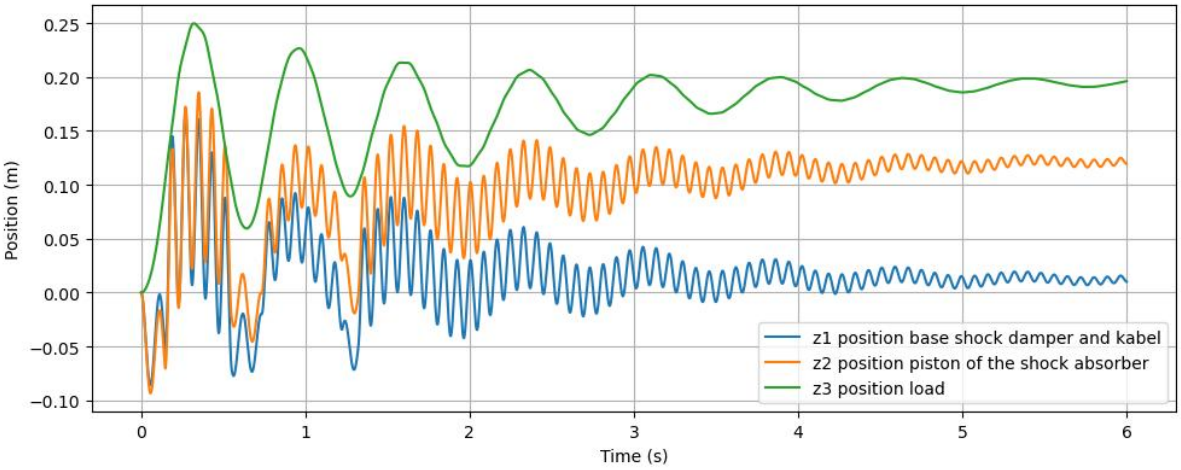
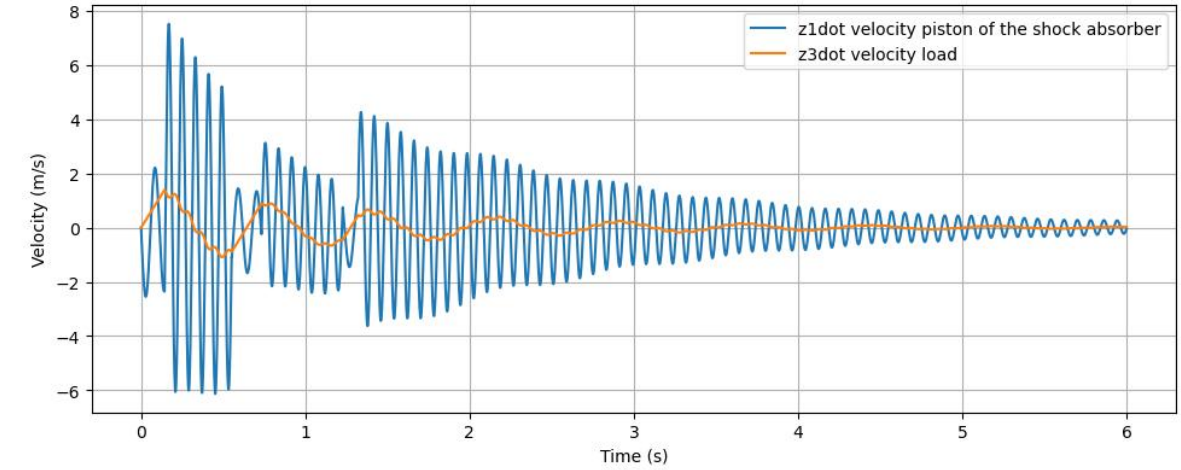
In [18]:

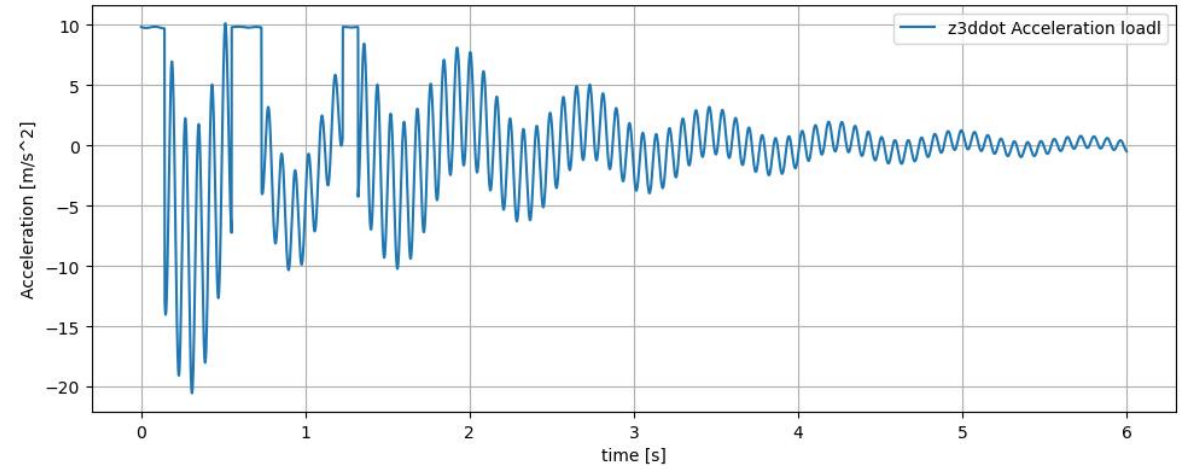
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# Properties
m1 = 25000  # Mass of the cable and base (kg)
m2 = 15000  # Mass of the piston and the rod (kg)
m3 = 850000  # Mass of the load (kg)
k1 = 2 * 5.7e7  # Spring constant for the cable (N/m)
k3 = 2 * 5.7e7  # Spring constant for the cable (N/m)
k2 = 9.661e6  # Spring constant for the gas stiffness (N/m)
c1 = 22590  # Damping coefficient for the cable (Ns/m)
c2 = 380e5  # Damping coefficient for the piston (Ns/m)
c3 = 22590  # Damping coefficient for the piston (Ns/m)
g = 9.81  # Acceleration due to gravity (m/s^2)
N = 2  # Number of cylinders

# Define damper and cylinder
V = 0.864  # Volume constant (m^3)
Dpiston = 500 * 1 / 1000.0  # Diameter of the piston in meters
Drod = 200 * 1 / 1000.0  # Diameter of the rod in meters
Apiston = 0.25 * np.pi * Dpiston ** 2  # Area of the piston (m^2)
Aannu = max(0.25 * np.pi * (Dpiston ** 2 - Drod ** 2), 1e-5)  # Area of the annular
Aannu2 = Aannu * N  # Total annular area for N cylinders (m^2)
initial_pressure_p1_annular = 180e5  # Initial pressure in the annular space (Pa)
Qleak = 5.199e-5  # Leak constant (m^3/s)
Bgas = 9.097 * 10 ** 7  # Gas modulus

# Slack cable parameters
z_slack = 0.1  # The displacement at which the cable becomes taut


def system_function(t, z):
    global m1, m2, m3, k1, k2, k3, c1, c2, c3, g, N, V, Aannu2, Qleak, Bgas, z_slac

    # Piecewise Linear Model for k3 and c3
    # Effective spring constant for k3
    k3_effective = 2 * 5.7e4 if z[4] <= z_slack else 2 * 5.7e7

    # Effective damping coefficient for c3
    c3_effective = 22590 * 0.7 if z[5] <= 0 else c3

    # Equations of Motion
    zdot = np.zeros_like(z)

    zdot[0] = z[1]  # speed
    zdot[1] = (-k1 * z[0] - c1 * z[1] + k2 * (z[2] - z[0]) + c2 * (z[3] - z[1]) + m
    zdot[2] = z[3]
    zdot[3] = (k2 * (z[0] - z[2]) + c2 * (z[1] - z[3]) - z[6] * Aannu2 + k3_effecti
               c3_effective * (z[5] - z[3]) + m2 * g) / m2
    zdot[4] = z[5]
    zdot[5] = (k3_effective * (z[2] - z[4]) + c3_effective * (z[3] - z[5]) + m3 * g
    zdot[6] = (Bgas / V) * (Aannu2 * z[3] - Qleak)  # Equation of motion load

    return zdot


# Initial conditions
initial_conditions = [0, 0, 0, 0, 0, 0, 180e5]

endtime = 6
```

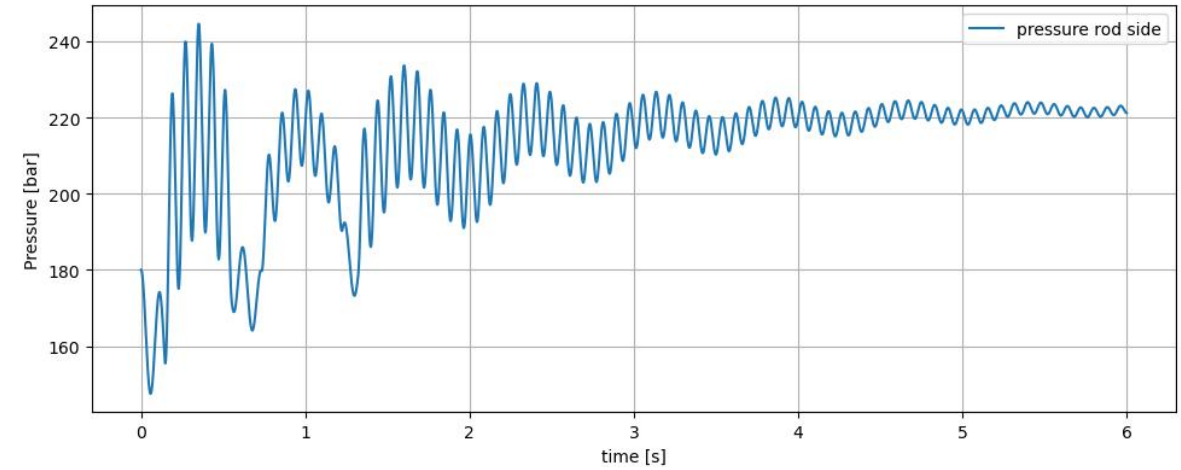
```
# Time span to integrate over
t_span = [0, endtime]

# Solving the equation using solve_ivp
solution = solve_ivp(system_function, t_span, initial_conditions, method='RK45', t_

# Extracting the solution
z_solution = solution.y.T

# Plotting
plt.figure(figsize=(10, 20))

plt.subplot(5, 1, 1)
plt.plot(solution.t, z_solution[:, 0], label='z1 position base shock damper and kab
plt.plot(solution.t, z_solution[:, 2], label='z2 position piston of the shock absor
plt.plot(solution.t, z_solution[:, 4], label='z3 position load')
plt.xlabel('Time (s)')
plt.ylabel('Position (m)')
plt.legend()
plt.grid(True)


plt.subplot(5, 1, 2)
#plt.plot(solution.t, z_solution[:, 1], label='z1ddot velocity base shock damper ar
plt.plot(solution.t, z_solution[:, 3], label='z1dot velocity piston of the shock ab
plt.plot(solution.t, z_solution[:, 5], label='z3dot velocity load')
plt.xlabel('Time (s)')
plt.ylabel('Velocity (m/s)')
plt.legend()
plt.grid(True)


plt.subplot(5, 1, 3)
# Add acceleration calculations for z1, z3, and z5
z1_acceleration = np.diff(z_solution[:, 1]) / np.diff(solution.t)
z3_acceleration = np.diff(z_solution[:, 3]) / np.diff(solution.t)
z5_acceleration = np.diff(z_solution[:, 5]) / np.diff(solution.t)
t_acceleration = (solution.t[:-1] + solution.t[1:]) / 2  # Adjust time vector for a

#plt.plot(t_acceleration, z1_acceleration, label='z1ddot Acceleration base shock da
#plt.plot(t_acceleration, z3_acceleration, label='z2ddot Acceleration piston')
plt.plot(t_acceleration, z5_acceleration, label='z3ddot Acceleration loadl')
plt.xlabel('time [s]')
plt.ylabel('Acceleration [m/s^2]')
plt.grid(True)
plt.legend()

plt.subplot(5, 1, 4)
plt.plot(solution.t, z_solution[:, 6] * 10 ** -5, label='pressure rod side')
plt.xlabel('time [s]')
plt.ylabel('Pressure [bar]')
plt.grid(True)
plt.legend()


plt.subplot(5, 1, 5)
# Calculate dynamic forces using mass and acceleration values
dynamicForce_z2 = m2 * z5_acceleration * 0.001 + 8500
steadyStateForce = 8500 * np.ones_like(t_acceleration)  # Steady-state force line

plt.plot(t_acceleration, steadyStateForce, label='Steady State Force (8500 kN)')
plt.plot(t_acceleration, dynamicForce_z2, label='Dynamic Force z2')
plt.xlabel('time [s]')
plt.ylabel('Dynamic Force [kN]')
```

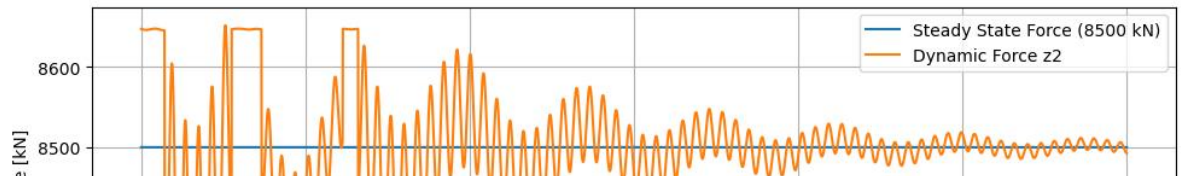
```
plt.grid(True)
plt.legend()


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

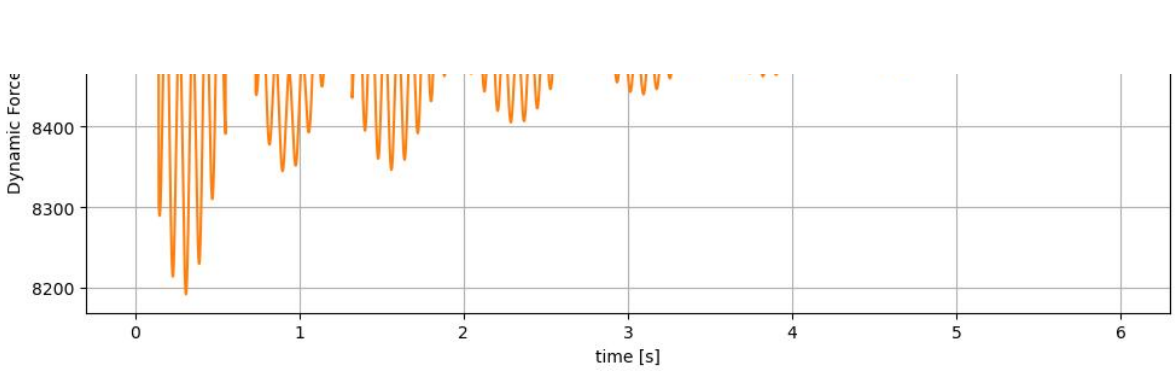

Position (m)
Time (s)
z1 position base shock damper and kabel
z2 position piston of the shock absorber
z3 position load


Velocity (m/s)
Time (s)
z1dot velocity piston of the shock absorber
z3dot velocity load


Acceleration [m/s^2]
time [s]
z3ddot Acceleration loadl


Pressure [bar]
time [s]
pressure rod side


Steady State Force (8500 kN)
Dynamic Force z2



In [ ]:

In [ ]: