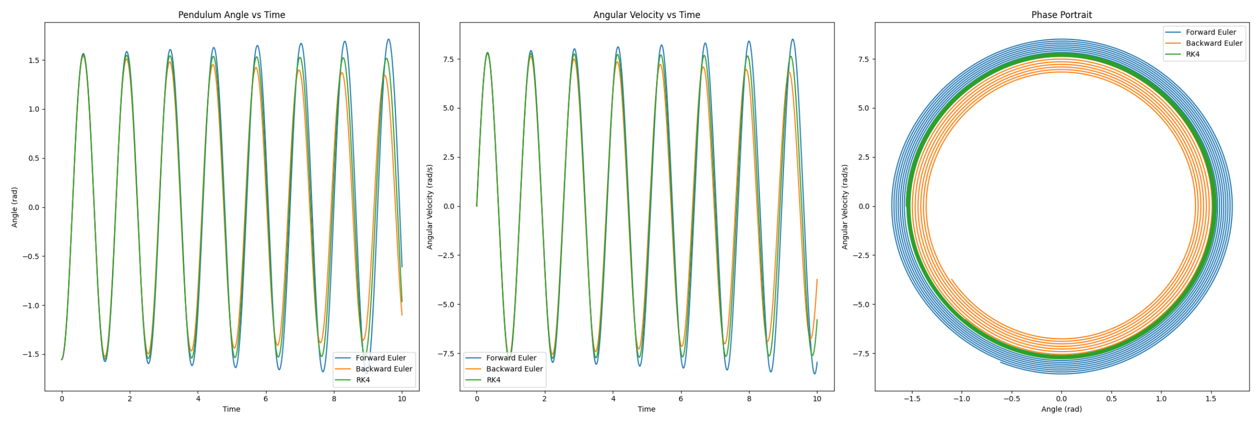

Figure 2 — Simulation result with small time step

2.1.2 Case study with medium time step (dt=0.01)

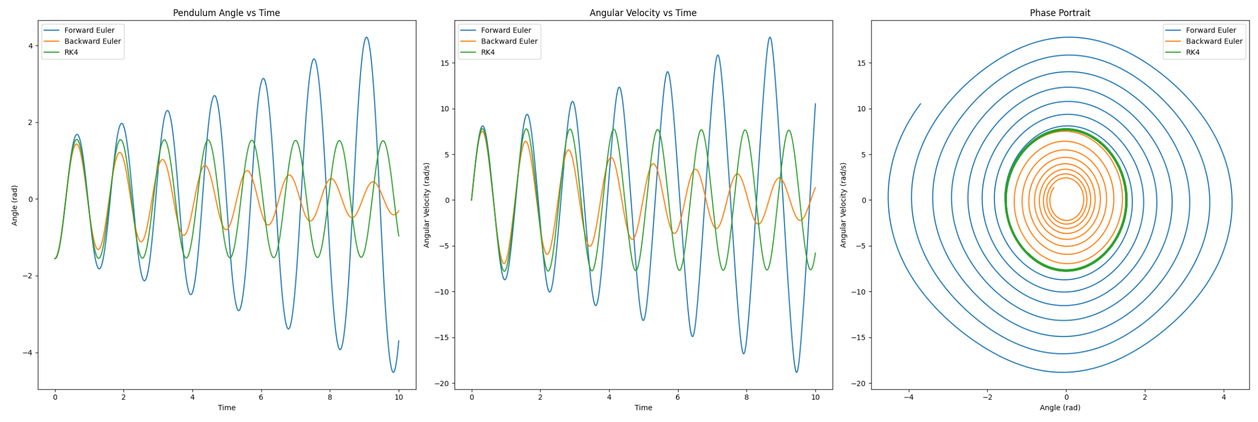

Figure 3 — Simulation result with medium time step

2.1.3 Case study with large time step (dt=0.1)

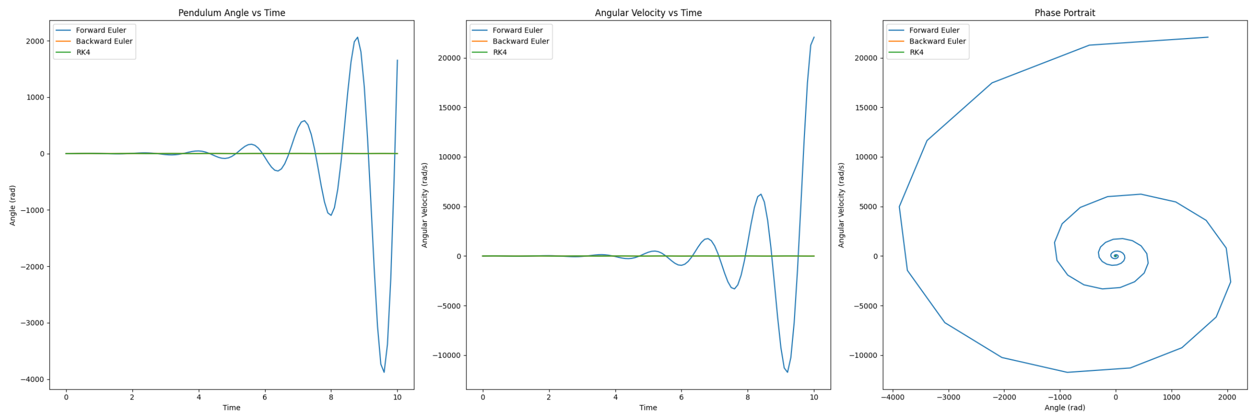

Figure 4 — Simulation result with large time step

3 CONCLUSION

In this lab, we have derive the equation of motion for a pendulum-spring-damping system using Lagrangian method. Then we conduct the simulation using 3 different integrator methods with different time step. The result shows that method RK4 is the most stable and accurate.

4 APPENDIX

Here is the code for the simulation:

```python
# %%
import numpy as np
import matplotlib.pyplot as plt


def pendulum_dynamics(x):
    """
    Pendulum dynamics:  d²/dt² = -(g/l) * sin () - k/(ml^2) * - b/(ml^2)* d/dt
    State vector x =  [,  d/dt]
    """
    l = 1.0
    g = 9.81
    k = 17.2 # spring constant
    m = 1.0 # mass
    b = 0.005 # damping coefficient

    theta = x[0]
    theta_dot = x[1]

    theta_ddot = -(g / l) * np.sin(theta) - k / (m * l**2) * theta - b / (m * l**2) * theta_dot

    return np.array([theta_dot, theta_ddot])


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

```

```python

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


def run_sim(Tf=10.0, h=0.01):
    # Test all integrators
    x0 = np.array([-1.555136664, 0.0]) # Initial state: [angle, angular_velocity]
    # 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.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.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.tight_layout()
    plt.show()


# %%
run_sim(10.0, 0.1)

# %%
run_sim(10.0, 0.01)

# %%
run_sim(10.0, 0.001)

# %%
run_sim(10.0, 0.0001)
```