ИТМО

Second Task for SRS

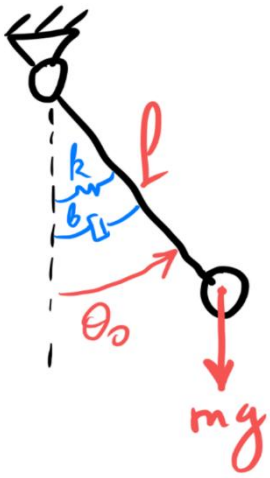
Salam Ali

503263

1.Dfferntial equation:

Kinetic energy:

The pendulum rotates about the pivot, so its kinetic energy is purely rotational:



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

Potential energy:

Potential energy has two contributions:

Gravity: $V_g = mgl(1-\cos\theta)$

Spring: $V_s = \frac{1}{2}k\theta^2$ then total potential energy will be

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

Lagrangian:

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

Lagrange equation:

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

Derivatives:

$$\frac{\partial L}{\partial \dot{\theta}} = ml^2\dot{\theta} \Rightarrow \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} = ml^2\ddot{\theta}$$

$$\frac{\partial L}{\partial \theta} = -\left(k\theta + mgl\sin\theta\right)$$

Also, $Q = -b\dot{\theta}$.

Final differential equation:

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

2.Analytical Solution:

The differential equation that we have is not linear due to the term $mgl\sin\theta$.So, it is difficult to solve it analytically. To obtain an approximate solution, we apply the small-angle approximation $\sin\theta \approx \theta$.

New linear equation:

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

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

So, we have:

$$\omega_n = \sqrt{\frac{k + mgl}{ml^2}}$$

$$\xi = \frac{b}{2\sqrt{ml^2(k + mgl)}}$$

Parameters:

$m = 0.4kg, l = 0.9m, k = 18.4Nm/rad, b = 0.02Nm.s/rad, g = 9.81m/s^2.$

Numerically:

$$\omega_n \simeq 8.2274$$

$$\zeta \simeq 0.00375$$

$$\omega_d = \omega_n\sqrt{1-\zeta^2} \simeq 8.2273$$

$$\zeta\omega_n = 0.0308$$

After substitution the initial conditions: we get the final solution:

$$\theta(t) = e^{-\zeta\omega_n t}\left(A\cos(\omega_d t) + B\sin(\omega_d t)\right)$$

We have $\theta_0 = -1.068174678, \dot{\theta}_0 = 0$, which gives:

$$\theta(t) = e^{-0.0308641975t}\left(-1.068174678\cos(8.2273428796t) - 0.00400716911\sin(8.2273428796t)\right)$$

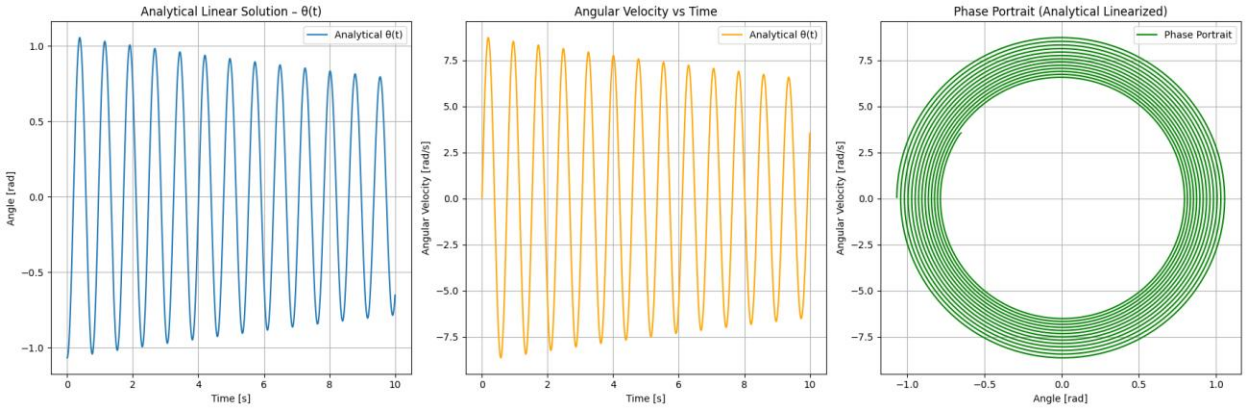
Note:

Despite the large initial angle, the analytical solution can be used. This is because the stiff spring (k=18.4) dominates the system's dynamics. So, we can use this approximation to compare between analytical and numerical methods and see how much this method is useful in this case (big k).

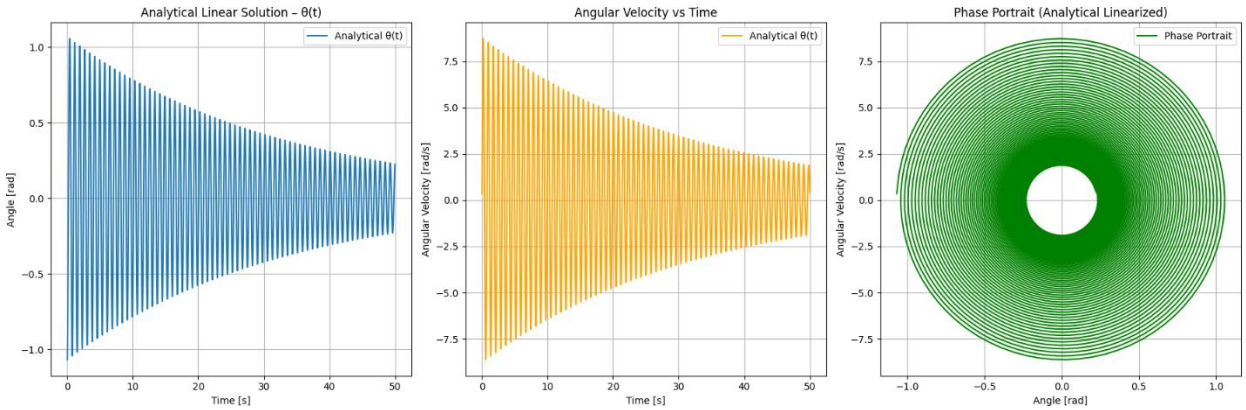
3.Simulation and discussion:

3.1Aalytical solution with approximation:

For simulation time 10 seconds



For simulation time 50 seconds



1.Angle:

The analytical solution for this system predicts that the angle oscillates around the equilibrium position with a gradually decreasing amplitude.

The oscillations follow a cosine-like waveform multiplied by an exponential decay term, which causes the envelope of the motion to shrink over time.

2.Velocity:

The angular velocity also oscillates with time, but it is out of phase with the angle. When the angle passes through zero, the velocity reaches its maximum value.

The amplitude of the velocity decays at the same rate as the angle, following the same exponential envelope.

3.phase:

The analytical phase portrait of the system forms a spiral pattern that moves inward toward the origin.

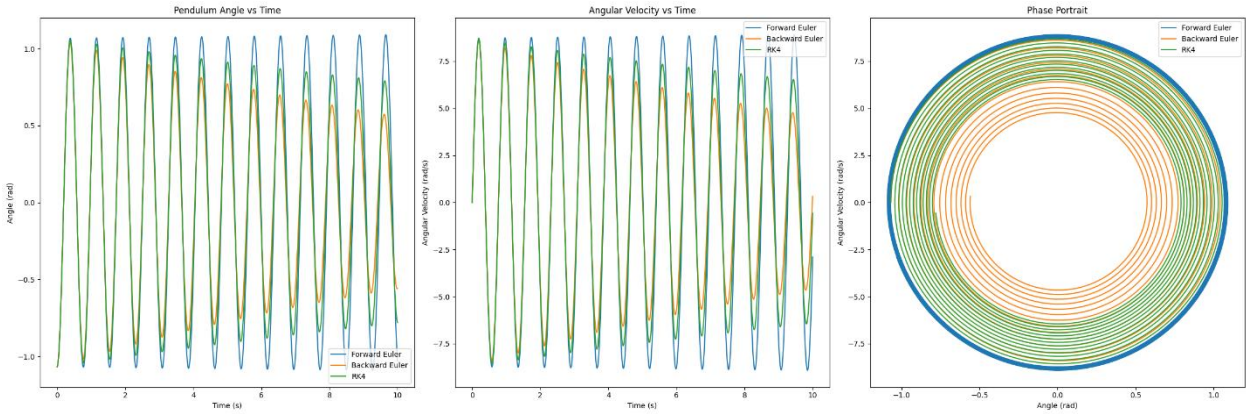
Each loop represents one complete oscillation, and the spiral's tightening shows the gradual loss of mechanical energy due to damping.

Note:

The damping is small, so the oscillations decay slowly. In short simulations, this gradual decay isn't clear. By extending the simulation time, the amplitude reduction and the phase spiral become visible, showing how the pendulum slowly settles at equilibrium.

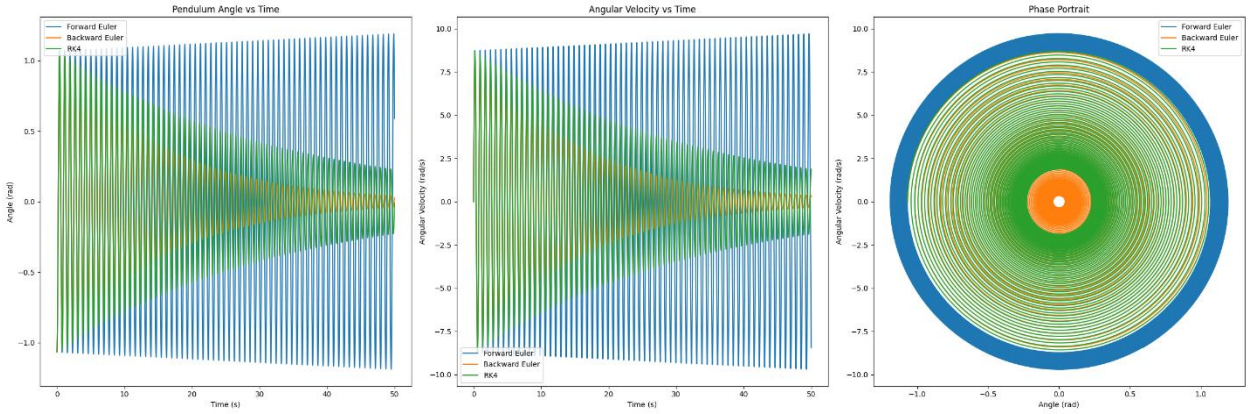
3.2 Numerical solution:

The simulation results for step 0.001



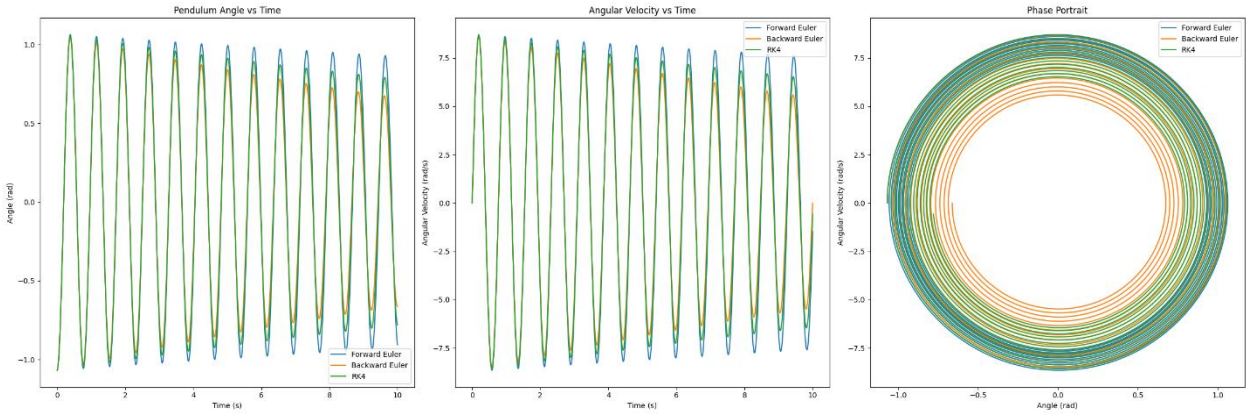
In our system, the damping is small and the spring is relatively stiff, giving the pendulum a high natural frequency. Forward Euler is an explicit method, meaning it estimates the next state using only the slope at the current step. For oscillatory systems with high frequency, even a seemingly small-time step of 0.001 seconds can be too large relative to the fastest dynamics, causing numerical errors to accumulate. These errors amplify over time, which makes the Forward Euler solution spread out and diverge, even though the physical system is lightly damped and should slowly converge. Backward Euler, being implicit, and RK4, with higher-order accuracy, do not suffer from this instability, which is why they remain smooth and convergent.

The phase portrait clearly illustrates how each method handles damping: RK4 is most faithful, Backward Euler is stable but slightly overdamped, and Forward Euler lags behind.



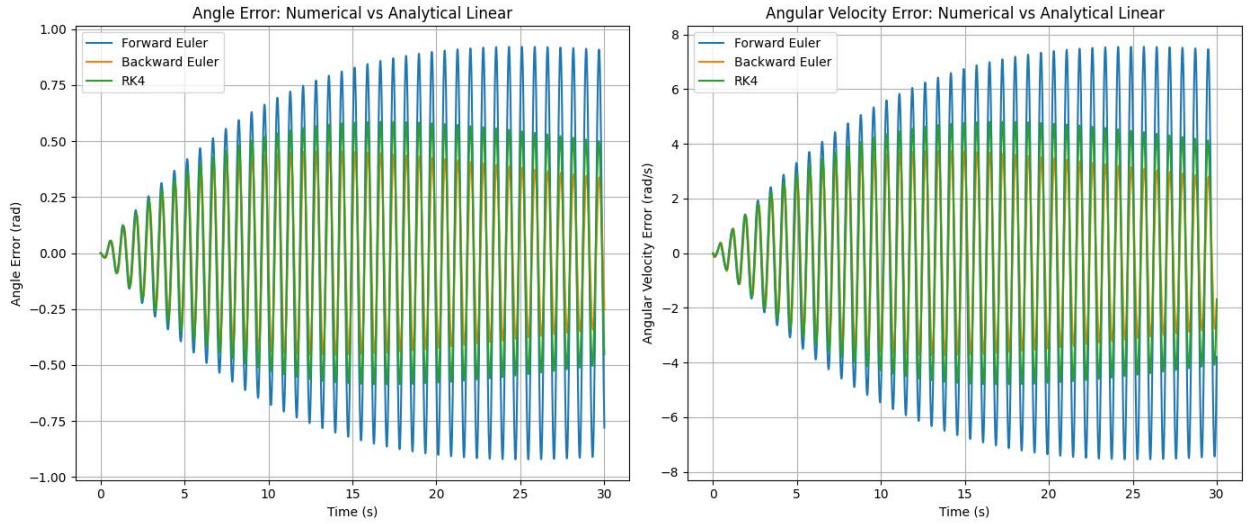
The second image shows how Forward Euler converges over time. We use bigger simulation time to see clearly this thing.

We can make Forward Euler efficient by making the step smaller like 0.0005 and the results will be as following:



3.3 Comparison between analytical and numerical:

To compare between the anlaytical and numerical solutions (h=0.0005) we draw the error function between each numeical solution and analytical one.



Despite the large initial angle, we initially hypothesized that the stiff spring (k=18.4) might dominate the system's dynamics, making the small-angle approximation usable. However, as shown in Section 3.3, this approximation proves invalid for large angles even with substantial k.

4.Appendix (code):

```
import numpy as np
import matplotlib.pyplot as plt
def pendulum_dynamics(x):
    """
    Pendulum dynamics with rotational spring and damper:
    m*L^2*θ_ddot + b*θ_dot + k*θ + m*g*L*sin(θ) = 0
    State vector x = [θ, θ_dot]
    """
    m = 0.4       # kg
    L = 0.9       # m
    g = 9.81      # m/s^2
    k = 18.4      # Nm/rad
    b = 0.02      # Nm*s/rad
    theta, theta_dot = x
    theta_ddot = -(b*theta_dot + k*theta + m*g*L*np.sin(theta)) / (m*L**2)
    return np.array([theta_dot, theta_ddot])
def analytical_solution(t):
    m = 0.4
    L = 0.9
    g = 9.81
    k = 18.4
    b = 0.02
    I = m*L**2
    mgl = m*g*L
    theta0 = -1.068174678
    theta_dot0 = 0.0
    wn = np.sqrt((k + mgl)/I)
    zeta = b/(2*np.sqrt(I*(k + mgl)))
    if zeta < 1:  # underdamped
```

```
        wd = wn * np.sqrt(1 - zeta**2)
        A = theta0
        B = (theta_dot0 + zeta*wn*theta0)/wd
        theta = np.exp(-zeta*wn*t) * (A*np.cos(wd*t) + B*np.sin(wd*t))
    elif zeta == 1:  # critically damped
        C1 = theta0
        C2 = theta_dot0 + wn*theta0
        theta = (C1 + C2*t) * np.exp(-wn*t)
    else:  # overdamped
        r1 = -wn*(zeta - np.sqrt(zeta**2 - 1))
        r2 = -wn*(zeta + np.sqrt(zeta**2 - 1))
        C2 = (theta_dot0 - r1*theta0)/(r2 - r1)
        C1 = theta0 - C2
        theta = C1*np.exp(r1*t) + C2*np.exp(r2*t)
    theta_dot = np.gradient(theta, t[1]-t[0])
    return np.vstack((theta, theta_dot))
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])
            if np.linalg.norm(x_next - x_hist[:,k+1]) < tol:
                break
            x_hist[:,k+1] = x_next
    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)*(k1 + 2*k2 + 2*k3 + k4)
    return x_hist, t
x0 = np.array([-1.068174678, 0.0])
Tf = 10.0
h = 0.0005
x_fe, t_fe = forward_euler(pendulum_dynamics, x0, Tf, h)
x_be, t_be = backward_euler(pendulum_dynamics, x0, Tf, h)
x_rk4, t_rk4 = runge_kutta4(pendulum_dynamics, x0, Tf, h)
x_lin = analytical_solution(t_rk4)
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_lin[0,:], '--', label='Analytical Linear')
plt.xlabel('Time (s)')
```

```
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.plot(t_rk4, x_lin[1,:], '--', label='Analytical Linear')
plt.xlabel('Time (s)')
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.plot(x_lin[0,:], x_lin[1,:], '--', label='Analytical Linear')
plt.xlabel('Angle (rad)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Phase Portrait')
plt.tight_layout()
plt.show()
```