МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

(Университет ИТМО)

Международный научно-образовательный центр

Физики наноструктур

Отчет лаб 2

по дисциплине:

Simulation of Robotic Systems

Студент:

Группа № R4134c *Буй Динь Кхай Нгуен*

Преподаватель: *Ракшин Егор Александрович*

Санкт-Петербург 2025

TABLE OF CONTENTS

1. INPUT DATA OVERVIEW

The objective is to model and analyze the dynamic behavior of **Variant 2**, a horizontal mass-spring-damper system, using the Lagrangian approach described in the provided lecture.

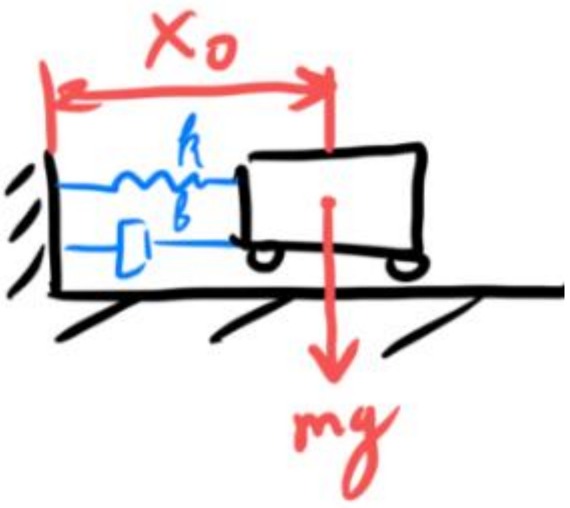


Figure 1: Variant 2 - Horizontal mass-spring-damper system

1.1 Parameters

The system consists of a mass (m) connected to a fixed frame through a spring (k) and a viscous damper (b). The motion is described by a single generalized coordinate $x(t)$, representing horizontal displacement.

Parameter	**Symbol**	**Value**	**Unit**	**Role**
Mass	m	1.0	kg	Inertia
Stiffness	k	11.8	N/m	Restoring force (spring)
Damping	b	0.02	N·s/m	Dissipative force (damper)
Initial Displacement	x_0	0.94	m	Initial position $x(0)$

Table 1: System Parameters (Variant 2)

1.2 Initial Conditions

The system is released from the initial displacement x_0:

$$x(0) = x_0 = 0.94 \text{ m}, \qquad \dot{x}(0) = 0 \text{ m/s}$$

2. DERIVATION OF THE EQUATION OF MOTION

The governing differential equation of motion is derived using Lagrange's Equation, which requires calculating the kinetic energy ($\mathcal{K}$) and potential energy ($\mathcal{P}$) to form the Lagrangian ($\mathcal{L}$).

2.1 Lagrangian Formulation

The generalized coordinate is the displacement x.

1. **Kinetic Energy**

$$\mathcal{K} = \frac{1}{2}m\dot{x}^2$$

2. **Potential Energy**
   Because the system is horizontal, gravitational potential energy $(mgx)$is neglected—it is balanced by the normal reaction force.
   Only the elastic potential energy of the spring is considered:

$$\mathcal{P} = \frac{1}{2}kx^2$$

3. **Lagrangian**

$$\mathcal{L}(x,\dot{x}) = \mathcal{K} - \mathcal{P} = \frac{1}{2}m\dot{x}^2 - \frac{1}{2}kx^2$$

4. **Non-Conservative Force** The external non-conservative force $Q$is the damping force:

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

2.2 Applying Lagrange's Equation

For a one-dimensional system:

$$\frac{d}{dt}(\frac{\partial\mathcal{L}}{\partial\dot{x}}) - \frac{\partial\mathcal{L}}{\partial x} = Q$$

Compute each term:

$$\frac{\partial\mathcal{L}}{\partial\dot{x}} = m\dot{x} \Rightarrow \frac{d}{dt}(\frac{\partial\mathcal{L}}{\partial\dot{x}}) = m\ddot{x}$$

$$\frac{\partial \mathcal{L}}{\partial x} = -kx$$

Substitute into Lagrange's equation:

$$m\ddot{x} - (-kx) = -b\dot{x}$$

Rearranging yields the standard **second-order linear ODE**:

$$m\ddot{x} + b\dot{x} + kx = 0$$

Substituting $m = 1$, $b = 0.02$, and $k = 11.8$:

$$\ddot{x} + 0.02\dot{x} + 11.8x = 0$$

3. ANALYTICAL SOLUTION

The derived model is a **linear, homogeneous, second-order ODE** with constant coefficients, which can be solved analytically.

3.1. General Solution

$$\ddot{x} + 0.02\dot{x} + 11.8x = 0$$

Assuming a solution of $x(t) = e^{rt}$ gives the **characteristic equation**:

$$r^2 + 0.02r + 11.8 = 0$$

Solving for r:

$$r = \frac{-0.02 \pm \sqrt{(0.02)^2 - 4(1)(11.8)}}{2} = \frac{-0.02 \pm i\sqrt{47.1996}}{2}$$

$$r \approx -0.01 \pm 3.435i$$

Since the roots are complex conjugates, the system is **underdamped**, and the general solution becomes:

$$x(t) = e^{\alpha t}(C_1 \cos(\omega_d t) + C_2 \sin(\omega_d t))$$

Where:

$$\alpha = -0.01 \text{ and } \omega_d \approx 3.435 \text{ rad/s.}$$

3.2. Specific Solution with Initial Conditions

At $t = 0$:

$$x(0) = 0.94 \Rightarrow C_1 = 0.94$$

$$\dot{x}(0) = 0 \Rightarrow \alpha C_1 + \omega_d C_2 = 0 \Rightarrow C_2 = -\frac{\alpha C_1}{\omega_d} = -\frac{(-0.01)(0.94)}{3.435} \approx 0.002736$$

Analytical Solution:

$$x(t) = e^{-0.01t}(0.94\cos(3.435t) + 0.002736\sin(3.435t))$$

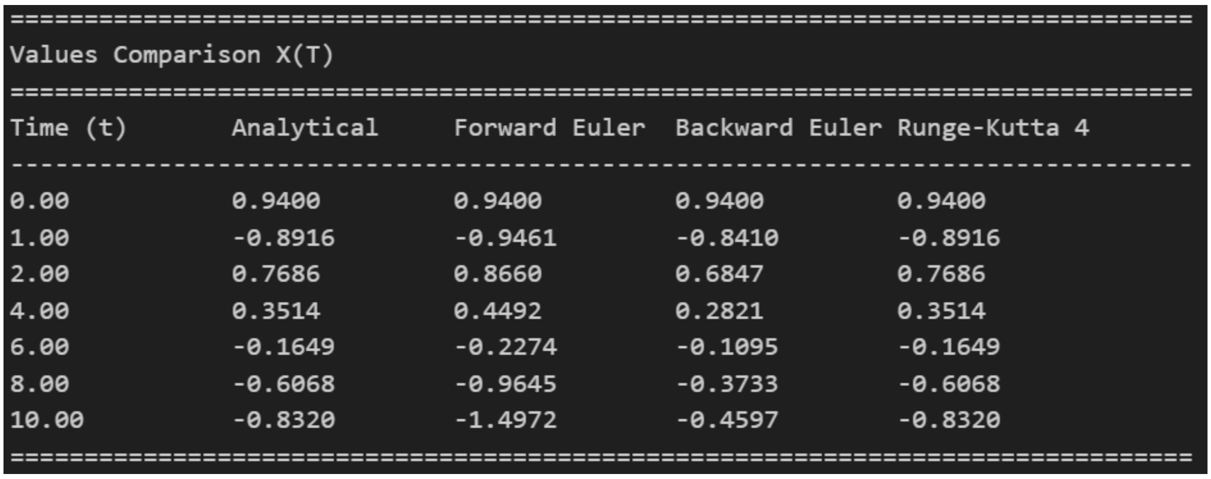
4. APPENDIX

Below is the code implemented in python to calculate $\boldsymbol{x}(\boldsymbol{t})$ in both analytical and numerical integration methods for comparison:

```python
import numpy as np

import matplotlib.pyplot as plt

# SYSTEM PARAMETERS
m = 1.0
k = 11.8
b = 0.02

# DYNAMICS FUNCTION
def my_ode_dynamics(x):
    """
    Dynamics for the Mass-Spring-Damper system (Variant 2): m*x'' + b*x' + k*x
= 0
    State vector x = [x, x_dot]

    Global parameters m, k, b
    """
    y1 = x[0]   # x
    y2 = x[1]   # x_dot

    # x'' = (-b*x' - k*x) / m
    y1_dot = y2
    y2_dot = (-b * y2 - k * y1) / m

    return np.array([y1_dot, y2_dot])

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

# ANALYTICAL SOLUTION
def analytical_solution(t):
    """
    x(t) = e^(alpha*t) * (C1*cos(omega_d*t) + C2*sin(omega_d*t))
    alpha = -b/(2m) = -0.01
    omega_d = sqrt(k/m - alpha^2) ~ 3.435 rad/s
    C1 = x(0) = 0.94
    C2 = -alpha*C1 / omega_d ~ 0.002736
    """
    alpha = -0.01
    omega_d = np.sqrt(k/m - alpha**2) # 3.43509
    C1 = 0.94
    C2 = -alpha * C1 / omega_d # 0.0027366

    return np.exp(alpha * t) * (C1 * np.cos(omega_d * t) + C2 * np.sin(omega_d
* t))

# PRINT COMPARISON TABLE
def print_comparison_results(t_values, t_full, x_an, x_fe, x_be, x_rk4):
    # Setup tables
    data_values = []
    data_errors = []

    for t in t_values:
        # find the closest index in t_full
        index = np.argmin(np.abs(t_full - t))

        val_an = x_an[index]
        val_fe = x_fe[0, index]
        val_be = x_be[0, index]
        val_rk4 = x_rk4[0, index]

        # Compute the absolute error
        error_fe = np.abs(val_fe - val_an)
        error_be = np.abs(val_be - val_an)
        error_rk4 = np.abs(val_rk4 - val_an)

        # Save data
        data_values.append((t, val_an, val_fe, val_be, val_rk4))
        data_errors.append((t, error_fe, error_be, error_rk4))

    # Values comparison
    print("\n" + "="*80)
    print("Values Comparison X(T)")
    print("="*80)
    print(f"{'Time (t)':<15}{'Analytical':<15}{'Forward Euler':<15}{'Backward
Euler':<15}{'Runge-Kutta 4':<20}")
    print("-"*80)
    for t, an, fe, be, rk4 in data_values:
        print(f"{t:<15.2f}{an:<15.4f}{fe:<15.4f}{be:<15.4f}{rk4:<20.4f}")
    print("="*80)

    # Error comparison
    print("\n" + "="*80)
    print("Absolute Error Comparison |X_NUM - X_AN|")
    print("="*80)
    print(f"{'Time (t)':<15}{'Error FE':<15}{'Error BE':<15}{'Error
RK4':<15}")
    print("-"*80)
    for t, err_fe, err_be, err_rk4 in data_errors:
        print(f"{t:<15.2f}{err_fe:<15.2e}{err_be:<15.2e}{err_rk4:<15.2e}")
    print("="*80)

# SET UP AND RUN SIMULATIONS
x0_ode = np.array([0.94, 0.0])  # [x(0), x'(0)]
Tf = 10.0                       # Simulation time
h = 0.01                       # Time step

# Run the integrators
x_fe, t_full = forward_euler(my_ode_dynamics, x0_ode, Tf, h)
x_be, t_full = backward_euler(my_ode_dynamics, x0_ode, Tf, h)
x_rk4, t_full = runge_kutta4(my_ode_dynamics, x0_ode, Tf, h)

# Compute the analytical solution at the time points
x_an = analytical_solution(t_full)

# Time points for comparison
t_comparison = [0.0, 1.0, 2.0, 4.0, 6.0, 8.0, 10.0]

# PRINT COMPARISON TABLE
print_comparison_results(t_comparison, t_full, x_an, x_fe, x_be, x_rk4)


# PLOT COMPARISON
plt.figure(figsize=(10, 6))

plt.plot(t_full, x_an, 'k-', label='Analytical Solution', linewidth=3,
alpha=0.7)
plt.plot(t_full, x_fe[0, :], 'r--', label='Forward Euler')
plt.plot(t_full, x_be[0, :], 'b-.', label='Backward Euler')
plt.plot(t_full, x_rk4[0, :], 'g:', label='Runge-Kutta 4')

plt.xlabel('Time (t)')
plt.ylabel('Position x(t)')
plt.title(f'Comparison of ODE Integrators (h={h})')
plt.legend()

plt.grid(True)
plt.show()
```

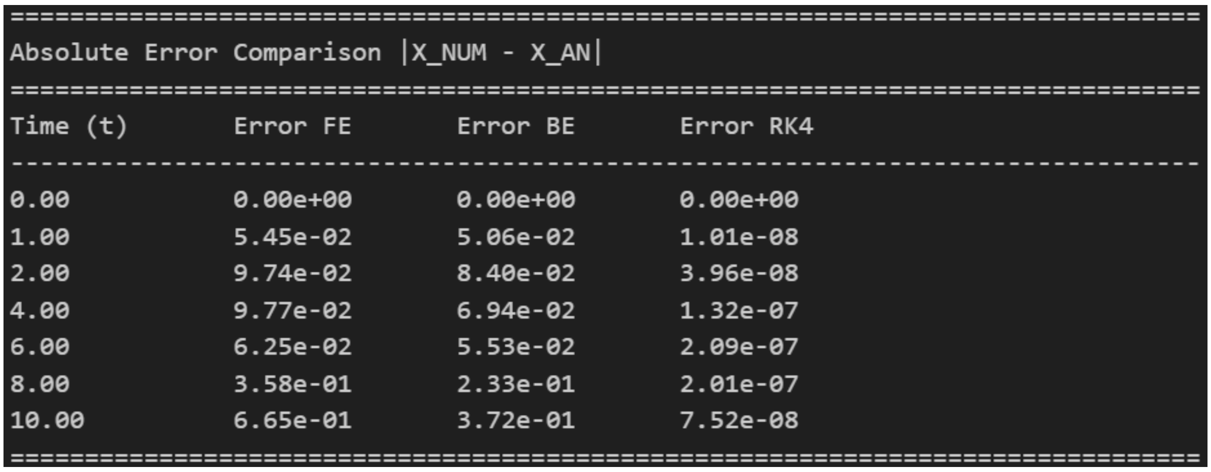
5. RESULTS

5.1 Comparison table



```
================================================================================
Values Comparison X(T)
================================================================================
Time (t)       Analytical     Forward Euler  Backward Euler Runge-Kutta 4
--------------------------------------------------------------------------------
0.00           0.9400         0.9400         0.9400         0.9400
1.00           -0.8916        -0.9461        -0.8410        -0.8916
2.00           0.7686         0.8660         0.6847         0.7686
4.00           0.3514         0.4492         0.2821         0.3514
6.00           -0.1649        -0.2274        -0.1095        -0.1649
8.00           -0.6068        -0.9645        -0.3733        -0.6068
10.00          -0.8320        -1.4972        -0.4597        -0.8320
================================================================================
```

Table 2: Comparison of $\boldsymbol{x(t)}$ Values (step size h=0.01)



```
================================================================================
Absolute Error Comparison |X_NUM - X_AN|
================================================================================
Time (t)       Error FE       Error BE       Error RK4
--------------------------------------------------------------------------------
0.00           0.00e+00       0.00e+00       0.00e+00
1.00           5.45e-02       5.06e-02       1.01e-08
2.00           9.74e-02       8.40e-02       3.96e-08
4.00           9.77e-02       6.94e-02       1.32e-07
6.00           6.25e-02       5.53e-02       2.09e-07
8.00           3.58e-01       2.33e-01       2.01e-07
10.00          6.65e-01       3.72e-01       7.52e-08
================================================================================
```

Table 3: Comparison of Absolute Error (step size h=0.01)

5.2 Graphic comparison

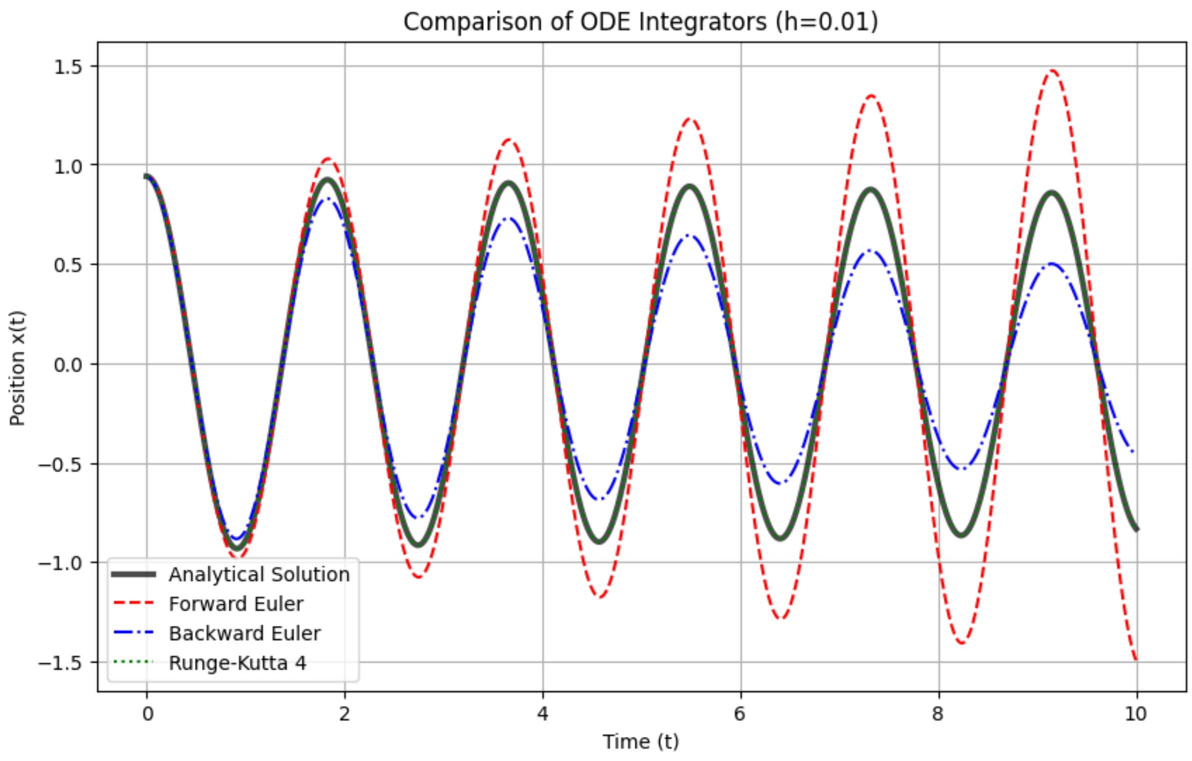


Figure 2: Comparison of ODE Integrators (step size h=0.01)

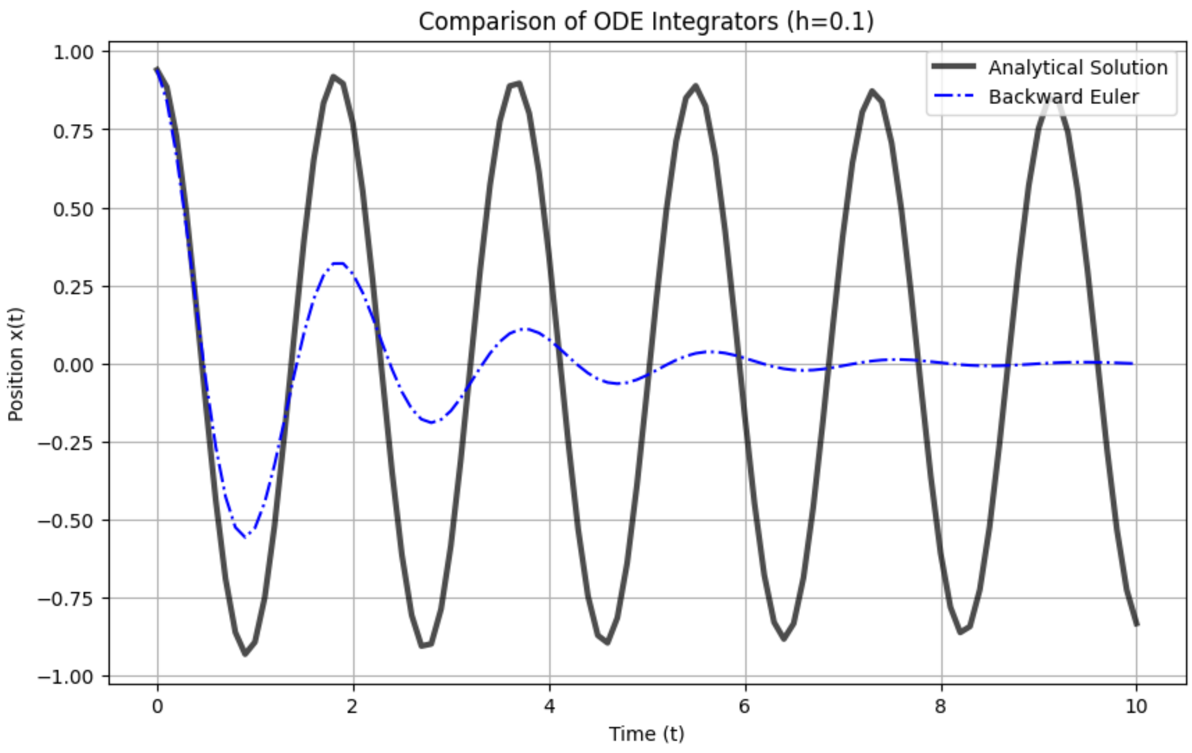


Figure 3: Comparison of ODE Integrators (step size h=0.1)

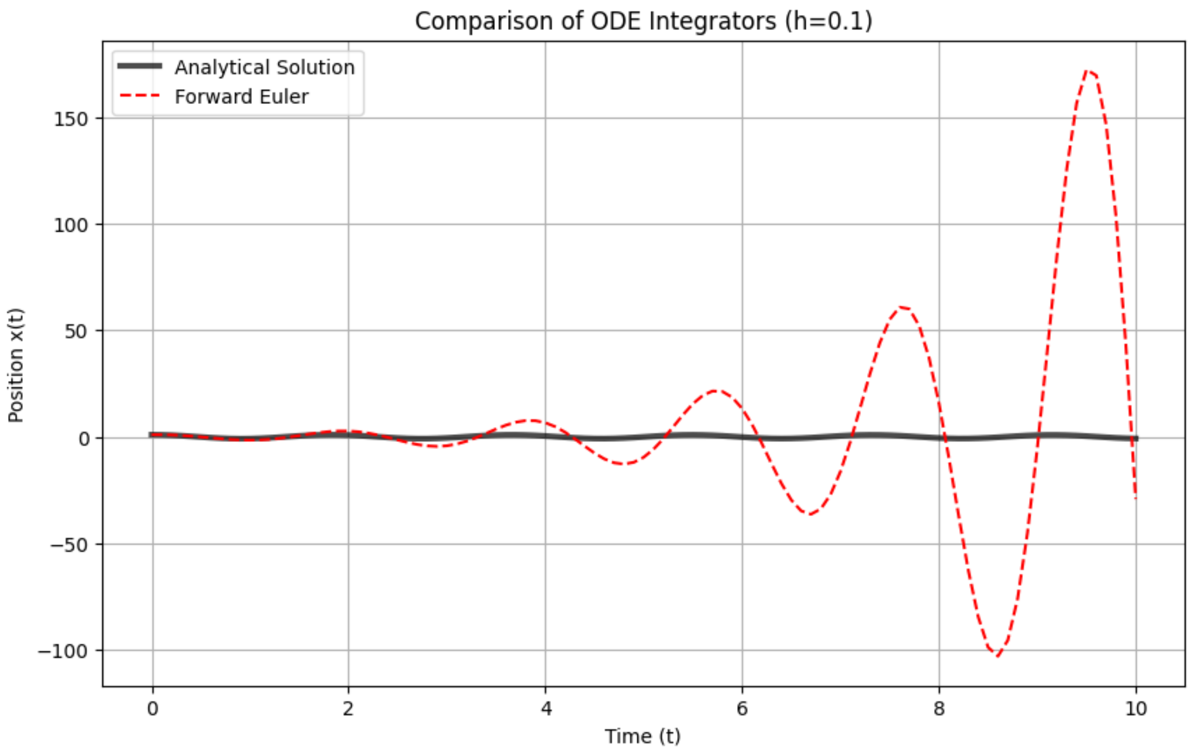


Figure 4: Comparison of ODE Integrators (step size h=0.1)

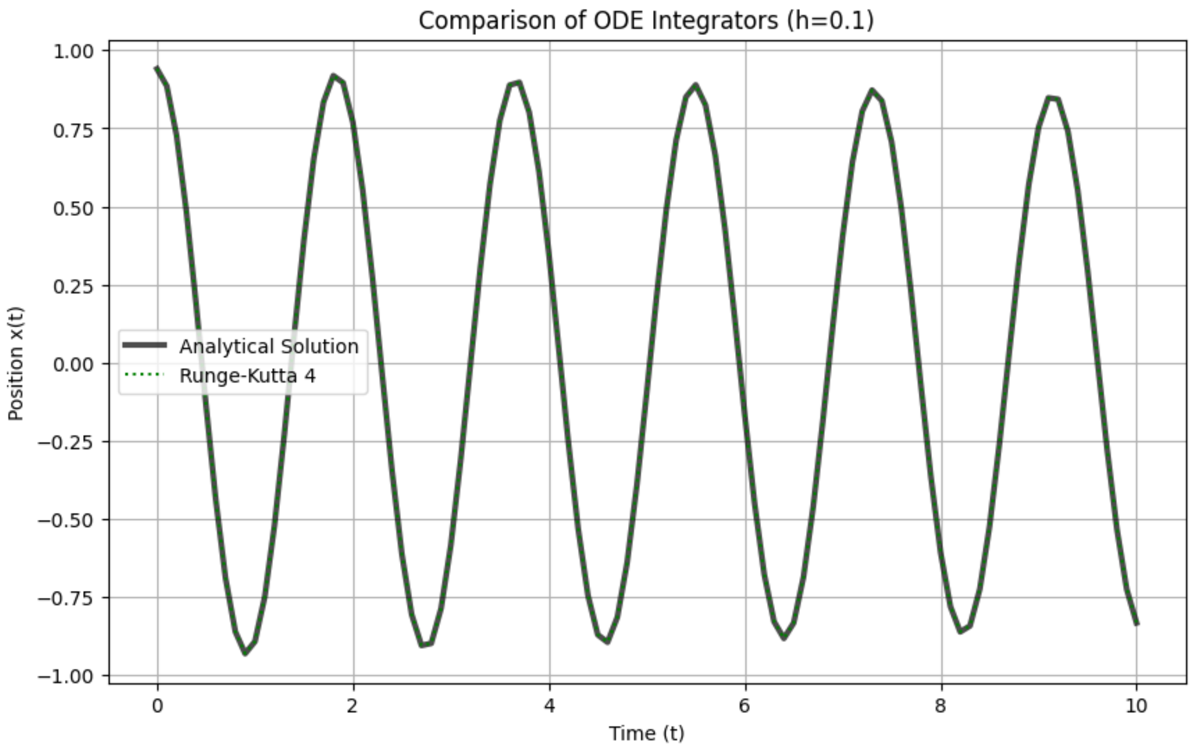


Figure 5: Comparison of ODE Integrators (step size h=0.1)

6. DISCUSSION AND CONCLUSION

The simulation results — illustrated in the plots and quantified in the error tables for $\mathbf{h = 0.01}$— reveal clear differences in accuracy and stability among the tested numerical integration methods when applied to the underdamped Mass–Spring–Damper system

6.1. Runge–Kutta 4 (RK4)

- **Accuracy:**
  RK4 exhibits exceptionally high accuracy, with a maximum absolute error of approximately $\mathbf{2 \times 10^{-7}}$at $\boldsymbol{t = 8.00}$ swhen using a time step of $\mathbf{h = 0.01}$.
  In Figure 2, the RK4 solution is visually indistinguishable from the analytical solution, confirming its high order of convergence.
- **Stability:**
  Even at $\mathbf{h = 0.1}$ (Figure 5), RK4 maintains perfect synchronization with the analytical solution, verifying its robustness for modeling systems with smooth, continuous dynamics.
  This demonstrates RK4's superiority for accurately simulating mechanical systems over long durations.

6.2. Forward Euler (FE)

- **Accuracy & Stability ($\mathbf{h = 0.01}$):**
  Although the initial response closely matches the analytical solution, FE quickly develops a major flaw:
  Numerical instability leading to amplitude growth.
  The error table shows that the absolute error rises sharply to
  $\mathbf{6.65 \times 10^{-1}}$ at $\boldsymbol{t = 10.00}$ **s**.
  In Figure 2 (dashed red line), the FE solution's amplitude erroneously increases over time, falsely suggesting that energy is being added to the system — a contradiction of the damped system's physics.

- **Instability ($\mathbf{h = 0.1}$):**
  With the larger step size ($\boldsymbol{h = 0.1}$), instability becomes catastrophic. The FE solution blows up, with amplitudes exceeding $\mathbf{\pm 150}$ (Figure 4), clearly proving that explicit Forward Euler is unstable for this type of oscillating system at this step size.

6.3. Backward Euler (BE)

- **Accuracy & Stability ($\mathbf{h = 0.01}$):**
  As an implicit method, BE is more stable than FE, but it suffers from a systematic error:
  Instead of increasing amplitude incorrectly, it damps the oscillations too quickly.
  At $\boldsymbol{t = 10.00}$ **s**, BE yields $\boldsymbol{|\ x\ | = 0.4597,}$ noticeably smaller than the analytical $\boldsymbol{|\ x\ | = 0.8320}$, with a maximum absolute error of
  $\boldsymbol{2.33 \times 10^{-1}}$ at $\boldsymbol{t = 8.00}$ **s**.

- **Stability ($\mathbf{h = 0.1}$):**
  This issue worsens at $\boldsymbol{h = 0.1}$ (Figure 3), where BE drives the solution to $\boldsymbol{x \approx 0}$ by $\boldsymbol{t \approx 6}$ **s**, while the analytical model still shows significant oscillations.
  This artificial energy loss prevents BE from accurately representing the true system dynamics.

6.4. Conclusion

The primary objective — producing an accurate simulation of the underdamped mass–spring–damper system's dynamics — has been successfully achieved.

- **Model Validation:**
  The close agreement between the RK4 numerical solution and the analytical solution validates:
  1. The correctness of the derived governing equation:
  $$m\ddot{x} + b\dot{x} + kx = 0$$
  2. The high accuracy of the RK4 integrator for this class of second-order mechanical systems.

- **Integrator Performance:**
  - **RK4:** The optimal choice among the tested methods — delivering high precision (error on the order of 10^{-7}) and maintaining stability across all tested step sizes.
  - **Forward Euler:** Highly unsuitable for oscillating systems; its numerical instability leads to spurious amplitude growth, violating the physical law of energy dissipation in damped systems.

- **Backward Euler:** Generally stable, but introduces excessive numerical damping, causing amplitudes to decay much faster than the real system's response.