THE MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION

ITMO University

Report for the task 2 "Variant 1"

for the subject

SIMULATION OF ROBOTIC SYSYTEMS

Name: Afif Hazem

ID_Number: 503259

Saint Petersburg 10/11/2025

Composing the ODE for mass spring damper simple pendulum

Let us assume that we have a single rigid mass m attached to massless rod of length l. Let $\theta(t)$ be the angular displacement measured from the downward vertical. There is a torsional spring at the pivot with stiffness k producing a restoring torque $-k\theta$. There is also a rotational viscous damper at the pivot with damping coefficient b producing a torque proportional to $-b\dot{\theta}$. The initial angle is θ_0.

Kinetic and potential energy

The kinetic energy is given by the following relation:

$$T = \frac{1}{2}\, m \left(l\dot{\theta}^2\right) = \frac{1}{2}\, m\, l^2\, \dot{\theta}^2$$

The potential energy has two parts: gravitational potential and spring potential.

Choosing zero of gravitational potential at the lowest position:

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

And spring potential:

$$V_s = \frac{1}{2}\, k\, \theta^2$$

Hence, the total potential energy:

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

Composing the ODE of the system using Lagrangian

Lagrangian:

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

For viscous rotational damping:

$$\mathcal{R} = \frac{1}{2}\, b\, \dot{\theta}^2$$

Thus, the generalized non-conservative torque contribution entering Lagrange's equation is:

$$Q_{nc} = -\frac{\partial \mathcal{R}}{\partial \dot{\theta}} = -b\dot{\theta}$$

Euler – Lagrange equation:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) - \frac{\partial \mathcal{L}}{\partial \theta} = Q_{nc}$$

Computing derivatives:

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}} = ml^2\dot{\theta} \quad , \qquad \frac{\partial \mathcal{L}}{\partial \theta} = -mgl\sin\theta - k\theta \quad , \qquad \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) = ml^2\ddot{\theta}$$

Thus:

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

Hence, The **Non-linear** ODE derived by the Lagrangian method is:

$$\boxed{ml^2\ddot{\theta} + b\dot{\theta} + mgl\sin\theta + k\theta = 0}$$

If the system were small angle **($\sin\boldsymbol{\theta} \approx \boldsymbol{\theta}$)**, the ODE becomes linear and we can solve it using the same method introduced in the task 1. But for the exact non-linear form we can not solve it analytically because of $\sin\theta$ term in the ODE which causes the non-linearity that makes the integration cannot be solved easily.

For large angles, we use numerical methods such as Runge – Kutta 4 and Euler methods.

Hence, the non-linear ODE only admits numerical solutions.

Solving the linearized ODE analytically for small angles

By substituting $\sin\theta \approx \theta$ we obtain the linear ODE:

$$ml^2\ddot{\theta} + b\dot{\theta} + (mgl + k)\,\theta = 0$$

Dividing by ml^2 gives the standard 2-nd order form:

$$\ddot{\theta} + \frac{b}{ml^2}\dot{\theta} + \frac{mgl + k}{ml^2}\theta = 0$$

Thus define

$$\omega_n = \sqrt{\frac{mgl + k}{ml^2}} = \sqrt{\frac{g}{l} + \frac{k}{ml^2}}\ , \quad 2\xi\omega_n = \frac{b}{ml^2}$$

The equation becomes:

$$\ddot{\theta} + 2\xi\omega_n\dot{\theta} + {\omega_n}^2\theta = 0$$

The characteristic equation is:

$$r^2 + 2\xi\omega_n\, r + {\omega_n}^2 = 0$$

With roots:

$$r_{1,2} = -\xi\omega_n \pm \omega_n\sqrt{\xi^2 - 1}$$

Depending on the value of ξ , we have three cases:

Case 1: Underdamped $\xi < 1$ Two complex conjugate roots

Define the damped frequency:

$$\omega_d = \omega_n\sqrt{1 - \xi^2}$$

And the general solution will be:

$$\theta(t) = e^{-\xi\omega_n t}(C_1\cos\omega_d t + C_2\sin\omega_d t)$$

At $t = 0$:

$$\theta(0) = C_1 = \theta_0$$

Differentiating:

$$\dot{\theta}(t) = e^{-\xi\omega_n t}[-\xi\omega_n(C_1\cos\omega_d t + C_2\sin\omega_d t) + \omega_d(C_2\cos\omega_d t - C_1\sin\omega_d t)]$$

At $t = 0$:

$$\dot{\theta}(0) = -\xi\omega_n C_1 + \omega_d C_2 = \dot{\theta}_0 \implies C_2 = \frac{\dot{\theta}_0 + \xi\omega_n\theta_0}{\omega_d}$$

Case 2: Critical damping $\xi = 1$

$$r_1 = r_2 = -\omega_n$$

And the general solution will be:

$$\theta(t) = (C_1 + C_2 t)\, e^{-\omega_n t}$$

At $t = 0$:

$$\theta(0) = C_1 = \theta_0$$

Differentiating:

$$\dot{\theta}(t) = C_2\, e^{-\omega_n t} - \omega_n(C_1 + C_2 t)\, e^{-\omega_n t}$$

At $t = 0$:

$$\dot{\theta}(0) = C_2 - \omega_n C_1 = \dot{\theta}_0 \implies C_2 = \dot{\theta}_0 + \omega_n\theta_0$$

Case 3: Overdamped $\xi > 1$ Two distinct real roots

$$r_{1,2} = -\xi\omega_n \pm \omega_n\sqrt{\xi^2 - 1}$$

The general solution will be:

$$\theta(t) = Ae^{r_1 t} + Be^{r_2 t}$$

At $t = 0$:

$$A + B = \theta_0 \quad , \quad Ar_1 + Br_2 = \dot{\theta}_0$$

Solving the linear system for A and B:

$$A = \frac{\dot{\theta}_0 - r_2\theta_0}{r_1 - r_2} \quad , \quad B = \frac{r_1\theta_0 - \dot{\theta}_0}{r_1 - r_2}$$

For my values in the table:

Let:

$m = 0.3\ kg,\ \ g = 9.81\ m.s^{-2}\ \ , k = 12\ \ Nm.rad^{-1},\ \ b = 0.035\ \ Nms.rad^{-1}\ ,\ l = 0.49\ m\ , \theta_0 = 0.55$

Then according to the relations mentioned previously we get these values:

$$\omega_n = \sqrt{\frac{mgl + k}{ml^2}} = \sqrt{\frac{g}{l} + \frac{k}{ml^2}} = 13.66\ rad/s$$

$$2\xi\omega_n = \frac{b}{ml^2} = 0.486\ s^{-1}$$

$$\xi = 0.0177 < 1 \Rightarrow \ Underdamped$$

$$\omega_d = \omega_n\sqrt{1-\xi^2} = 13.659\,rad/s$$

And the general solution will be:

$$\theta(t) = e^{-0.0177\ \cdot\ 13.66\ \ t}(0.55\ \cos 13.659\,t + 0.0097\sin 13.659t)$$

Solving the ODE numerically

We will write a code in Python that solves the ODE in three numerical methods:

- Runge-Kutta 4 method
- Implicit Euler method
- Explicit Euler method

Then we will calculate the general solution of the ODE analytically and then plot the function $\theta(t)$ in the 4 different methods and compare the results.

Code

```
import numpy as np

import matplotlib.pyplot as plt

l = 0.49

g = 9.81

k = 12.0

b = 0.035

m = 0.3

def pendulum_dynamics(x):

  """

  Pendulum dynamics: d²θ/dt² = -(g/l) * sin(θ) - (k/(m*(l**2)))*θ - (b/(m*(l**2))) * θ'

  State vector x = [θ, dθ/dt]

  """

  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
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
```

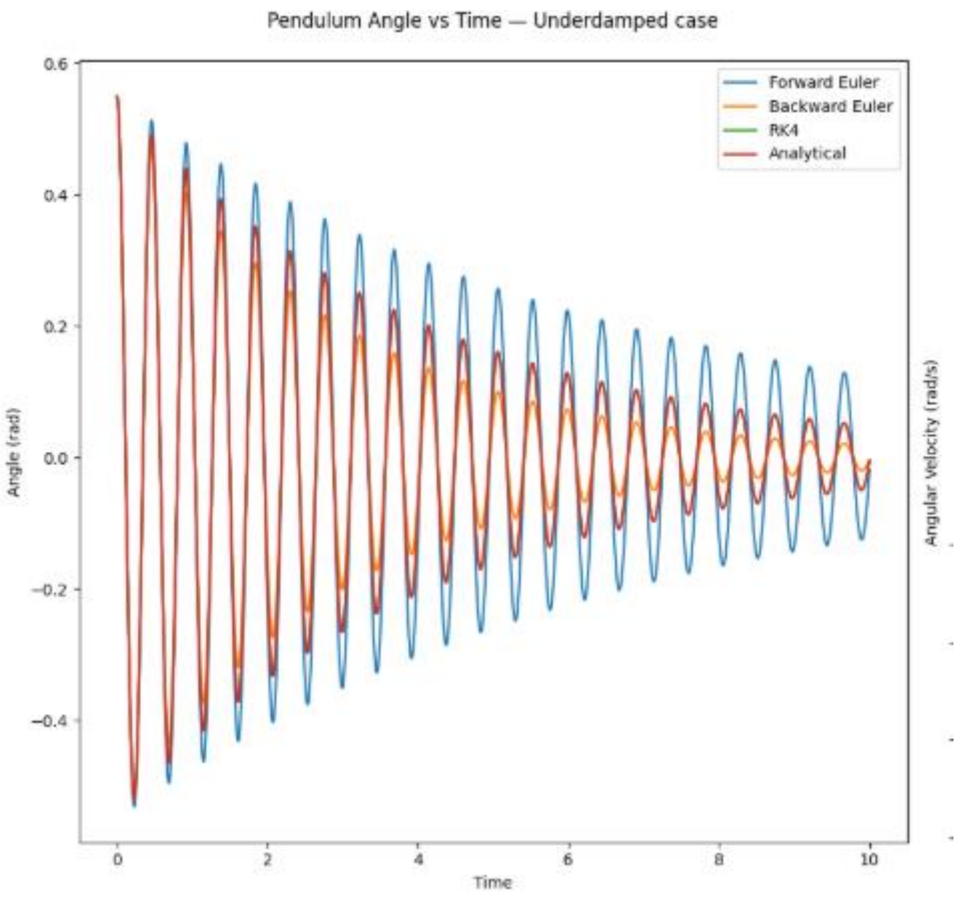
```
def theta_response(t, theta0, theta_dot0, omega_n, zeta):
    if zeta < 1:  # Underdamped
        omega_d = omega_n * np.sqrt(1 - zeta**2)
        C1 = theta0
        C2 = (theta_dot0 + zeta * omega_n * theta0) / omega_d
        theta_t = np.exp(-zeta * omega_n * t) * (C1 * np.cos(omega_d * t) + C2 * np.sin(omega_d * t))
        theta_t_dot= np.gradient(theta_t,t[1]-t[0])
        case = "Underdamped"
    elif np.isclose(zeta, 1.0, atol=1e-3):  # Critically damped
        C1 = theta0
        C2 = theta_dot0 + omega_n * theta0
        theta_t = (C1 + C2 * t) * np.exp(-omega_n * t)
        theta_t_dot= np.gradient(theta_t,t[1]-t[0])
        case = "Critically Damped"
    else:  # Overdamped
        r1 = -zeta * omega_n + omega_n * np.sqrt(zeta**2 - 1)
        r2 = -zeta * omega_n - omega_n * np.sqrt(zeta**2 - 1)
        A = (theta_dot0 - r2 * theta0) / (r1 - r2)
        B = (r1 * theta0 - theta_dot0) / (r1 - r2)
        theta_t = A * np.exp(r1 * t) + B * np.exp(r2 * t)
        theta_t_dot= np.gradient(theta_t,t[1]-t[0])
        case = "Overdamped"
    return theta_t,theta_t_dot, case
# Test all integrators
x0 = np.array([0.5497957901, 0.0])  # Initial state: [angle, angular_velocity]
Tf = 10.0
h = 0.001
# 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)
# Analytical Solution after linearization
theta0= x0[0]
```

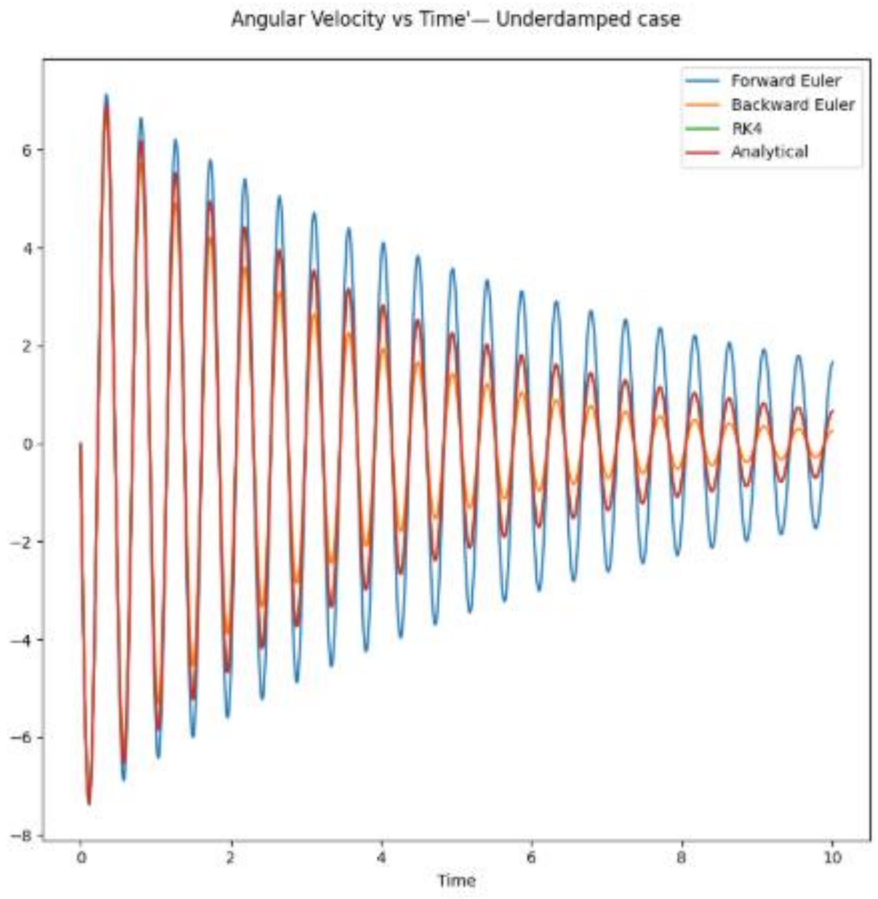
```
theta_dot0 = x0[1]
t = np.arange(0, Tf + h, h)
omega_n = np.sqrt((m * g * l + k) / (m * l**2))  # natural frequency (rad/s)
c = b / (m * l**2)                          # normalized damping coefficient
zeta = c / (2 * omega_n)                    # damping ratio
theta_t,theta_t_dot, case = theta_response(t, theta0, theta_dot0, omega_n, zeta)
# 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.plot(t,theta_t[:], label='Analytical')
plt.xlabel('Time')
plt.ylabel('Angle (rad)')
plt.legend()
plt.title(f"Pendulum Angle vs Time — {case} case\n")
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.plot(t,theta_t_dot[:], label='Analytical')
plt.xlabel('Time')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title(f"Angular Velocity vs Time'— {case} case\n")
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.plot(theta_t[:],theta_t_dot[:], label='Analytical')
plt.xlabel('Angle (rad)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Phase Portrait')
plt.tight_layout()

plt.show()
```

Results

Plotting the function $\theta(t)$ using the 3 numerical methods comparing to the analytical solution:





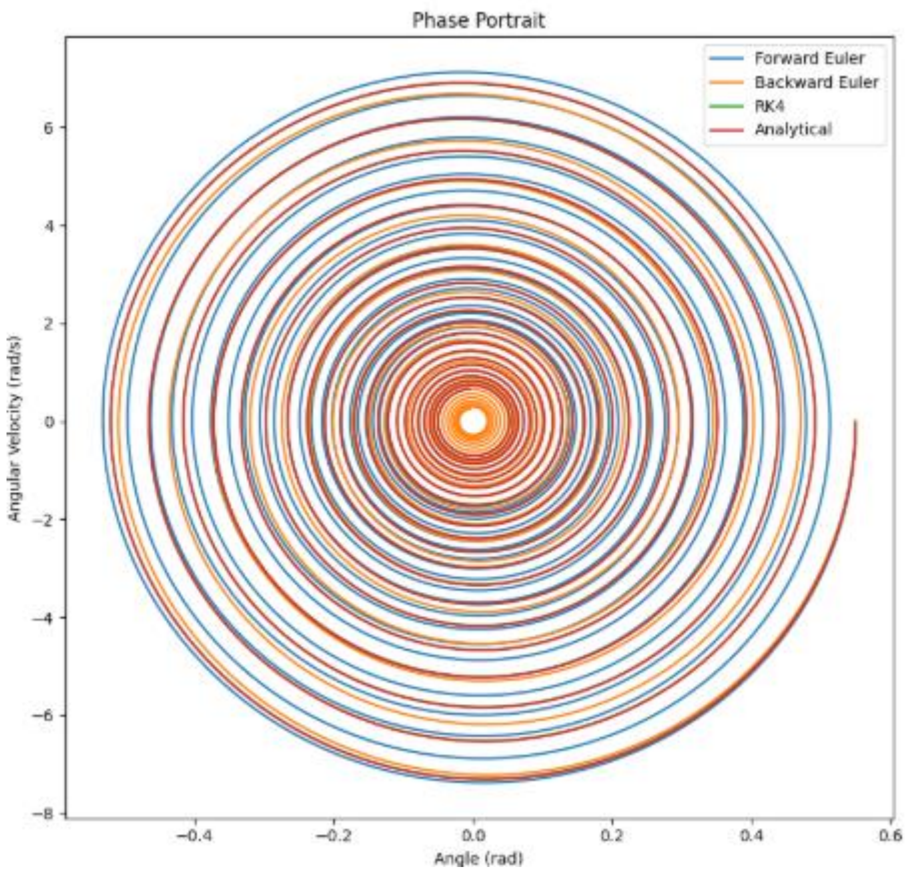
We can note that the Forward Euler method (blue line) diverges slightly from the analytical curve (red) over time.

It tends to overestimate the amplitude and oscillation frequency due to its explicit and first-order nature, which accumulates numerical errors at each step.

The Backward Euler method is much more stable than the Forward Euler method.

The RK4 solution (green) almost perfectly overlaps with the analytical solution (red). This demonstrates RK4's superior balance between accuracy and stability for smooth nonlinear systems.

The analytical solution provides a precise reference for small angles under the linearized assumption ($\sin\theta \approx \theta$). The close match with RK4 confirms that the small-angle approximation holds for this case.



From the phase portrait we can note that the Analytical solution (red) and RK4 (green) are nearly identical, showing consistent amplitude decay. Meanwhile, Forward Euler shows outward spirals which means numerical instability (energy gain) whereas, Backward Euler shows inward spirals which means excessive damping (energy loss).