THE MINISTRY OF SCIENCE AND HIGHER EDUCATION
OF THE RUSSIAN FEDERATION

ITMO University
(ITMO)

Faculty of Control Systems and Robotics

SYNOPSIS
for the subject
*"Simulation of Robotic Systems"*

on the topic:
Practice 2

Student:
*R4131c, 506185* *Movsesyan Arthur*

Tutor:
*Assistant, MSc* *Rakshin Egor*

Saint Petersburg 2025

## Task

1. Compose the ODE from the mass spring damper system.
2. **Try** to solve analytically the ODE you composed.
3. Compare results of these methods with analytical solution (if it exists), dicsuss and conclude your thoughts.

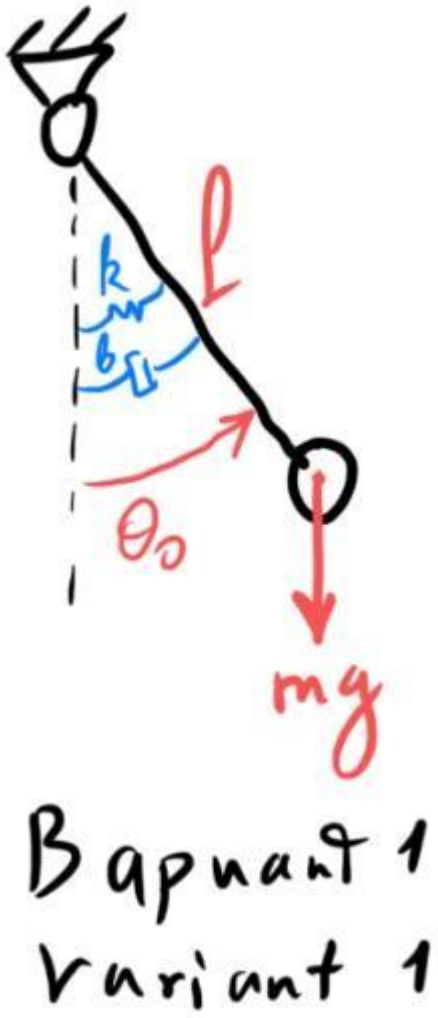


Fig. 1. Mass spring damper system.

| m, kg | k, N/m, Nm/rad | b, N*s/m, Nm*s/rad | l, m | theta_0, rad | x_0, m |
|---|---|---|---|---|---|
| 0.3 | 17.4 | 0.05 | 0.25 | 0.1256676092 | 0.75 |

Fig. 2. Parameters table.

**Solution:**

Pendulum-Spring-Damper System: Analytical Derivation

Kinetic Energy of the Pendulum:
$K = \frac{1}{2} m l^2 \dot{\theta}^2$

Potential Energy of the Pendulum:
$P = m g l (1 - \cos \theta) + \frac{1}{2} k \theta^2$

Lagrangian:
$\mathscr{L} = K - P = \frac{1}{2} m l^2 \dot{\theta}^2 - \frac{1}{2} k \theta^2 - m g l (1 - \cos \theta)$

Equation of Motion with Damping ($Q = -b \dot{\theta}$):
$I \ddot{\theta} + b \dot{\theta} + k \theta + m g l \sin \theta = 0, \quad$ where $I = m l^2$

**Analytical Solution for the Linearized System (Small Angles)**

Linearized Equation of Motion ($\sin \theta \approx \theta$):
$I \ddot{\theta} + b \dot{\theta} + (k + m g l) \theta = 0$

Effective Stiffness:
$k\_eff = k + m g l$

Natural Frequency:
$\omega_n = \sqrt{(k\_eff / I)}$

Damping Ratio:
$\zeta = b / (2 \sqrt{(k\_eff\ I)})$

Damped Natural Frequency:
$\omega\_d = \omega_n \sqrt{(1 - \zeta^2)}$

General Solution (Underdamped, $\zeta < 1$):
$\theta(t) = e^{(-\zeta \omega_n t)} [C_1 \cos(\omega\_d\ t) + C_2 \sin(\omega\_d\ t)]$

Constants from Initial Conditions ($\theta(0) = \theta_0$, $\dot{\theta}(0) = 0$):
$C_1 = \theta_0$
$C_2 = (\zeta \omega_n / \omega\_d) \theta_0$

Final Closed-Form Solution:

$\theta(t) = e^{(-\zeta \omega_n t)} [ \theta_0 \cos(\omega_d t) + (\zeta \omega_n \theta_0 / \omega_d) \sin(\omega_d t) ]$

Final Numerical Solution:

$\theta(t) = e^{(-1.333 t)} [ 0.12567 \cos(31.07 t) + 0.004291 \sin(31.07 t) ]$ (rad)

Angular Velocity (Derivative):

$\dot{\theta}(t) = e^{(-1.333 t)} [ -0.0329 \cos(31.07 t) - 3.905 \sin(31.07 t) ]$ (rad/s)

Python solution:

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

def pendulum_dynamics(x):
    """
    Pendulum dynamics
    State vector x = [θ, dθ/dt]
    """
    m = 0.3
    k = 17.4
    b = 0.05
    l = 0.25
    g = 9.81

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

    theta_ddot = -1 / (m * l * l) * ( b * theta_dot + m * g * l *
np.sin(theta) + k * theta)

    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 analytical_solution(t, theta0=0.1256676092):
    """
    Analytical solution for the linearized pendulum system (small
angles)
    theta(t) = exp(-zeta*wn*t) * [C1*cos(wd*t) + C2*sin(wd*t)]
    with initial conditions: theta(0) = theta0, theta_dot(0) = 0
    """
    m = 0.3
    k = 17.4
    b = 0.05
    l = 0.25
    g = 9.81

    I = m * l**2
    K_eff = k + m * g * l

    wn = np.sqrt(K_eff / I)          # natural frequency
    zeta = b / (2 * np.sqrt(K_eff * I))  # damping ratio
    wd = wn * np.sqrt(1 - zeta**2)   # damped frequency

    C1 = theta0
    C2 = (zeta * wn * theta0) / wd

    theta = np.exp(-zeta * wn * t) * (C1 * np.cos(wd * t) + C2 *
np.sin(wd * t))
```

```
    # Derivative for angular velocity
    theta_dot = np.exp(-zeta * wn * t) * (
        (-zeta * wn * C1 + wd * C2) * np.cos(wd * t) +
        (-zeta * wn * C2 - wd * C1) * np.sin(wd * t)
    )
    return np.vstack([theta, theta_dot])

# Test all integrators
x0 = np.array([0.1256676092, 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)

# Compute analytical solution on the same time grid as RK4 (most
accurate)
x_analytical = analytical_solution(t_rk4)

# Plot results — updated to include analytical solution
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_rk4, x_analytical[0, :], 'k--', label='Analytical',
linewidth=2)
plt.xlabel('Time (s)')
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.plot(t_rk4, x_analytical[1, :], 'k--', label='Analytical',
linewidth=2)
plt.xlabel('Time (s)')
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.plot(x_analytical[0, :], x_analytical[1, :], 'k--',
label='Analytical', linewidth=2)
plt.xlabel('Angle (rad)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Phase Portrait')
plt.grid(True)

plt.tight_layout()
plt.show()
```

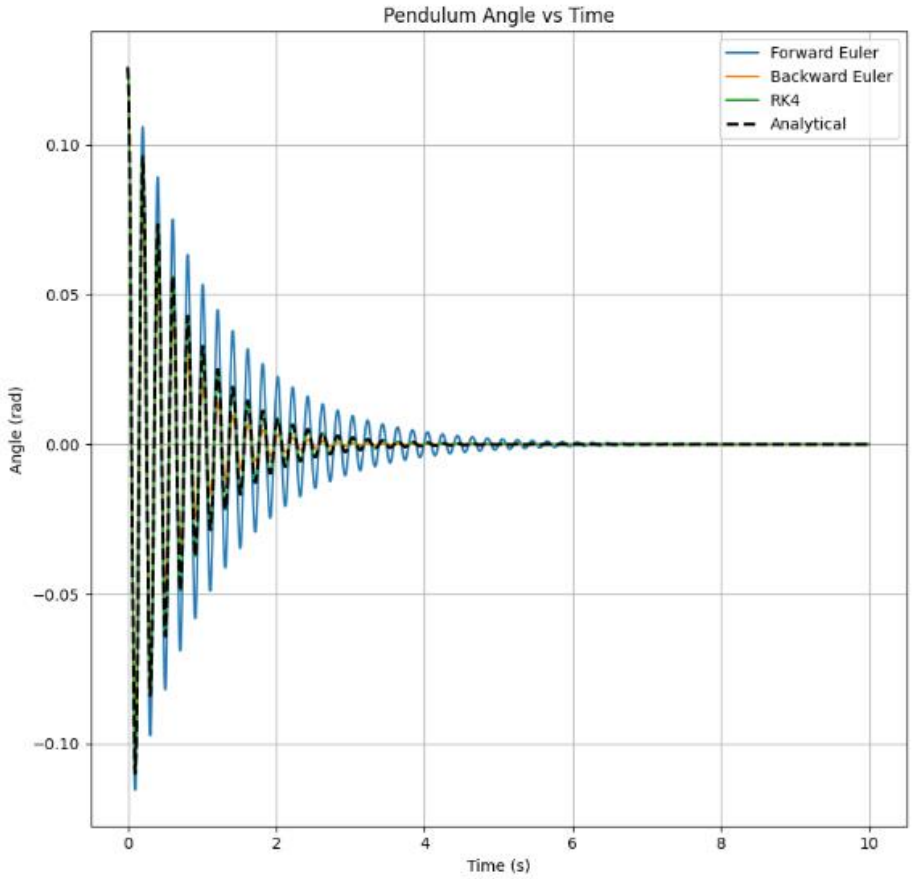


Fig. 3. Angle as function of time

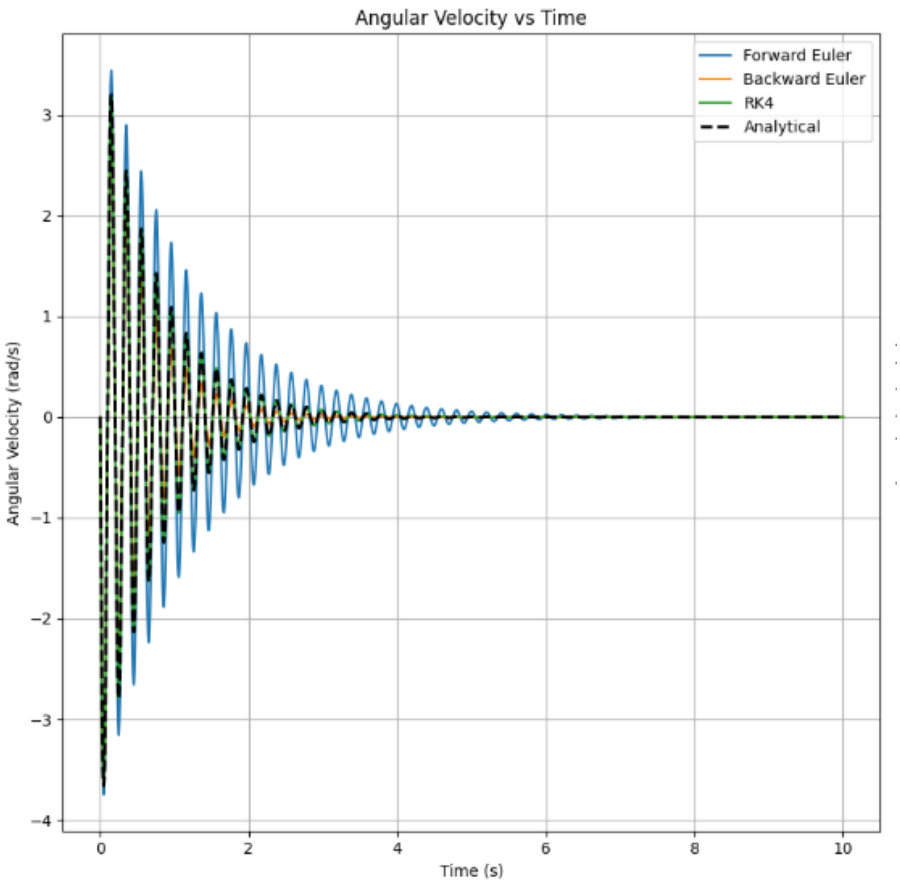


Fig. 4. Angular velocity as function of time

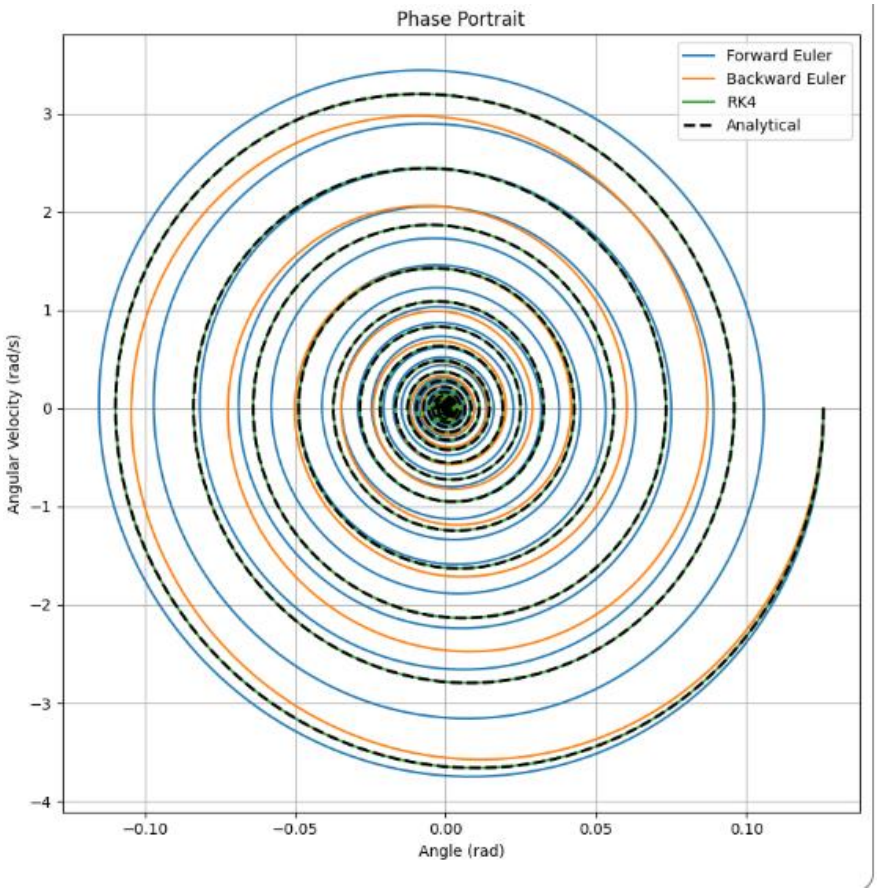

Fig. 4. Phase Portrait.

**Conclusions:**

The 4th-order Runge-Kutta (RK4) method closely matches the analytical solution for both angle and angular velocity, confirming its high accuracy for this damped pendulum system. The Forward Euler method exhibits unstable, non-decaying oscillations, while the Backward Euler method overdamps the response — both deviate significantly from the true dynamics. In the phase portrait, RK4 and the analytical solution spiral inward toward equilibrium with nearly identical trajectories, whereas Forward Euler spirals outward (unstable) and Backward Euler collapses too rapidly.