Ministry of Education and Science of the Russian Federation,

Federal State Autonomous Educational Institution of Higher Education, St. Petersburg National Research University of Information Technology, Mechanics, and Optics

Report on Practical Work No. 2 on the course

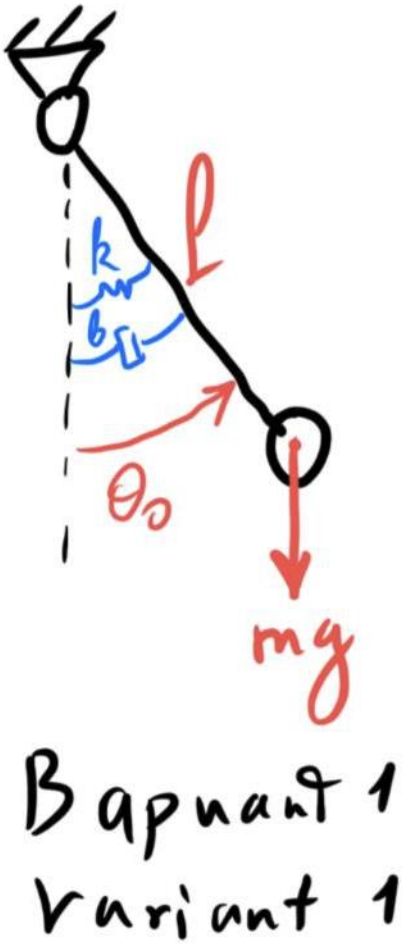
"Simulation Modeling of Robotic Systems"

Report completed by: Xiangzhang (508513)

Instructor: E.A. Rakshin (373529)

Date completed: November 10, 2025

Saint Petersburg, 2025



Variant1

1.Handwrite analytical solution:

Correspond parameters: m=0.4kg, k=9.6N/m, b=0.01Ns/m,

l=0.95m, theta=-0.1413760603rad

$L(x,\dot{x}) = k(x,\dot{x}) - P(x) = \frac{1}{2}m\dot{x}^2 - (mgx + \frac{1}{2}kx^2) = \frac{1}{2}mL^2\dot{\theta}^2 - [mgL(1-\cos\theta) + \frac{1}{2}k(R_k\theta)^2]$

Linear velocity : $V = L\cdot\dot{\theta}$ $\quad \frac{\partial L}{\partial \dot{\theta}} = m\cdot L^2\cdot\dot{\theta}$ $\quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = m\cdot L^2\cdot\ddot{\theta}$

arc length : $L = r\cdot\theta$

$$\frac{\partial L}{\partial \theta} = -mgL\sin\theta - kR_k^2\theta$$

External Force: $Q = \tau_b = \vec{F_b}\times\vec{R_b} = (-b\cdot R_b\cdot\dot{\theta})\cdot R_b = -b\cdot R_b^2\cdot\dot{\theta}$

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

$$m\cdot L^2\cdot\ddot{\theta} + mgL\sin\theta + k\cdot R_k^2\cdot\theta = -b\cdot R_b^2\cdot\dot{\theta}$$

We assume that the $R_b = \frac{1}{2}L$, $R_k = \frac{1}{4}L$.

$$\ddot{\theta} + \frac{b\cdot R_b^2}{mL^2}\dot{\theta} + \frac{mg\cdot L\cdot\sin\theta}{mL^2} + \frac{k\cdot R_k^2}{mL^2}\theta = 0$$

$$\ddot{\theta} + \frac{b\cdot\frac{1}{4}L^2}{mL^2}\cdot\dot{\theta} + \frac{g}{L}\sin\theta + \frac{k\cdot\frac{1}{16}L^2}{mL^2}\cdot\theta = 0$$

Input parameters : $m = 0.4\,kg$, $K = 9.6\,N/m$, $b = 0.01\,Ns/m$ $L = 0.95\,m$

$$\theta = -0.141376063\ rad \approx -8.1^\circ$$

$$\ddot{\theta} + \frac{1}{160}\dot{\theta} + 10.32\sin\theta + 0.24\theta = 0$$

because $\theta = -8.1^\circ$, $|\theta| < 10^\circ$, $\sin\theta \approx \theta$

$$\ddot{\theta} + \frac{1}{160}\dot{\theta} + 10.56\theta = 0.$$

$$\theta(t) = e^{-0.003125t}\left[C_1\cos(3.25t) + C_2\sin(3.25t)\right]$$

2.Code Implementation：

First we input know parameters and we can calculate the coefficient of the components of ODE.

```
# Known physical parameters

m = 0.4
b = 0.01
k = 9.6
l = 0.95
g = 9.81
theta0 = -0.1413760603
omega0 = 0.0

# We assume the positions of the damper b and the spring k
Rb = 0.5 * l
Rk = 0.25 * l



# Based on the derived physical equations:
# m*l^2*θ¨ + m*g*l*sinθ + k*Rk^2*θ = -b*Rb^2*θ˙
# => θ¨ = -(b*Rb²/(m*l²)) θ˙ - (g/l + k*Rk²/(m*l²)) sinθ

c = b * Rb**2 / (m * l**2)
omega_sq = (g / l) + (k * Rk**2) / (m * l**2)
```

Then we can use Explicit Euler ,Implicit Euler and RK4 methods to integrate ODE.

```
# Explicit Euler

def forward_euler(f, x0, t0, tf, h):
    t = np.arange(t0, tf + h, h)
    x = np.zeros((len(x0), len(t)))
    x[:, 0] = x0
    for i in range(len(t) - 1):
        x[:, i + 1] = x[:, i] + h * f(x[:, i])
    return x, t



# Implicit Euler method

def backward_euler(f, x0, t0, tf, h, tol=1e-8, max_iter=100):
    t = np.arange(t0, tf + h, h)
    x = np.zeros((len(x0), len(t)))
```

```
    x[:, 0] = x0
    for i in range(len(t) - 1):
        x[:, i + 1] = x[:, i]
        for _ in range(max_iter):
            x_next = x[:, i] + h * f(x[:, i + 1])
            if np.linalg.norm(x_next - x[:, i + 1]) < tol:
                break
            x[:, i + 1] = x_next
    return x, t


# Fourth-order Runge-Kutta method

def rk4(f, x0, t0, tf, h):
    t = np.arange(t0, tf + h, h)
    x = np.zeros((len(x0), len(t)))
    x[:, 0] = x0
    for i in range(len(t) - 1):
        k1 = f(x[:, i])
        k2 = f(x[:, i] + 0.5 * h * k1)
        k3 = f(x[:, i] + 0.5 * h * k2)
        k4 = f(x[:, i] + h * k3)
        x[:, i + 1] = x[:, i] + h / 6 * (k1 + 2*k2 + 2*k3 + k4)
    return x, t
```

3.Comparison of analytical solutions:

Since we derive the OED by analytical method :$\ddot{\theta} + \frac{1}{160}\dot{\theta} + (10.32 + 0.24)\theta = 0$

And we generate the graph comparison and compare three methods

```
c_lin = 1 / 160
omega_lin_sq = 10.32 + 0.24
omega_d = np.sqrt(omega_lin_sq - (c_lin / 2)**2)
def analytical(t):
    return theta0 * np.exp(-c_lin * t / 2) * np.cos(omega_d * t)

theta_analytical = analytical(t_rk4)


# Graph comparison

plt.figure(figsize=(20,8))

# θ(t)
plt.subplot(1,2,1)
```

```
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', linewidth=2)
plt.plot(t_rk4, theta_analytical, 'k--', linewidth=2, label='Analytical (linear)')
plt.xlabel('t [s]')
plt.ylabel('θ [rad]')
plt.title('Damped Pendulum θ(t)')
plt.legend()
plt.grid(True)
```

Generate the phase diagram

```
plt.subplot(1,2,2)
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('θ [rad]')
plt.ylabel('θ̇ [rad/s]')
plt.title('Phase Portrait')
plt.legend()
plt.grid(True)

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

4.we get solutions;

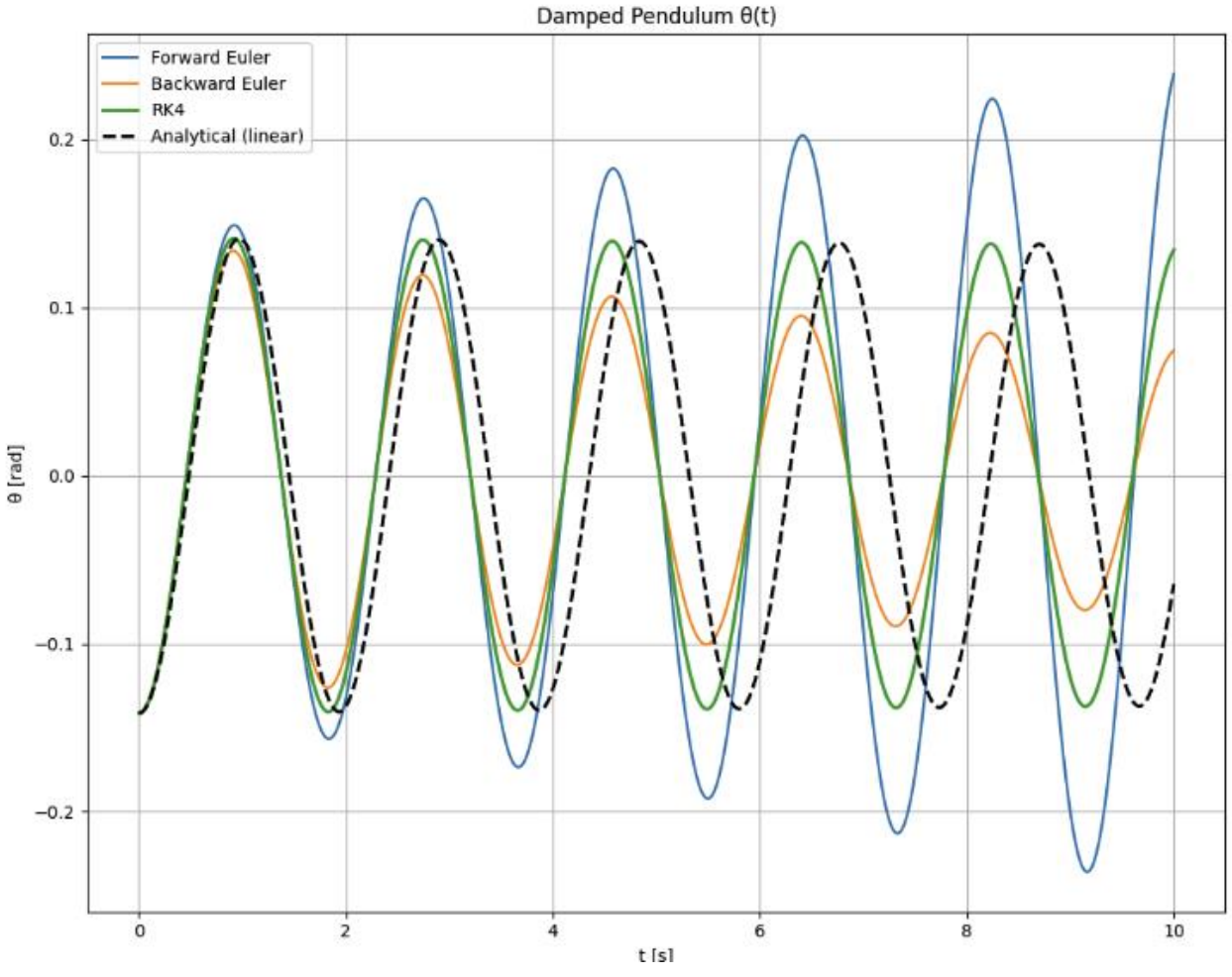
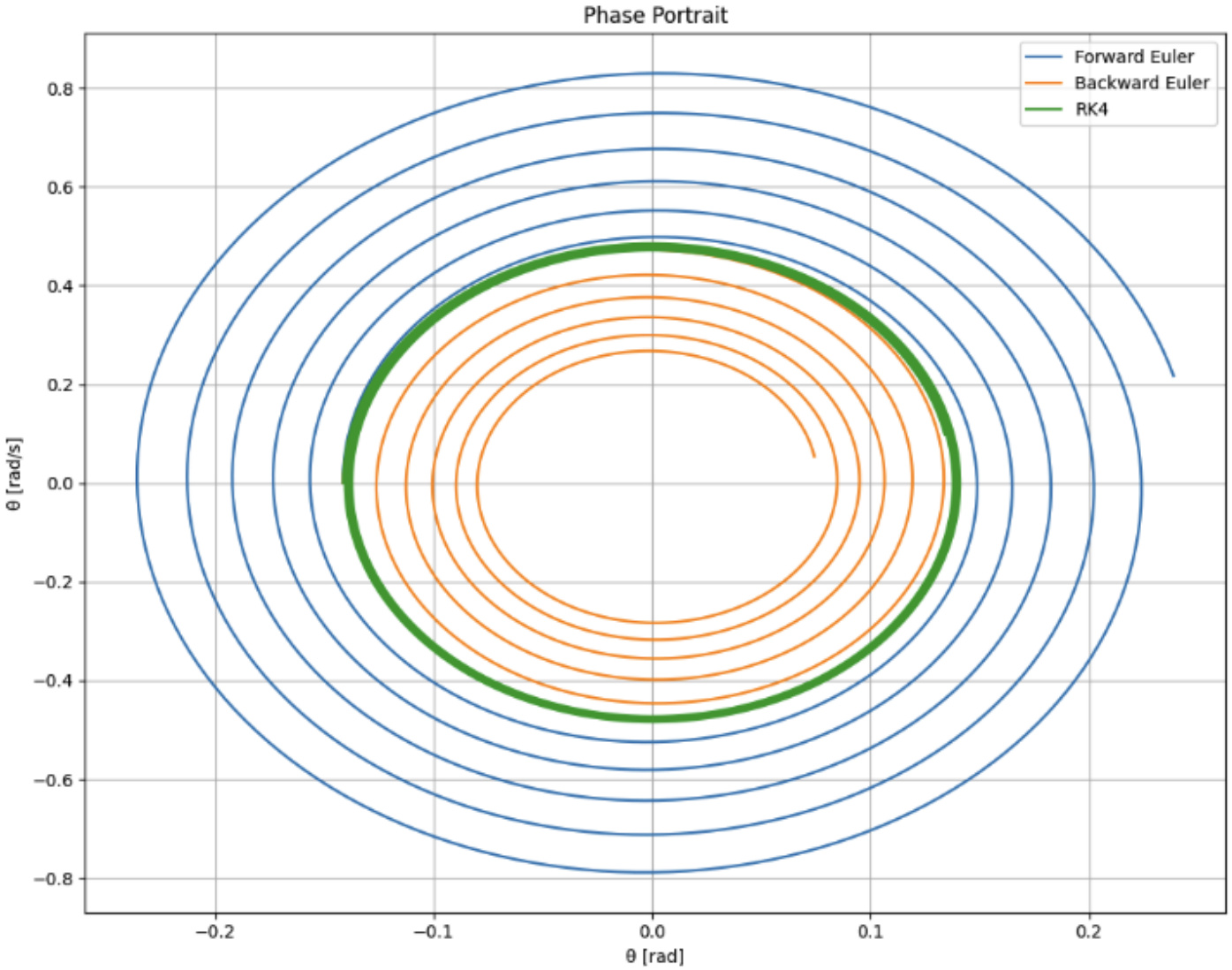


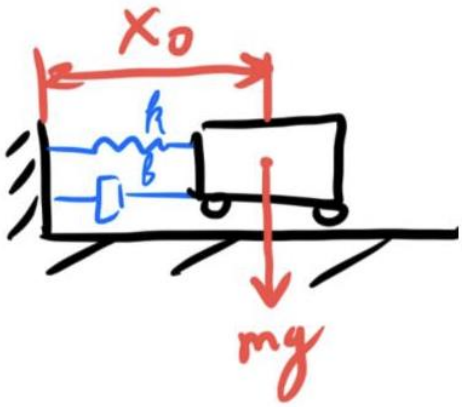
Figure 1



5.Conclutons

In the numerical calculation of this linear damped pendulum system, the Forward Euler method, Backward Euler method, and the RK4 were used to integrate the system's differential equations, and the results were compared with analytical solutions. The results show that different integration algorithms exhibit significant differences in stability and accuracy. The forward Euler method shows that the amplitude gradually diverges over time, with a virtual increase in energy, indicating numerical instability in vibration-related problems. While the backward Euler method is numerically stable, the amplitude decays too quickly, exhibiting obvious virtual energy dissipation behavior, leading to an overestimation

of system damping. In contrast, the curve obtained by the RK4 method closely matches the analytical solution, accurately reflecting the amplitude decay process of damped oscillations while maintaining a reasonable energy contraction trajectory in the phase diagram. Overall, RK4 demonstrates the best comprehensive performance among the three methods, with calculation results closest to the analytical solution, accurately describing the true dynamic behavior of the linear damped pendulum.



Вариант 2

variant 2

Variant2

1. **Handwrite analytical solution:**

   Known parameters:

   M=0.4kg,

   K=9.6N/m

   B=0.01NS/m

   X_0=0,88m

Variant 2.

$$\begin{cases} L(x,\dot{x}) = K(x,\dot{x}) - P(x) = \frac{1}{2}m\dot{x}^2 - \frac{1}{2}kx^2 \\ Q = -b\dot{x} \end{cases}$$

$$\begin{cases} \frac{\partial L}{\partial x} = -kx_0 \\ \frac{\partial L}{\partial \dot{x}} = m\cdot\dot{x} \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) = m\cdot\ddot{x} \end{cases}$$

Combining above equations:

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

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

ODE we get: $m\cdot\ddot{x} + b\dot{x} + kx = 0$

Analytical solution:

Known parameters: $m = 0.4kg$, $K = 9.6N/m$, $b = 0.01Ns/m$, $x_0 = 0.88m$.

Impore parmeters: ~~$\ddot{x} + \frac{b}{m}\dot{x} + \frac{k}{m}x = 0$~~

$$\ddot{x} + \frac{0.01}{0.4}\dot{x} + \frac{9.6}{0.4}x = 0$$

$$\begin{cases} \ddot{x} = r^2 e^{rt} \\ \dot{x} = r e^{rt} \\ x = e^{rt} \end{cases}$$

$$r^2 + 0.025r + 24 = 0$$

$\Delta = b^2 - 4ac = 0.025^2 - 4\times1\times24 = -95.999375 < 0$

Therefore, two complex characteristic roots: $r = \frac{-b \pm \sqrt{\Delta}}{2a} \Rightarrow r_1 = -0.0125 + 4.899i$

$r_2 = -0.0125 - 4.899i$

$$x(t) = e^{-0.0125t}\left(C_1\cos(4.899t) + C_2\sin(4.899t)\right)$$

2. Code Implementation：

First we input know parameters and we can calculate the coefficient of the components of ODE.

Linear damped oscillator: m*x'' + b*x' + k*x = 0

```
def damped_vibration(x):
    """
    Linear damped oscillator: m*x'' + b*x' + k*x = 0
    State vector: x = [position, velocity]
    """
    m = 0.4
    b = 0.01
    k = 9.6

    pos = x[0]
    vel = x[1]

    acc = -(b / m) * vel - (k / m) * pos
    return np.array([vel, acc])
```

Then we can use Explicit Euler ,Implicit Euler and RK4 methods to integrate ODE.

```
# ---------- Numerical Integration Methods ------------------
def forward_euler(fun, x0, Tf, h):
    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):
    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]
        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):
    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
```

3.Comparison of analytical solutions:

Since we derive the OED by analytical method : m*x'' + b*x' + k*x = 0

And we generate the graph comparison and compare three methods

```
# Analytical Solution
def analytical_solution(t):
    m, b, k = 0.4, 0.01, 9.6
    x0, v0 = 0.88, 0.0

    omega_n = np.sqrt(k / m)
    zeta = b / (2 * np.sqrt(k * m))
    omega_d = omega_n * np.sqrt(1 - zeta ** 2)

    # Analytic expression
    A = x0
    B = (v0 + zeta * omega_n * x0) / omega_d

    x_t = np.exp(-zeta * omega_n * t) * (A * np.cos(omega_d * t) + B * np.sin(omega_d *
t))
    v_t = np.gradient(x_t, t)  # numerical derivative for plotting
    return np.vstack((x_t, v_t))


# Run Simulations and Compare
x0 = np.array([0.88, 0.0])
Tf = 5
h = 0.01
```

```
x_fe, t = forward_euler(damped_vibration, x0, Tf, h)
x_be, _ = backward_euler(damped_vibration, x0, Tf, h)
x_rk4, _ = runge_kutta4(damped_vibration, x0, Tf, h)
x_ana = analytical_solution(t)

# -Plotting
plt.figure(figsize=(18, 6))

# (1) Position vs Time
plt.subplot(1, 3, 1)
plt.plot(t, x_ana[0], 'k', linewidth=2, label='Analytical')
plt.plot(t, x_fe[0], '--', label='Forward Euler')
plt.plot(t, x_be[0], '--', label='Backward Euler')
plt.plot(t, x_rk4[0], '--', label='RK4')
plt.xlabel('Time [s]')
plt.ylabel('Displacement [m]')
plt.legend()
plt.title('Displacement vs Time')

# (2) Velocity vs Time
plt.subplot(1, 3, 2)
plt.plot(t, x_ana[1], 'k', linewidth=2, label='Analytical')
plt.plot(t, x_fe[1], '--', label='Forward Euler')
plt.plot(t, x_be[1], '--', label='Backward Euler')
plt.plot(t, x_rk4[1], '--', label='RK4')
plt.xlabel('Time [s]')
plt.ylabel('Velocity [m/s]')
plt.legend()
plt.title('Velocity vs Time')

# (3) Phase Plot
plt.subplot(1, 3, 3)
plt.plot(x_ana[0], x_ana[1], 'k', linewidth=2, label='Analytical')
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('Displacement [m]')
plt.ylabel('Velocity [m/s]')
plt.legend()
plt.title('Phase Portrait')

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

4.we get solutions;

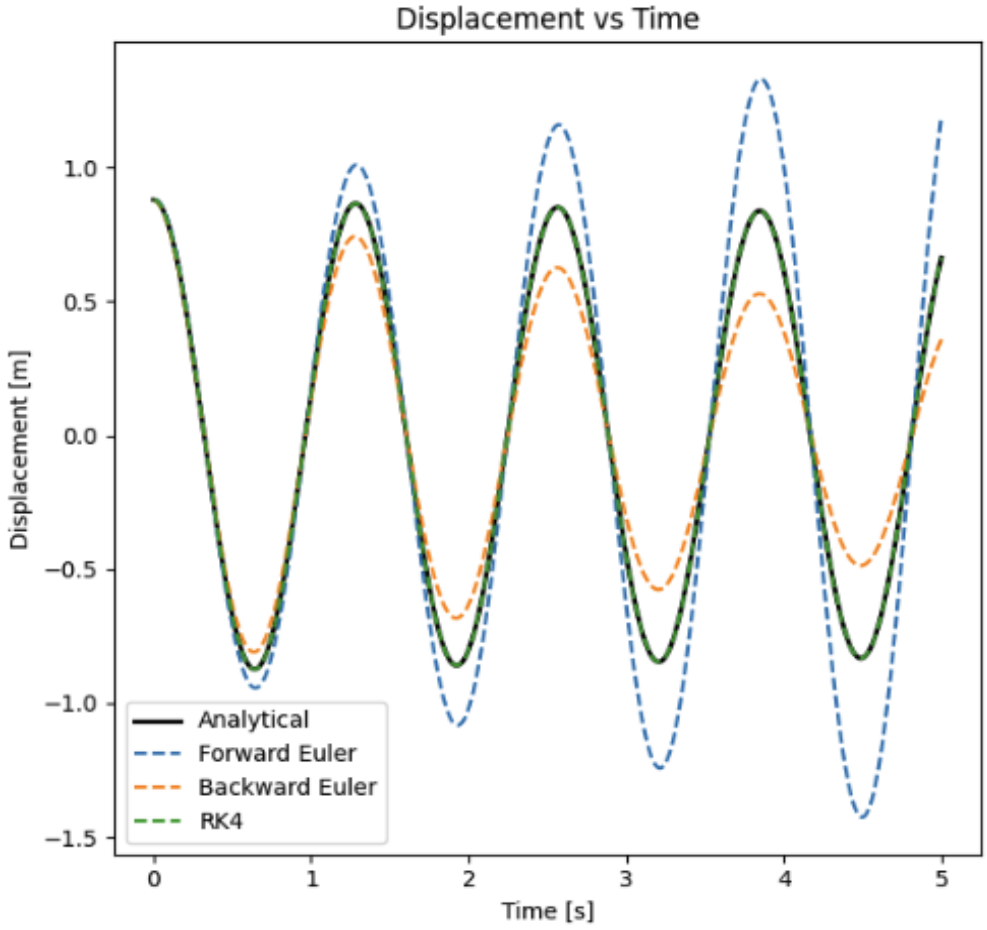


Figure1Displacement vs tiome

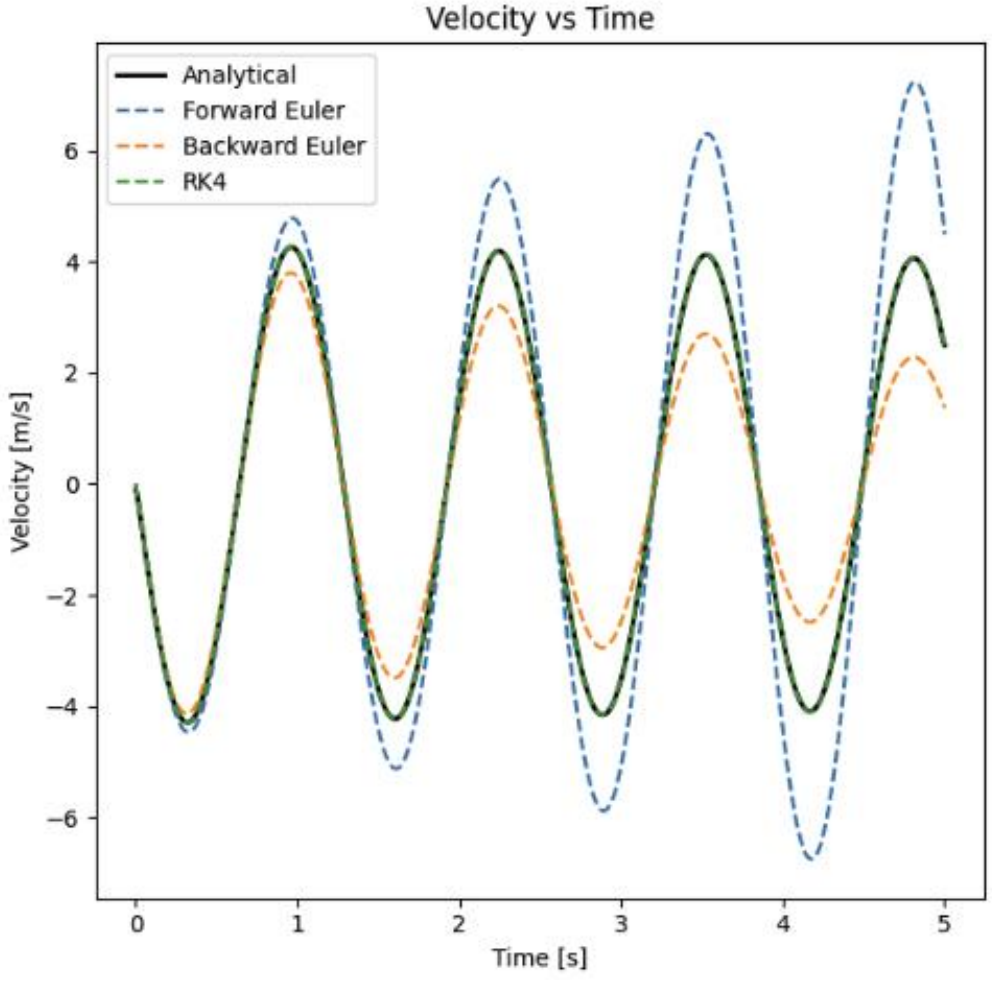


Figure 2 Velocity vs Time

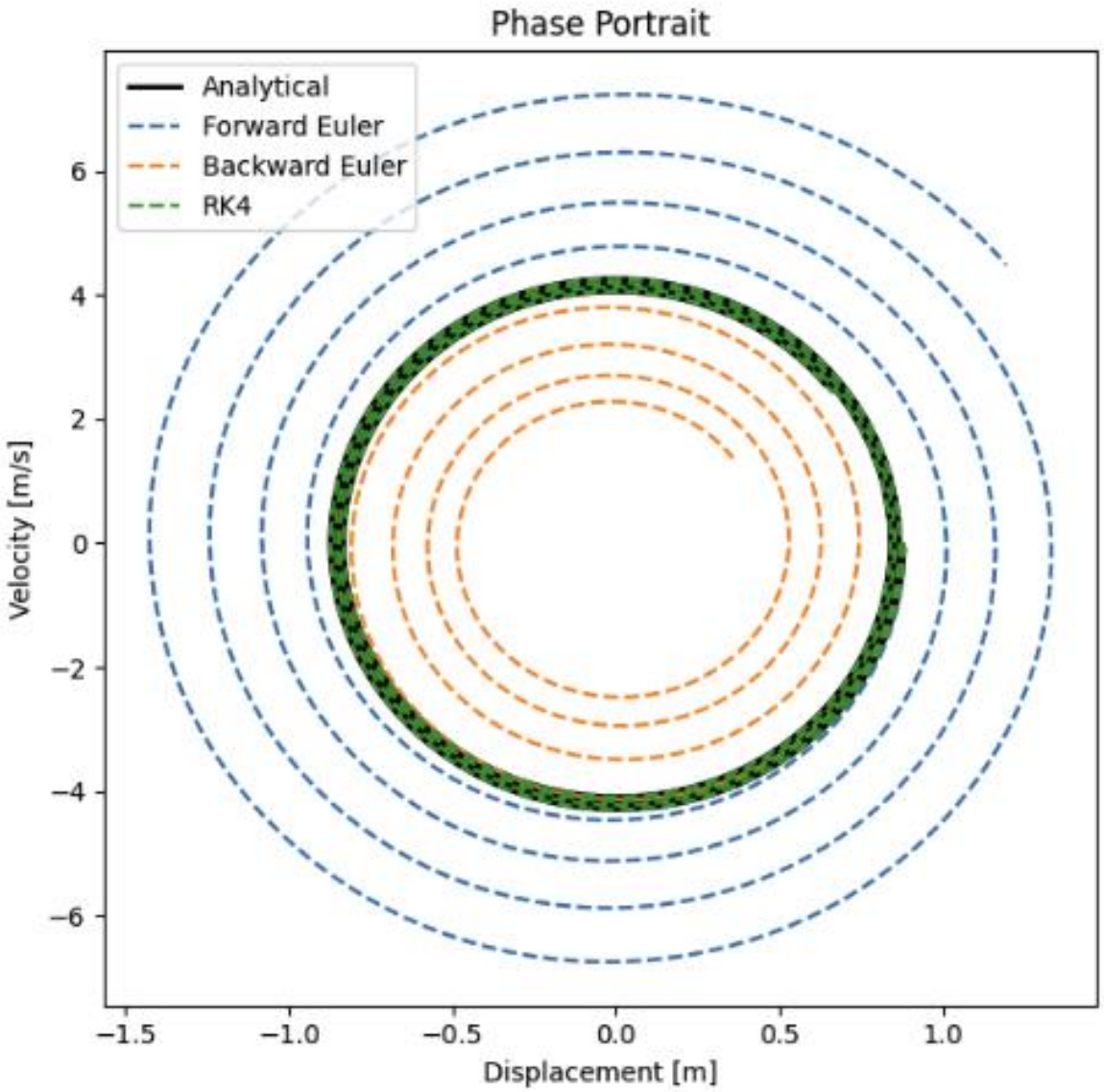


Figure 3 Phase Portrait

5.Conclution

Numerical simulations of the damped vibration equation m x ¨ + b x ˙ + k x = 0 show significant differences in the accuracy and stability of different integration methods in describing the system's dynamic behavior. The analytical solution reveals time-decreasing periodic vibrations, serving as a benchmark for evaluating the correctness of numerical methods. Comparison of the three integration methods reveals that: the forward Euler method is computationally simple but its numerical energy continuously increases, leading to divergence in displacement and velocity amplitudes over time and poor stability; the

backward Euler method, while unconditionally stable, excessively suppresses oscillations, causing the system to exhibit unrealistically strong damped characteristics; the fourth-order Runge-Kutta method offers the best overall performance, with its calculation results almost perfectly matching the analytical solution, accurately reflecting amplitude decay and phase changes.

In summary, the RK4 method performs best in terms of accuracy, stability, and physical consistency, making it the recommended numerical method for solving such vibration systems.