Spring-Mass-Damper System Analysis

1. Mathematical Derivation

1.1. Introduction

The spring-mass-damper system is a fundamental model in mechanical engineering and physics used to describe the behavior of oscillating systems with damping. It consists of a mass attached to a spring and a damper, which are connected to a fixed support. This system is crucial for understanding vibrations, designing suspension systems, and analyzing the dynamic response of structures to external forces. The analysis of this system typically involves solving a second-order ordinary differential equation (ODE).

1.2. System Components and Forces

Consider a mass m attached to a spring with stiffness k and a damper with damping coefficient c. Let $x(t)$ be the displacement of the mass from its equilibrium position at time t. The forces acting on the mass are:

1. **Spring Force (F_s):** This is a restoring force exerted by the spring, proportional to the displacement x. According to Hooke's Law, $F_s = -kx$. The negative sign indicates that the force opposes the displacement.

2. **Damping Force (F_d):** This force is exerted by the damper and is proportional to the velocity of the mass, $\dot{x} = \frac{dx}{dt}$. The damping force opposes the motion, so $F_d = -c\dot{x}$.

3. **External Force ($F(t)$):** An optional external force that may act on the mass. For now, we will consider the case of free vibration (no external force), so $F(t) = 0$.

1.3. Equation of Motion

Applying Newton's Second Law ($\Sigma F = ma$) to the mass, where $a = \ddot{x} = \frac{d^2x}{dt^2}$ is the acceleration:

$$\Sigma F = F_s + F_d + F(t) = m\ddot{x}$$

Substituting the expressions for the forces:

$$-kx - c\dot{x} + F(t) = m\ddot{x}$$

Rearranging the terms to form the standard second-order linear ordinary differential equation for a spring-mass-damper system:

$$m\ddot{x} + c\dot{x} + kx = F(t)$$

For free vibration, where $F(t) = 0$, the equation becomes:

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

1.4. Characteristic Equation and Damping Ratios

To solve this homogeneous second-order linear ODE, we assume a solution of the form $x(t) = e^{\lambda t}$. Substituting this into the equation:

$$m(\lambda^2 e^{\lambda t}) + c(\lambda e^{\lambda t}) + k(e^{\lambda t}) = 0$$

Dividing by $e^{\lambda t}$ (since $e^{\lambda t} \neq 0$):

$$m\lambda^2 + c\lambda + k = 0$$

This is the **characteristic equation** of the system, a quadratic equation whose roots λ determine the nature of the system's response. The roots can be found using the quadratic formula:

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4mk}}{2m}$$

The behavior of the system depends on the value of the discriminant ($c^2 - 4mk$). We introduce two important parameters to characterize the system's damping:

1. **Natural Frequency (ω_n):** The frequency at which the system would oscillate if there were no damping ($c = 0$).

$$\omega_n = \sqrt{\frac{k}{m}}$$

2. **Damping Ratio (ζ):** A dimensionless parameter that describes the level of damping in the system relative to critical damping. Critical damping is the minimum damping required to prevent oscillation.

   $$\zeta = \frac{c}{2\sqrt{mk}}$$

We can express the characteristic equation in terms of ω_n and ζ. Dividing the characteristic equation by m:

$$\lambda^2 + \frac{c}{m}\lambda + \frac{k}{m} = 0$$

We know that $\frac{k}{m} = \omega_n^2$ and from the definition of ζ, $c = 2\zeta\sqrt{mk} = 2\zeta m\omega_n$, so $\frac{c}{m} = 2\zeta\omega_n$. Substituting these into the characteristic equation:

$$\lambda^2 + 2\zeta\omega_n\lambda + \omega_n^2 = 0$$

Now, the roots of this equation are:

$$\lambda = \frac{-2\zeta\omega_n \pm \sqrt{(2\zeta\omega_n)^2 - 4\omega_n^2}}{2}$$

$$\lambda = \frac{-2\zeta\omega_n \pm \sqrt{4\zeta^2\omega_n^2 - 4\omega_n^2}}{2}$$

$$\lambda = \frac{-2\zeta\omega_n \pm 2\omega_n\sqrt{\zeta^2 - 1}}{2}$$

$$\lambda = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$$

1.5. System Response Based on Damping Ratio

The nature of the system's response depends on the value of the damping ratio ζ:

1. **Underdamped ($\zeta < 1$):** The system oscillates with decreasing amplitude. The roots are complex conjugates:

   $$\lambda = -\zeta\omega_n \pm i\omega_n\sqrt{1 - \zeta^2}$$

   The solution is of the form:

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

   Where $\omega_d = \omega_n\sqrt{1 - \zeta^2}$ is the damped natural frequency.

2. **Critically Damped ($\zeta = 1$):** The system returns to equilibrium as quickly as possible without oscillating. The roots are real and equal:

   $\lambda = -\omega_n$

   The solution is of the form:

   $x(t) = (A + Bt)e^{-\omega_n t}$

3. **Overdamped ($\zeta > 1$):** The system returns to equilibrium slowly without oscillating. The roots are real and distinct:

   $\lambda = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$

   The solution is of the form:

   $x(t) = Ae^{\lambda_1 t} + Be^{\lambda_2 t}$

Where A and B are constants determined by initial conditions.

2. Simulation Code

2.1. Introduction

This section provides a Python-based simulation of the spring-mass-damper system, illustrating the different types of responses (underdamped, critically damped, and overdamped) based on the damping ratio. The simulation uses the analytical solutions derived from the characteristic equation to plot the displacement of the mass over time.

2.2. Code Description

The simulation code is structured as follows:

1. **System Parameters:** We define the mass (`m`) and spring stiffness (`k`). The natural frequency (`omega_n`) is calculated from these parameters.

2. **Time Array:** A time array (`time`) is created using `numpy.linspace` to cover the duration of the simulation.

3. **Initial Conditions:** Initial displacement (`x0`) and initial velocity (`v0`) of the mass are set.

4. **Case-by-Case Simulation:** The code then calculates and plots the response for three distinct damping scenarios:

    - **Underdamped:** A damping coefficient (`c_under`) is chosen such that the damping ratio (ζ) is less than 1 (e.g., 0.5). The damped natural frequency (ω_d) is calculated, and the analytical solution for underdamped motion is used to compute the displacement at each time step.
    - **Critically Damped:** The damping coefficient (`c_crit`) is set to achieve a damping ratio of exactly 1. The analytical solution for critically damped motion is then applied.
    - **Overdamped:** A damping coefficient (`c_over`) is chosen such that the damping ratio is greater than 1 (e.g., 1.5). The two distinct real roots (λ_1, λ_2) of the characteristic equation are calculated, and the analytical solution for overdamped motion is used.

5. **Plotting:** `matplotlib.pyplot` is used to plot the displacement versus time for each damping case on the same graph. This allows for a direct comparison of the different system behaviors. Legends, titles, and axis labels are included for clarity.

2.3. Python Code

```
 import numpy as np
import matplotlib.pyplot as plt

# System Parameters
m = 1.0  # Mass (kg)
k = 100.0  # Spring stiffness (N/m)

# Calculate natural frequency
omega_n = np.sqrt(k / m)

# Time array for simulation
time = np.linspace(0, 5, 500)  # 5 seconds, 500 points

# Initial conditions
x0 = 0.1  # Initial displacement (m)
v0 = 0.0  # Initial velocity (m/s)

plt.figure(figsize=(10, 6))

# --- Underdamped Case (zeta < 1) ---
c_under = 0.5 * 2 * m * omega_n  # Choose c for zeta = 0.5
zeta_under = c_under / (2 * m * omega_n)
omega_d_under = omega_n * np.sqrt(1 - zeta_under**2)

# Constants for underdamped solution
A_under = x0
B_under = (v0 + zeta_under * omega_n * x0) / omega_d_under

x_under = np.exp(-zeta_under * omega_n * time) * \
          (A_under * np.cos(omega_d_under * time) + B_under *
np.sin(omega_d_under * time))
plt.plot(time, x_under, label=f\"Underdamped (ζ = {zeta_under:.2f})\")

# --- Critically Damped Case (zeta = 1) ---
c_crit = 1.0 * 2 * m * omega_n  # Choose c for zeta = 1.0
zeta_crit = c_crit / (2 * m * omega_n)

# Constants for critically damped solution
A_crit = x0
B_crit = v0 + omega_n * x0

x_crit = (A_crit + B_crit * time) * np.exp(-omega_n * time)
plt.plot(time, x_crit, label=f\"Critically Damped (ζ = {zeta_crit:.2f})\",
linestyle=\'--\')

# --- Overdamped Case (zeta > 1) ---
c_over = 1.5 * 2 * m * omega_n  # Choose c for zeta = 1.5
zeta_over = c_over / (2 * m * omega_n)

lambda1_over = -zeta_over * omega_n + omega_n * np.sqrt(zeta_over**2 - 1)
lambda2_over = -zeta_over * omega_n - omega_n * np.sqrt(zeta_over**2 - 1)

# Constants for overdamped solution
# Using initial conditions: x(0) = x0, v(0) = v0
# x0 = A + B
# v0 = A*lambda1 + B*lambda2
# Solve for A and B
B_over = (v0 - lambda1_over * x0) / (lambda2_over - lambda1_over)
```

```
A_over = x0 - B_over

x_over = A_over * np.exp(lambda1_over * time) + B_over * np.exp(lambda2_over *
time)
plt.plot(time, x_over, label=f\"Overdamped (ζ = {zeta_over:.2f})\",
linestyle=\':\")

plt.title(\'Spring-Mass-Damper System Response\')
plt.xlabel(\'Time (s)\')
plt.ylabel(\'Displacement (m)\')
plt.grid(True)
plt.legend()
plt.show()
```

3. Conclusion

The spring-mass-damper system is a cornerstone in the study of vibrations and dynamic systems. Its behavior, governed by a second-order ordinary differential equation, is entirely characterized by the mass, spring stiffness, and damping coefficient. The concept of the damping ratio (ζ) is particularly crucial, as it categorizes the system's response into underdamped (oscillatory decay), critically damped (fastest return to equilibrium without oscillation), and overdamped (slow, non-oscillatory return to equilibrium). The Python simulation effectively visualizes these distinct behaviors, providing a clear understanding of how varying damping levels impact system performance. This model finds widespread application in engineering, from designing stable vehicle suspensions to earthquake-resistant building structures.

4. References

1. Inman, D. J. (2014). *Engineering Vibration*. Pearson.
2. Ogata, K. (2010). *Modern Control Engineering*. Prentice Hall.
3. Meriam, J. L., Kraige, L. G., & Bolton, J. N. (2018). *Engineering Mechanics: Dynamics*. John Wiley & Sons.
4. Matplotlib Development Team. (2024). *Matplotlib: Python plotting*. https://matplotlib.org/
5. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. (2020). *Array programming with NumPy*. Nature 585, 357–362. https://numpy.org/