ИТМО

First Task for SRS

Salam Ali

503263

1.Analytival Solution:

Differential equation:

$$a\ddot{x} + b\dot{x} + cx = d$$

The characteristic equation:

$$ar^2 + br + c = 0$$

The discriminant Δ calculated by the equation:

$$\Delta = b^2 - 4ac$$

According to the discriminant we have three cases:

Case1: two distinct real roots $(\Delta > 0)$:

$$r_1 = \frac{-b + \sqrt{\Delta}}{2a}, r_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

$$x_h(t) = C_1 e^{r_1 t} + C_2 e^{r_2 t}$$

Case2: repeated real roots $(\Delta = 0)$:

$$r_1 = r_2 = r = \frac{-b}{2a}$$

$$x_h(t) = (C_1 + C_2 t)e^{rt}$$

Case3: complex conjugate roots $(\Delta < 0)$:

$$r_{1,2} = \alpha \pm \beta i \text{ where } \alpha = -\frac{b}{2a}, \beta = \frac{\sqrt{-\Delta}}{2a}$$

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

Now we will find the particular solution x_p for the non-homogeneous part d assuming that $d \neq 0$:

$$a(0) + b(0) + cx_p = d \rightarrow x_p = \frac{d}{c}$$

1.1Solving differential equation for given values:

$$a = -8.69, b = -7.7, c = -5.57, d = -4.85$$

$$\ddot{x} + \frac{7.7}{8.69}\dot{x} + \frac{5.57}{8.69}x = \frac{4.85}{8.69}$$

$$\ddot{x} + 0.886\dot{x} + 0.641x = 0.558$$

The homogeneous solution:

$$\ddot{x}_h + 0.886\dot{x}_h + 0.641x_h = 0$$

$$\Delta = 0.886^2 - 4 \times 0.641 = -1.779$$

Since $\Delta < 0$, the roots are complex, which indicates the system is underdamped.

$$r_{1,2} = \frac{-0.86 \pm \sqrt{-1.779}}{2} = -0.443 \pm 0.667i$$

The homogeneous solution:

$$x_h(t) = e^{-0.443t}(C_1 \cos(0.667t) + C_2 \sin(0.667t))$$

The particular solution:

$$x_p = \frac{0.558}{0.641} \approx 0.87$$

Final solution:

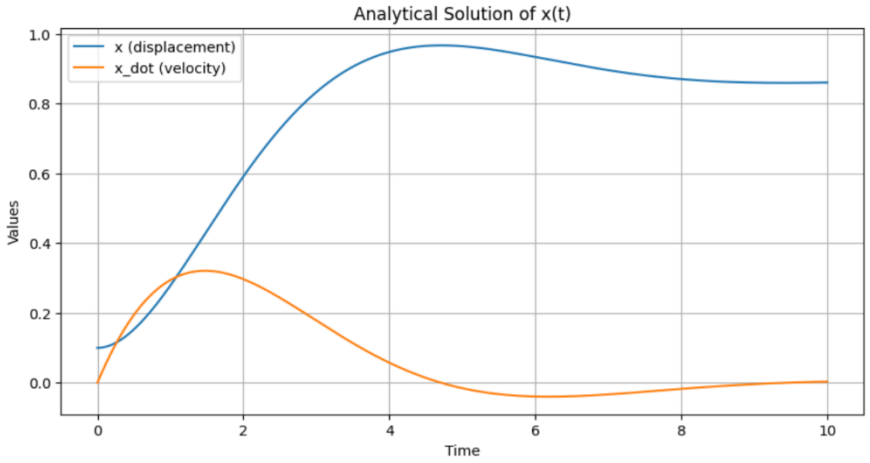
$$x(t) = e^{-0.443t}(C_1 \cos(0.667t) + C_2 \sin(0.667t)) + 0.87$$

After substitution the initial conditions $x(0) = 0.1, \dot{x}(0) = 0$ we get:

$$x(t) = 0.87 + e^{-0.443t}(-0.7743\cos(0.667t) - 0.5514\sin(0.667t))$$

$$\dot{x}(t) = e^{-0.443t}(0.3439\cos(0.667t) + 0.5156\sin(0.667t))$$

Analytical solution after executing code



The obtained analytical solutions show that the system is underdamped, since the characteristic equation has complex conjugate roots.

This means the response exhibits oscillations that gradually decay over time due to the exponential term $e^{-0.443t}$.

- The displacement $x(t)$ starts from the initial value $x(0) = 0.1$ and oscillates around the steady-state value $x_{steady} = 0.8743$.
- The exponential term represents energy dissipation (damping), causing the amplitude to decrease as time increases.
- Velocity $\dot{x}(t)$ also oscillates with the same damped frequency (0.667) but is phase-shifted relative to $x(t)$.
- Both $x(t)$ and $\dot{x}(t)$ eventually converge to their steady-state values:

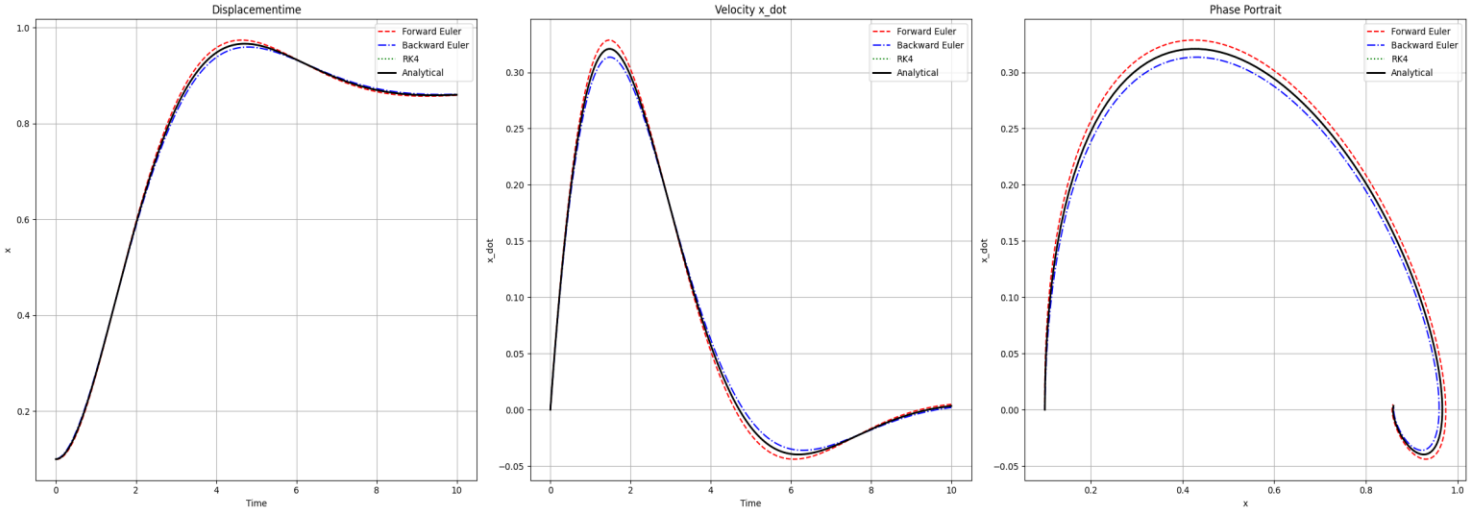
$$\lim_{t\to\infty} x(t) = 0.8743, \lim_{t\to\infty} \dot{x}(t) = 0$$

2.Numerical Solution:

After solving the equation analytically, we can also find the solution using numerical methods. These methods calculate the system response step by step in time.

The numerical solution gives the displacement $x(t)$ and velocity $\dot{x}(t)$ over the time interval. We can also look at the phase portrait (displacement in terms of velocity) to see how the system oscillates and gradually reaches the steady state.

After executing our code, we get:



3.Comparison of Numerical and Analytical Solutions:

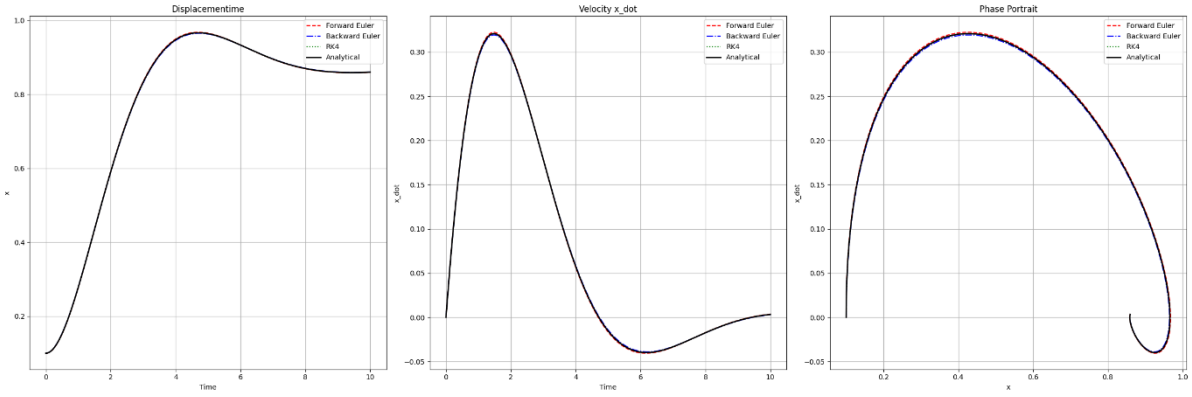
The system was solved analytically and numerically using Forward Euler, Backward Euler, and 4th-order Runge-Kutta (RK4) methods.

- Displacement:
  All numerical solutions follow the analytical curve very closely. RK4 almost exactly matches the analytical solution. Forward Euler and Backward Euler are also very close, with only small shifts over time.
- Velocity:
  The velocity from all numerical methods is very similar to the analytical solution. Small differences appear in Forward Euler and Backward Euler, but the main trend is captured well.
- Phase Portrait:
  All numerical methods reproduce the system motion accurately. RK4 matches the analytical loop almost perfectly. Forward and Backward Euler show very small offsets, which do not change the overall behavior.

4.Notes:

1.All numerical methods—Forward Euler, Backward Euler, and RK4—closely follow the analytical solution. Initially, with a very small-time step $(h = 0.01)$, all curves almost overlapped, making it hard to see any differences between the methods. To make the differences visible, the time step was increased to $(h = 0.05)$. This allowed small deviations of Forward Euler and Backward Euler from the analytical solution to appear, while RK4 still closely matched the analytical curve.

Overall, RK4 remains the most accurate and reliable method, showing minimal deviation from the analytical solution. Forward and Backward Euler are reasonably close, but small offsets appear due to their numerical approximations. Using a slightly larger time step makes these differences clear while keeping all methods stable and correct.



2.Although $\Delta < 0$, indicating an underdamped oscillation, the damping factor 0.443 is relatively large. This causes the exponential decay $e^{-0.443t}$ to reduce the amplitude very quickly. As a result, the oscillations die out almost immediately, and the plot appears as a smooth curve approaching the steady-state value.

5.Appendex(code):

```
import numpy as np

import matplotlib.pyplot as plt

# analytical solutio

def analytical_solution(t, a, b, c, d, x0, tol=1e-12):

    x_init, x_dot_init = x0

    alpha = b / a

    beta  = c / a

    x_steady = d / c

    discriminant = alpha**2 - 4 * beta

    t = np.asarray(t)


    if discriminant > -tol:

        if abs(discriminant) < tol:

            discriminant = 0.0

        sqrt_d = np.sqrt(discriminant)

        r1 = (-alpha + sqrt_d)/2

        r2 = (-alpha - sqrt_d)/2

        if abs(r1 - r2) < tol:

            r = r1

            A = x_init - x_steady

            B = x_dot_init - r*A

            exp_rt = np.exp(r*t)

            x_t = x_steady + (A + B*t)*exp_rt

            x_dot_t = exp_rt*(B + r*(A + B*t))

        else:

            A = (x_dot_init - r2*(x_init - x_steady)) / (r1 - r2)

            B = (x_init - x_steady) - A

            x_t = x_steady + A*np.exp(r1*t) + B*np.exp(r2*t)
```

```
            x_dot_t = A*r1*np.exp(r1*t) + B*r2*np.exp(r2*t)
    else:
        real = -alpha/2
        imag = np.sqrt(-discriminant)/2
        C = x_init - x_steady
        D = (x_dot_init - real*C)/imag
        exp_rt = np.exp(real*t)
        x_t = x_steady + exp_rt*(C*np.cos(imag*t) + D*np.sin(imag*t))
        x_dot_t = exp_rt*((D*imag + C*real)*np.cos(imag*t) + (D*real - C*imag)*np.sin(imag*t))
    return np.vstack([x_t, x_dot_t])


def ODE_solution(x):
    a, b, c, d = -8.69, -7.7, -5.57, -4.85
    x_val, x_dot_val = x
    x_ddot = (d/a) - (b/a)*x_dot_val - (c/a)*x_val
    return np.array([x_dot_val, x_ddot])


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)*(k1 + 2*k2 + 2*k3 + k4)
    return x_hist, t


x0, Tf, h = np.array([0.1, 0.0]), 10.0, 0.05
t_values = np.arange(0, Tf+h, h)


x_fe, t_fe = forward_euler(ODE_solution, x0, Tf, h)
x_be, t_be = backward_euler(ODE_solution, x0, Tf, h)
x_rk4, t_rk4 = runge_kutta4(ODE_solution, x0, Tf, h)
x_analytical = analytical_solution(t_values, -8.69, -7.7, -5.57, -4.85, x0)
```

```
step_fe = 1

step_be = 1

step_rk4 = 1

step_analytical = 1


#  ploting

plt.figure(figsize=(24,8))




plt.subplot(1,3,1)

plt.plot(t_fe[::step_fe], x_fe[0,::step_fe], 'r--', label='Forward Euler')

plt.plot(t_be[::step_be], x_be[0,::step_be], 'b-.', label='Backward Euler')

plt.plot(t_rk4[::step_rk4], x_rk4[0,::step_rk4], 'g:', label='RK4')

plt.plot(t_values[::step_analytical], x_analytical[0,::step_analytical], 'k', linewidth=2, label='Analytical')

plt.xlabel('Time')

plt.ylabel('x')

plt.title('Displacementime')

plt.legend()

plt.grid(True)

# Velocity

plt.subplot(1,3,2)

plt.plot(t_fe[::step_fe], x_fe[1,::step_fe], 'r--', label='Forward Euler')

plt.plot(t_be[::step_be], x_be[1,::step_be], 'b-.', label='Backward Euler')

plt.plot(t_rk4[::step_rk4], x_rk4[1,::step_rk4], 'g:', label='RK4')

plt.plot(t_values[::step_analytical], x_analytical[1,::step_analytical], 'k', linewidth=2, label='Analytical')

plt.xlabel('Time')

plt.ylabel('x_dot')
```

```
plt.title('Velocity x_dot')

plt.legend()

plt.grid(True)

# Phase portrait

plt.subplot(1,3,3)

plt.plot(x_fe[0,::step_fe], x_fe[1,::step_fe], 'r--', label='Forward Euler')

plt.plot(x_be[0,::step_be], x_be[1,::step_be], 'b-.', label='Backward Euler')

plt.plot(x_rk4[0,::step_rk4], x_rk4[1,::step_rk4], 'g:', label='RK4')

plt.plot(x_analytical[0,::step_analytical],     x_analytical[1,::step_analytical],     'k',     linewidth=2,
label='Analytical')

plt.xlabel('x')

plt.ylabel('x_dot')

plt.title('Phase Portrait')

plt.legend()

plt.grid(True)



plt.tight_layout()

plt.show()
```