Control Systems Theory: Comprehensive Lab Report

Jean Ephraim M. Bayaton, Daryll Aaron Camato, Alein Miguel P. Capañarihan, and Christian G. Cuaresma
Department of Electronics Engineering, Faculty of Engineering, University of Santo Tomas, Manila, Philippines,
{jeanephraim.bayaton, daryllaaron.camato, aleinmiguel.capanarihan,
christian.cuaresma }.eng@ust.edu.ph

***Abstract*—In this paper, the group presented the various experimental procedures that were accomplished throughout the preliminary period of their Control System Theory Laboratory Class. This paper consists of experiments related to the basics of Control Systems, Equation Modelling of Circuits and Physical Systems, and as well Equation Modelling for State-Space Representation. The group explored various commands and ways to formulate and come up with outputs that are needed for each experiments. Through the use of manual solving approached learned in this course, MATLAB, SIMULINK, and SIMSCAPE the group seeks similarity through the output using different approaches in a specific problem or task.**

Keywords—MATLAB, SIMULINK, SIMSCAPE, Transfer Function

I. Introduction

Control Systems are the basis for the open-loop and closed-loop systems we use daily, such as microwave ovens, elevators, air conditioning units, etc. This allows a specific system to be further developed and optimized and to make the system more stable [1-2].

Through the use of MATLAB, Simulink, and Simscape, the group was introduced to Control System Basics through the use of simulations. From introduction to Control Systems using MATLAB to creating State-Space Representation through manual solving, and through Coding.

MATLAB® is a high-level programming and numeric computing platform used for engineering and scientific applications such as data analysis, signal processing, control systems, and robotics. It features an intuitive programming language, interactive apps, built-in visualization tools, and specialized toolboxes for various fields. MATLAB also supports automatic code generation for embedded systems and integrates with Simulink® for block diagram-based simulations. Its ease of use, powerful computation capabilities, and industry-wide adoption make it a preferred tool for solving complex problems efficiently. [3]

Simulink® enables Model-Based Design, allowing engineers to create virtual models to simulate and test systems early in development. It supports Hardware-in-the-Loop testing, rapid prototyping, and automatic code generation for direct deployment to embedded systems. Simulink also ensures traceability across the development process and enables predictive maintenance and fault analysis by extending models to real-world operations. [4]

Simscape® allows you to quickly model physical systems within Simulink® by using component-based modeling that integrates with block diagrams. It enables simulation of systems like electric motors, hydraulic actuators, and refrigeration systems while supporting custom component creation using the MATLAB-based Simscape language. Simscape also facilitates control system design, model parameterization, and hardware-in-the-loop (HIL) deployment. [5]

In these experiments, the group was introduced to these three platforms as we utilize all three platforms in performing different tasks and come up with reliable yet meaningful outputs that may be further analyzed and compare with eachother. Through the use of these three platforms together with manual solution and/or computations, the group may be able to give insights on how a task can be completed or accomplished using four different approach in processing different tasks or experiments.

II. Experiment Proper

A. ***Experiment 1****: Introduction to Control Systems using Software Simulators*

i) Part 1

In the first Part, the tf command *(sys = tf([numerator], [denominator]))* was applied to identify the transfer function of the given row vectors. The row vectors applied to the command are the representation of polynomials that are in the s-domain [6]. When representing a polynomial in row vectors, the specific matrix represents the coefficients of a particular polynomial, starting from the leading term to the constant term.

Given the row vectors of [1 0] and [1 10], it can be clearly stated that these vectors correspond to s and s + 10, respectively.

```
%Part 1
G = tf([1 0], [1 10])
```

Fig E1.A Application through the use of TF Command.

ii) Part 2

Part 2 makes use of the same *tf* function to generate the transfer function given the row vectors of the numerator and denominator respectively. This transfer function is then analyzed as a system with a gain of 1 i.e unity gain, using negative feedback. This syntax for this code is as shown in Fig E1.B.

```
%Part 2
G = tf(num,denom);
x = feedback(G,H,-1);
```

Fig E1.B Part 2 Application through the use of Feedback Command

x = feedback(G,H,-1) indicates the system, gain, and lastly whether it is a positive or negative closed-loop feedback system, since the last parameter is *-1* this indicates that the feedback is negative

iii) Part 3

In Part 3, we analyze the closed-loop transfer function of a system with unity gain and positive feedback. For this configuration, the feedback function H is set to 1, and the closed-loop transfer function is determined using the `feedback(G, -H)` command in MATLAB. Adding a negative sign for the H in this command will give the function positive feedback because the default for this command is in the negative feedback state [7]. The code shows that the resulting transfer function captures the system's behavior under the influence of positive feedback.

```
%Part 3
H = 1;
y = feedback(G, -H)
```

Fig E1.C Application through the use of Feedback Command

The output transfer function describes the system's behavior with positive feedback. With H = -1, the transfer function G is modified to produce `y(s)= s/10`. Here, the numerator `s` represents the open-loop gain, while the denominator `10` reflects the system's dynamics influenced by feedback.

iv) Part 4

In Part 4 we determine the zero-pole-gain model of the given transfer function $G(s) = \frac{(20s^2 + 120s + 160)}{s^4 + 24s^3 + 191s^2 + 504s}$

The transfer function coefficients were extracted for the numerator since they are all multiples of 20; we factor 20, which gives [1 6 8], and set the gain *(k)* to 20. On the other hand, the denominator of the coefficients is [1 24 191 504 0].

Using the MATLAB command *roots()*, the roots given by the coefficients are returned. The numerator represents the zeros(z), while the denominator represents the poles(p).

The zpk() function was used to input the calculated zeros (z), poles (p), and gain (k) into MATLAB, yielding the ZPK model of the transfer function [8].

```
% Part 4
%Input the numerator and denominator of
the transfer function
numerator = [1 6 8]; k=20;%factor out 20
denominator = [1 24 191 504 0];

num_z = roots(numerator);
denom_p = roots(denominator)


%Zero pole gain model
tf=zpk(num_z,denom_p,[k])
```

Fig E1.D Transfer function using the ZPK command

v) **Part 5**

In the last Part, the group was tasked to replicate or follow the given block diagram, and the group should present the graph shown based on the block diagram being implemented in this Part. The block diagram and the simulation were performed through Simulink.

Fig E1.E shows the block diagram of the transfer function being implemented in this Part. The block diagram consists of three transfer functions combined with a unit-step pulse. These results were later applied to the feedback of the system presented. This type of system is an example of a closed-loop system because of a feedback signal.

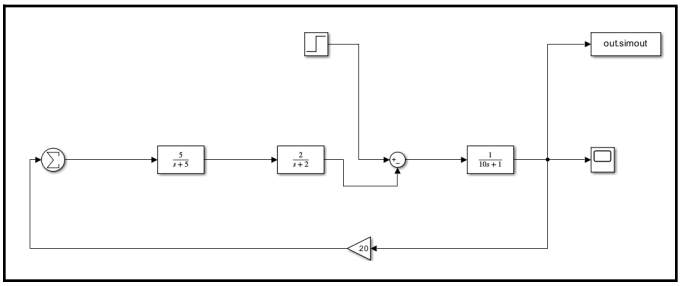


Fig 1.E Closed-Loop System Block Diagram through Simulink

B. ***Experiment 2:*** *Mathematical Modelling of Electrical Networks*

i) Part 1

For the first part of this experiment, the group is tasked to find the time-domain response of the given circuit, as shown in Fig E2.A below.

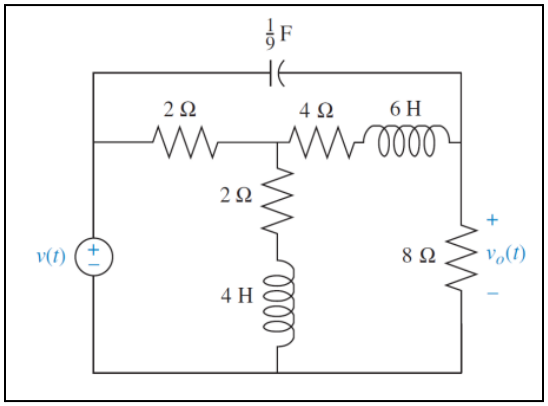


Fig E2.A Experimental Circuit

Before finding the needed time-domain response, the group identified the transfer function. The group manually solved for the matrices, which are the key values used throughout this part of the experiment. The solution for the matrix is as follows.

Mesh 1:

$$v(s) - 2I_1(s) - 2I_1(s) - 4sI_1(s) + 2I_2(s) + 4sI_2(s) + 2I_3(s) = 0 \quad \textbf{(1)}$$

$$[-v(s)] + 2I_1(s) + 2I_1(s) + 4sI_1(s) - 2I_2(s) - 4sI_2(s) - 2I_3(s) = 0 \quad \textbf{(2)}$$

$$(2 + 2 + 4s)I_1(s) + (-2 - 4s)I_2(s) - 2I_3(s) = v(s) \quad \textbf{(3)}$$

$$(4 + 4s)I_1(s) - (2 + 4s)I_2(s) - 2I_3(s) = v(s) \quad \textbf{(eq.1)}$$

Mesh 2:

$$-4sI_2(s) - 2I_2(s) - 4I_2(s) - 6sI_2(s) - 8I_2(s) + 4sI_1(s) + 2I_1(s) + 4I_3(s) + 6sI_3(s) = 0 \quad \textbf{(4)}$$

$$4sI_2(s) + 2I_2(s) + 4I_2(s) + 6sI_2(s) + 8I_2(s) - 4sI_1(s) - 2I_1(s) - 4I_3(s) - 6sI_3(s) = 0 \quad \textbf{(5)}$$

$$(4s + 2 + 4 + 6s + 8)I_2(s) - (4s + 2)I_1(s) - (4 + 6)I_3(s) = 0 \quad \textbf{(6)}$$

$$-(2 + 4s)I_1(s) + (14 + 10s)I_2(s) - (4 + 6s)I_3(s) = 0 \quad \textbf{(eq. 2)}$$

Mesh 3:

$$-2I_3(s) - 4I_3(s) - 6sI_3(s) - \frac{9}{s}I_3(s) + 2I_1(s) + 4I_2(s) + 6sI_2(s) = 0 \quad \textbf{(7)}$$

$$2I_3(s) + 4I_3(s) + 6sI_3(s) + \frac{9}{s}I_3(s) - 2I_1(s) - 4I_2(s) - 6sI_2(s) = 0 \quad \textbf{(8)}$$

$$(2 + 4 + 6s + \frac{9}{s})I_3(s) - 2I_1(s) - (4 + 6s)I_2(s) = 0 \quad \textbf{(9)}$$

$$-2I_1(s) - (4 + 6s)I_2(s) + (\frac{9+6s+6s^2}{s})I_3(s) = 0 \quad \textbf{(eq.3)}$$

The group developed the following matrix equation through this mesh analysis, as shown in Fig E2.B

$$\begin{bmatrix} 4+4s & -2-4s & -2 \\ -2-4s & 14+10s & -4-6s \\ -2 & -4-6s & \frac{6s^2+6s+9}{s} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} V_i(s) \\ 0 \\ 0 \end{bmatrix}$$

Fig E2.B Matrix Representation

This was later applied as an array in MATLAB to find our transfer function for this part. The transfer function was found using Cramer's Rule to solve determinants. The group simplified the Transfer Function using the simplifyFraction() command, which allowed the group to determine the transfer function denoted by G(s).

$$G(s) = \frac{V_o(s)}{V_i(s)} \quad \textbf{(10)}$$

After these procedures, the group found the time-domain response of the given transfer function and later analyzed the results from the graph provided in Fig E2.b

```
syms v s t
%Coefficients found using Manual Mesh
Analysis
A = [4+4*s -2-4*s -2; -2-4*s 14+10*s
-4-6*s; -2 -4-6*s 6+6*s+(9/s)];
A1 = A;
B = [v; 0; 0];
%Used for obtaining the variable I2
%which is important for finding Vo(t)
A1(:, 2) = B;
m = det(A);
n = det(A1);

%Vo(s)
Cs = simplifyFraction(n/m);
%Vo(s)/Vi(s)
Gs = simplifyFraction(8*Cs/v);

vt = 3.3 * heaviside(t);
Vs = laplace(vt);
Vcs = simplify(Gs*Vs);
```

```
% For symbolic time-domain
vc_t_symbolic = ilaplace(Vcs);

time = 0:0.1:10;

% Numerical evaluation of inverse Laplace
[num, den] = numden(Gs);

num_coeff = double(coeffs(num, s,
'All')); % numerator coefficients
den_coeff = double(coeffs(den, s,
'All')); % denominator coefficients

% Create transfer function system
sys = tf(num_coeff, den_coeff);
% Input signal (numerical)
v_input = 3.3 * ones(size(time)); % v(t)
= 3.3u(t)
% Simulate the system response
v_output = lsim(sys, v_input, time);
% Plot the output response
figure;
plot(time, v_output, 'LineWidth', 2);
grid on;
xlabel('Time (s)');
ylabel('v_o(t) (Volts)');
title('Time-Domain Response of the
Circuit');
% Optional: Display the symbolic solution
disp('Symbolic Time-Domain Response:');
disp(vc_t_symbolic);
```

Fig E2.C MATLAB Code to generate Time-Domain graph

ii) Part 2

MATLAB has a built-in tool called Simulink, which is used for graphical simulation and model design for control systems. Part 2 of this experiment will use the block diagram to confirm the Transfer Function solved in Part 1 of the experiment. The setup of the block diagram is shown below in Fig 1.I

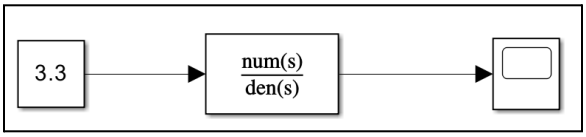


Fig E2.D Block Diagram in Simulink

The input is v(t) = 3.3u(t); this is simply represented by a *constant block* with a value of 3.3, as the unit step function only has values when t > 0. This input goes into the *transfer function block* with the transfer function previously obtained in Part 1, denoted by G =. The output of this block is read by the *scope block* and is shown in Fig 2.H

iii) Part 3

The essential tool for modeling and simulating physical systems, Simscape, will be used, and the information on how this electrical circuit is implemented and further analyzed under varying conditions is followed step by step.

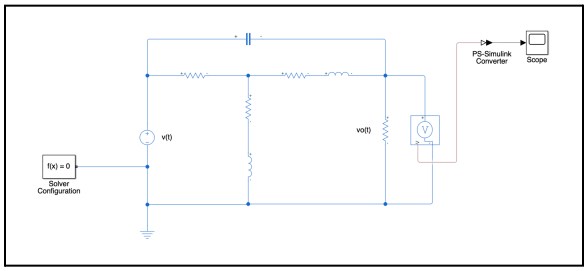


Fig E2.E Simulink Circuit Model

a. Model Implementation

To create a new Simulink model. All needed Simscape components must be placed on the model, such as resistors, capacitors, inductors, and AC or DC sources for the inputs. A voltage sensor is connected across the capacitor. Connect the ground and the Solver Configuration block to the circuit; these are usually included in the Simscape Electrical template. The Solver Configuration block is essential for running the simulation. Furthermore, the required electrical parameters of the circuit model are set. To view the output, use a voltage sensor to capture the signal, then convert it to a Simulink signal using a PS-Simulink Converter before connecting it to the scope.

b. Electrical Parameter Variation

Parameter variations are made to understand more about the behavior of the circuit. The applied input voltage is raised to 5V to determine the output response. Furthermore, the system's dynamics are analyzed by reducing the capacitance to 1/20 F. Moreover, both inductance values are lowered to 0.1 H, and the effects of these modifications on the system's stability are also discussed. These variances show the kinds of factors that are present and impact on the overall system performance.

c. Comparison with MATLAB and Simulink Results

The results from the Simscape model are compared with those implemented using MATLAB code and Simulink. Through this comparison, we can verify whether the derived transfer function and circuit model are accurate regardless of the chosen method.

C. ***Experiment 3***: *Mathematical Modelling of Physical Systems*

***i)* Part A.1**

Exercise A1 involves developing separate MATLAB functions where the variable *r*, which introduces nonlinearity

to the spring, is assigned three different values. To achieve this, each function must incorporate the parameters of the mass-spring-damper system and its governing differential equations. Initially, a function file defines the mass-spring system, including its parameters and operations. This process is then repeated three times, each corresponding to a different value of *r*, ensuring the system's differential equations reflect these variations

```
%mass spring function
function dxdt = mass_spring1(t,x)
M = 750; %kg
B = 30;  %Ns/m
Fa = 300;%N
K = 15;  %N/m
r = 1;
%dx/dt
dxdt(1,1) = x(2);
dxdt(2,1) = (-B*x(2)/M)-(K*(x(1).^r)/M)+(Fa/M);
end
```

Fig E3.A Code for the Mass-Spring System when r = 1

An additional M-file is developed to solve the differential equation using MATLAB's ode45 solver and to visualize the results for each value of r. The subplot() function displays multiple plots within a single window, allowing for a clear comparison of the system's response for different values of r. A superimposed plot is generated to illustrate the differences between the three cases in a single graph.

```
%solving the ODE
x0 = [0;0];
options = odeset('RelTol',[1e-4 1e-4],'AbsTol',[1e-5 1e-5],'Stats','on');
[t1,x1] = ode45('mass_spring1',[0 200],x0);
[t2,x2] = ode45('mass_spring2',[0 200],x0);
[t3,x3] = ode45('mass_spring3',[0 200],x0);
%plotting the velocity and displacement at r=1, 2 and 3

figure
hold on
plot(t1,x1(:,1),'k')
plot(t2,x2(:,1),'r')
plot(t3,x3(:,1),'b')
title('Displacement')
legend('r=1','r=2','r=3')
figure
hold on
plot(t1,x1(:,2),'k')
plot(t2,x2(:,2),'r')
plot(t3,x3(:,2),'b')
title('Velocity')
legend('r=1','r=2','r=3')
```

Fig E3.B Code for ODE Solving and Plotting

ii) Part A.2

Exercise A2 investigates the dynamic behavior of a mass-spring-damper system influenced by an external force. As depicted in the figure, the system comprises a mass *M*, a spring with a stiffness constant k, and a damper with a coefficient *B*. The displacement *y(t)* of the mass represents the output, while the applied force *f(t)* serves as the input.

By applying Newton's Second Law, we derive the governing differential equation for the system. A MATLAB script is then used to compute and plot the system's response for specific parameters: mass M=10, damping coefficient B=0.5, and spring constant k=1. The aim is to examine the system's behavior and verify that the response reaches a peak amplitude of approximately 1.8. The figure below illustrates the mechanical system under study:

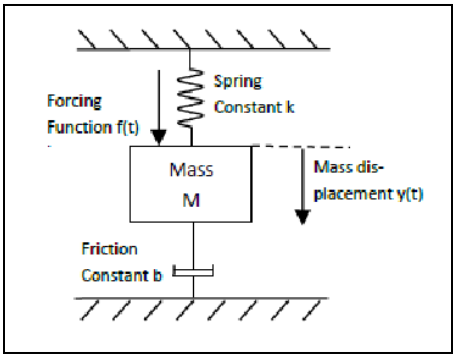


Fig E3.C Block Diagram of the Mass-Spring-Damper System

To analyze and solve the system response, we developed a MATLAB script that applies numerical methods to compute and visualize *y(t)*. This function, implemented in the first M-file, represents the second-order differential equation governing the system's motion. It incorporates system parameters such as mass, spring constant, and damping coefficient, converting them into state-space form. The differential equation model used in the MATLAB implementation is illustrated in the following figure.

```
%Item A2 Function of Mass-Spring-Damper
function dxdt = mass_spring(t, x)

M = 10;
B = 0.5;
Fa = 1;
K = 1;
r = 1;

dxdt = zeros(2,1);
dxdt(1,1) = x(2);
dxdt(2,1) = (-B*x(2)/M) - (K*(x(1)^r)/M) + (Fa/M);
end
```

Fig E3.D Function Code for the Mass-Spring-Damper System

Once the system equations are formulated, the next step is to solve and visualize the response. This is achieved through a second M-file, which utilizes MATLAB's `ode45` solver to solve the differential equation numerically. The script then plots the system's displacement and velocity over time. The subsequent figure illustrates the time-domain behavior of the system, including both position and velocity.

```
%Item A2 System and Plot
x0 = [0; 0];

options = odeset('RelTol', [1e-4 1e-4], 'AbsTol', [1e-5 1e-5], 'Stats', 'on');

[t,x] = ode45(@mass_spring, [0 200], x0);

plot(t,x);
xlabel('Time (s)');
ylabel('Response');
title('Mass-Spring System Response');
legend('Position (m)', 'Velocity (m/s)');
grid on;
```

Fig E3.E Code for Solving ODE and Plotting

iii) Part B.1

Part B.1 explores the utilization of Simulink to determine the response of the Mass-Spring System shown in Figure B1.1.

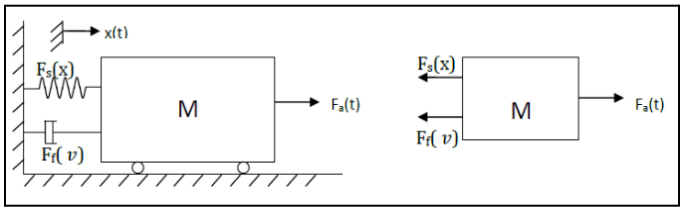


Fig E3.F Code for Solving ODE and Plotting

The parameters are M = 2 kg, K = 16 N/m, and B = 4 Ns/m. The input is a step parameter with initial value at zero and final value of eight at a time of 1 second. The Simulink model representing the system with the given parameters is shown below.

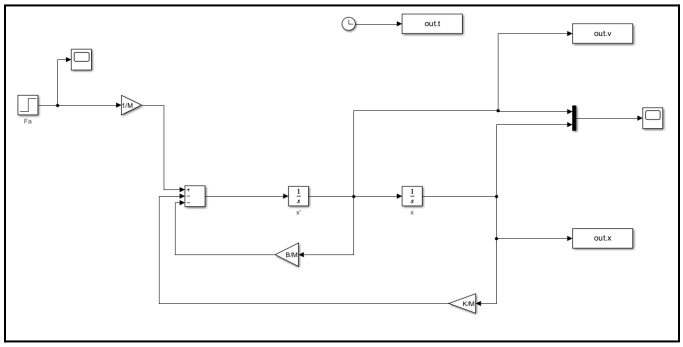


Fig E3.G Code for Solving ODE and Plotting

Once the model is complete, it is important both the parameter list and plots. The codes are as shown:

MATLAB (Editor Window):

```
%exp3b_1_param_file
%establishes parameter list

M = 2; %kg
K = 16; %N/m
B = 4; %Ns/m
```

Fig E3.H m-file to initialize parameters

```
%exp3b2_plot file
%plots the model of expb_1 model
%x-plot when B = 4
%v-plot when B = 4

subplot(2,1,1)
plot(out.x);
title('Displacement Time Series Plot')
grid
xlabel('Time(s)');
ylabel('Displacement(m)');
subplot(2,1,2)
plot(out.v)
title('Velocity Time Series Plot')
grid
xlabel('Time(s)');
ylabel('Velocity(m/s)');
```

Fig E3.I m-file to graph system output

iv) Part B.2

In this part of the experiment, the damping coefficient (B) will be varied. The objective is to examine how variations in B influence the mass-spring system over time, specifically regarding its overshoot, undershoot, settling time, and steady-state behavior.

The procedures are as follows

a. Block Diagram Construction

The Simulink model in Exercise B.1. was reconstructed as "exerciseB2.slx".

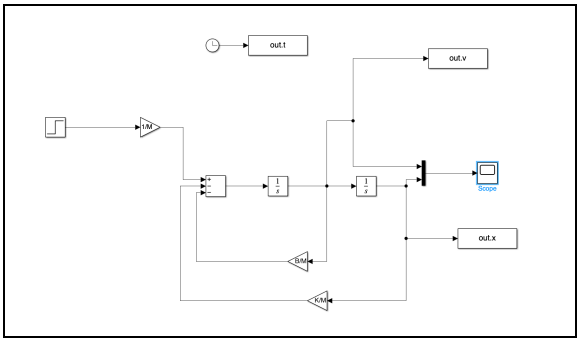


Fig E3.J Block Diagram of Simulink Model

b. Parameter Setting and Plotting of Graphs

A MATLAB script is created for the simulation of different values of *B*, and each system response is plotted. The system parameters are initialized as follows:

- *Mass (M) = 2 Kg*
- *Spring Constant (K) = 16 N/m*
- *Damping coefficient (B)* = varied between 4, 8, 12, and 25 Ns/m.

The *"sim"* command is utilized to simulate the system's response of the Simulink model ('exerciseB2.slx') in both the editor and command window. In the editor, it is executed to generate a combined plot on a single figure, enabling a direct comparison of varying the Damping Coefficient (B) values. In the command window, the scope readings for each damping value are displayed for individual comparison. Each simulation involves assigning the specified B value, running the model to retrieve time (out.t.data) and displacement (out.x.data), which are then used as inputs for the plot generation using the *"plot"* command.

MATLAB (Editor Window):

```
%% parameter list
M = 2; % mass (kg)
K = 16; % spring constant (N/m)
% B = varying damping value (Ns/m)
%% plot varying values of B wrt time
figure;
title('Damping Parameter Variation on
System Response');
xlabel('Time (s)');
ylabel('Displacement x(t)');
grid on;
hold on;
%% for B= 4(Ns/m)
B = 4;
out = sim('exp3');
plot(out.t.data,
out.x.data,'b','LineWidth',2);
grid on;
hold on;
%% for B= 8(Ns/m)
B = 8;
out = sim('exp3');
plot(out.t.data,
out.x.data,'r','LineWidth',2);
grid on;
hold on;
%% for B= 12(Ns/m)
B = 12;
out = sim('exp3');
plot(out.t.data,
out.x.data,'y','LineWidth',2);
grid on;
hold on;
%% for B= 25(Ns/m)
B = 25;
out = sim('exp3');
plot(out.t.data,
out.x.data,'m','LineWidth',2);
grid on;
hold on;
lgd = legend('B = 4','B = 8','B = 12','B
= 25');
lgd.Location = "north";
lgd.Orientation = "horizontal";
```

Fig E3.K Parameter Initialization and Variation

MATLAB (Command Window):

```
Command Window
>> %% parameter list
M = 2; % mass (kg)
K = 16; % spring constant (N/m)
>> B = 4;
>> B = 8;
>> B = 12;
>> B = 25;
>>
```

Fig E3.L Individual Scope plots for varied B-parameter

D. ***Experiment 4****:*

i) **Part 1**

Given the mechanical system shown in Fig. 1.1, Derive the state-space representation. Consider the output to be $x_3(t)$.

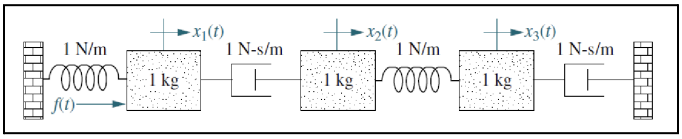
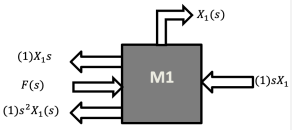


Fig E4.A Given Translational Mechanical System

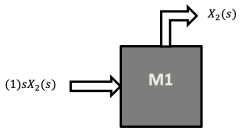
The components of the given translational mechanical system are first identified, involving masses M1, M2, and M3, dampers, and springs. A fundamental step was establishing the direction of motion and interaction among the masses to determine the system's dynamic behavior and applying Newton's Second Law to each mass.

Modeling in the Laplace Domain: The solution begins with manually writing the equations of motion, into a set of algebraic equations. These equations will be transformed into the Laplace variable. These equations represent the mathematical model of the dynamical system in the frequency domain. The solution is as follows:

Move M1, Hold M2



Move M2, Hold M1



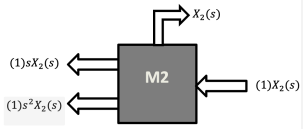
$-(s^2 + s + 1)x_1(s) + F(s) + sx_2(s) = 0$ **(1)**

$-1[-(s^2 + s + 1)x_1(s) + F(s) + sx_2(s) = 0] - 1$ **(2)**

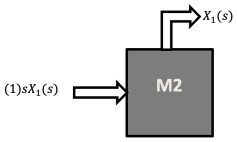
$F(s) = (s^2 + s + 1)x_1(s) - sx_2(s)$ **(3)**

$F(s) = s^2x_1(s) + sx_1(s) + x_1(s) - sx_2(s)$ **(eq. 4)**

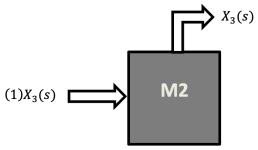
Move M2, Hold M1, M3



Move M1, Hold M2



Move M3, Hold M2



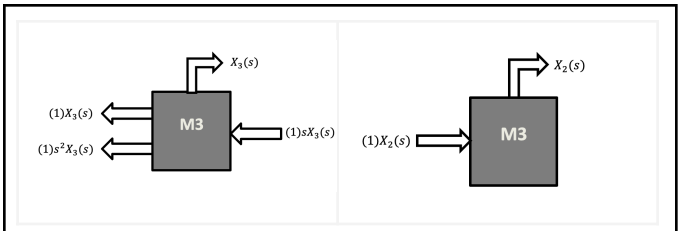
$-(s^2 + s + 1)x_2(s) + sx_1(s) + x_3(s) = 0$ **(5)**

$-1[-(s^2 + s + 1)x_2(s) + sx_1(s) + x_3(s) = 0] - 1$ **(6)**

$-sx_1(s) + s^2x_2(s) + sx_2(s) + x_2(s) - x_3(s) = 0$ **(eq. 7)**

Move M3, Hold M2

Move M2, Hold M3



$-(s^2 + s + 1)x_3(s) + x_2(s) = 0$ **(8)**

$-1[-(s^2 + s + 1)x_3(s) + x_2(s)] - 1$ **(9)**

$-x_2(s) + s^2x_3(s) + sx_3(s) + x_3(s) = 0$ **(eq. 10)**

Fig E4.B Given Translational Mechanical System

Introduction of state variables: The Laplace domain equations are equivalent to linear differential equations in the time domain. To represent the system in state space, the differential equations are transformed into a set of state variables, and into their equivalent state equations.

From equation 4:

$f(t) = \frac{d^2x_1}{dt^2} + \frac{dx_1}{dt} + x_1 - \frac{dx_2}{dt}$ **(11)**

$\frac{d^2x_1}{dt^2} = -\frac{dx_1}{dt} - x_1(s) + \frac{dx_2}{dt} + f(t)$ **(12)**

$\ddot{x}_1 = -\dot{x}_1 - x_1 + \dot{x}_2 + f(t)$ **(eq. 13)**

From equation 7:

$-\frac{dx_1}{dt} + \frac{d^2x_2}{dt^2} + \frac{dx_2}{dt} + x_2 - x_3 = 0$ **(14)**

$-\dot{x}_1 + \ddot{x}_2 + \dot{x}_2 + x_2 - x_3 = 0$ **(15)**

$\ddot{x}_2 = \dot{x}_1 - \dot{x}_2 - x_2 + x_3$ **(eq.16)**

From equation 10:

$-x_2 + \frac{d^2x_3}{dt^2} + \frac{dx_3}{dt} + x_3 = 0$ **(17)**

$-x_2 + \ddot{x}_3 + \dot{x}_3 + x_3 = 0$ **(18)**

$\ddot{x}_3 = x_2 - \dot{x}_3 - x_3$ **(eq. 19)**

Fig E4.C Given Translational Mechanical System

Each derived equation resulted in the following second-order differential equations describing the dynamics of each mass

State variables: A set of first-order state variables was assigned to represent the system dynamics:

$$z_1 = x_1$$
$$z_2 = \dot{x}_1$$
$$z_3 = x_2$$
$$z_4 = \dot{x}_2$$
$$z_5 = x_3$$
$$z_6 = \dot{x}_3$$

Fig E4.D Given Translational Mechanical System

State equations: These state variables were substituted into the second-order differential equations and were all expressed in the first-order form:

$$\dot{z}_1 = z_2$$

$$\dot{z}_2 = \ddot{x}_1 = -\dot{x}_1 - x_1 + \dot{x}_2 + f(t)$$
$$\dot{z}_2 = -z_2 - z_1 + z_4 + f(t)$$

$$\dot{z}_3 = z_4$$

$$\dot{z}_4 = \ddot{x}_2 = \dot{x}_1 - \dot{x}_2 - x_2 + x_3$$
$$\dot{z}_4 = z_2 - z_4 - z_3 + z_5$$

$$\dot{z}_5 = z_6$$

$$\dot{z}_6 = \ddot{x}_3 = x_2 - \dot{x}_3 - x_3$$
$$\dot{z}_6 = z_3 - z_6 - z_5$$

Fig E4.E Given Translational Mechanical System

ii) Part 2

In mechanical systems, the transfer function $\frac{X(s)}{F(s)}$ relates the displacement $X(s)$ to the applied force $F(s)$ in the Laplace domain. For a translational system consisting of a mass m, damping coefficient c, and spring constant k, the equations of motion can be derived from Newton's second law. By transforming these equations into the Laplace domain, we can obtain the transfer function, which is essential for analyzing system behavior and designing controllers.

$$f(t) = \frac{d^2x}{dt^2} + \frac{dx_1}{dt} + x_1 - \frac{dx_2}{dt}$$
$$= F(s) = (s^2 + s + 1)\,X_1(S) - sX_2(s)$$

$$0 = -\frac{dx_1}{dt} + \frac{d^2x}{dt^2} + \frac{dx_2}{dt} - x_2 - x_3$$
$$= X_3(s) = -sX_1 + (s^2 + s + 1)X_2(s)$$

$$0 = -x_2 + \frac{d^2x_3}{dt^2} + \frac{dx_3}{dt} + x_3$$
$$= X_2(s) = (s^2 + s + 1)X_3(s)$$

$$X_3(s) = -sX_1(s) + (s^2 + s + 1)(s^2 + s + 1)X_3(s)$$

$$X_1(s) = \frac{((s^2+s+1)-1)X_3(s)}{s}$$

$$F(s) = (s^2 + s + 1)\frac{(s^2+s+1)X_3(s)}{s} - s(s^2 + s + 1)X_3(s)$$

$$F(s) = (s^2 + s + 1)\frac{(s^2+s+1)-1}{s} - s(s^2 + s + 1)X_3(s)$$

$$\frac{X_3(s)}{F(s)} = \frac{1}{s^6+3s^5+5s^4+6s^3+4s^2+2s}$$

Fig E4.F Solving for the Transfer Function of the State Space

iii) Part 3

In this part of the experiment, the group was tasked to use the state-space representation that was solved in Part 1 and make a code in MATLAB that will represent the state-space that was solved previously. The group first called or set the matrices for the matrix A, B, C, D, that are represented through the equations

ż = Az + Bu **(20)**

y = Cz + Du **(21)**

After calling the needed matrices, the group used the command ss() which outputs the state space model for the given matrices that was set.

```
%% Step 3
```

```
A = [0 1 0 0 0 0;
   -1 -1 0 1 0 0;
   0 0 0 1 0 0;
   0 1 -1 -1 1 0;
   0 0 0 0 0 1;
   0 0 1 0 -1 -1];
B = [0; 1; 0; 0; 0; 0];
C = [0 0 0 0 1 0];
D = 0;

z = ss(A,B,C,D)
```

Fig E4.G Solving for the Transfer Function via ss() command

***iv)* Part 4**

Part 4 is the continuation of part 3; same as what was asked in Part 2, the group was tasked to find the transfer function of the given state-space representation through MATLAB. By using the command ss2tf() the group may be able to find the numerator and denominator of the given state-space matrices denoted by matrix A, B, C, and D, but using the values for numerator and denominator as the final values for the transfer function will only output a specific constant because the output of ss2tf() command which is denoted by num and den will only output a matrix for the representation of the polynomial.

Thus, the group further fixed this problem by using the command tf(). Through the use of the num and den values, the tf() command will have a fraction-based output which is equal to the C(s)/R(s) or what we call the transfer function.

```
%% Step 4
[num, den] = ss2tf(A,B,C,D,1);
trans_func = tf(num, den)
```

Fig E4.H Transfer Function using ss2tf() and tf() commands

III. Data And Results

A. *Experiment 1: Introduction to Control Systems using Software Simulators*

i) Part 1

```
G =

    s
  ------
  s + 10

Continuous-time transfer function.
```

Fig E1.a Output Transfer Function using TF Command

ii) Part 2

```
x =

    s
  ------
  2s + 10

Continuous-time transfer function.
```

Fig E1.b Negative Feedback Output Transfer Function

iii) Part 3

```
y =

  s
  --
  10

Continuous-time transfer function.
```

Fig E1.c Positive Feedback Output Transfer Function

iv) Part 4

```
denom_p =

         0
   -9.0000
   -8.0000
   -7.0000


tf =

    20 (s+4) (s+2)
  -------------------
  s (s+9) (s+8) (s+7)

Continuous-time zero/pole/gain model.
Model Properties
```

Fig E1.d Output Poles and Zero-Pole-Gain Model

v) Part 5

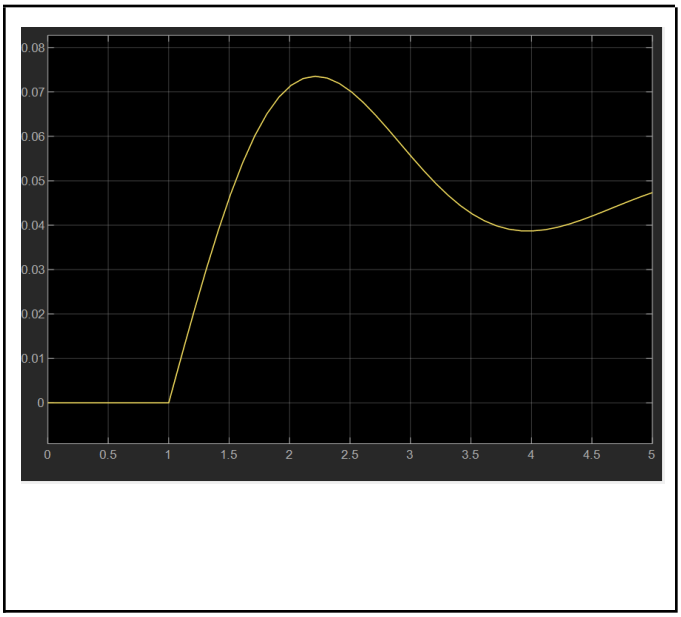


Fig E1.e Output of Block Diagram in Simulink

B. ***Experiment 2:*** *Mathematical Modelling of Electrical Networks*

i) Part 1

```
>> Gs

Gs =

(4*(12*s^3 + 24*s^2 + 28*s + 9))/(48*s^3 + 150*s^2 + 220*s + 117)
```

Fig E2.a Transfer Function G(s)

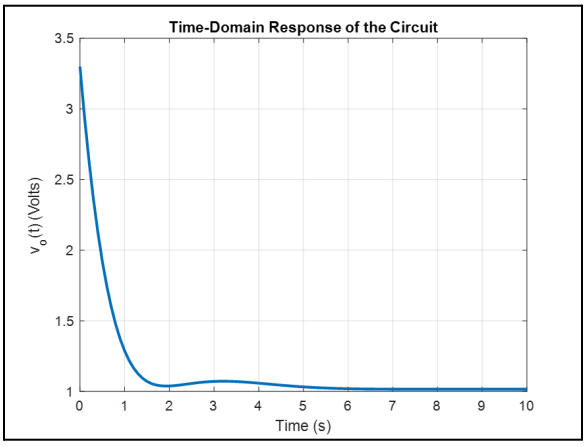


Fig E2.b Time-domain response of the circuit

At 0 seconds, the voltage initially measures around 3.3V, aligning with the given supply voltage expression, 3.3u(t). As time advances, the voltage drops sharply before stabilizing around 2 seconds, marking the settling time. Beyond this point, the voltage remains steady at approximately 1V and persists at this level until the end of the observed time frame (10 seconds), representing the final value of vo(t).

ii) Part 2

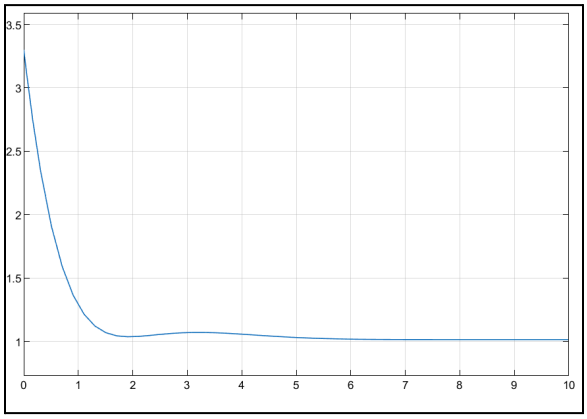


Fig E2.c Scope Output from Simulink

iii) Part 3

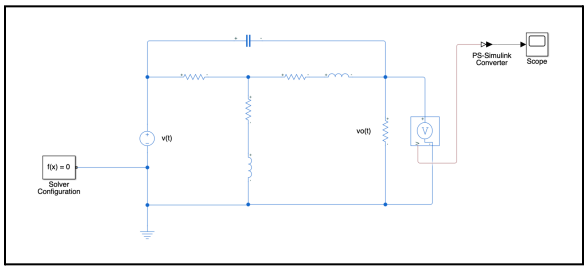


Fig E2.d Circuit Snapshot

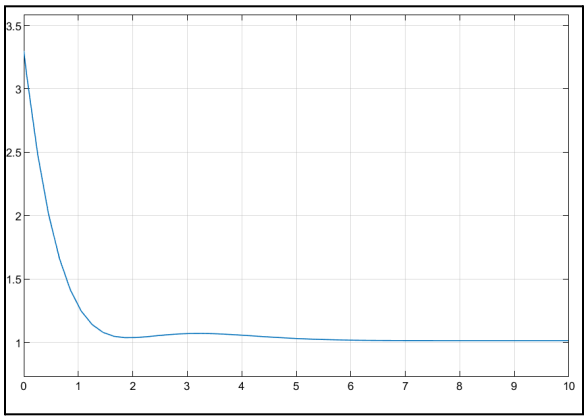


Fig E2.e Time-Domain response Vo(s)

As illustrated in Fig 2.J , the results from Simulink closely resemble those from Step 1

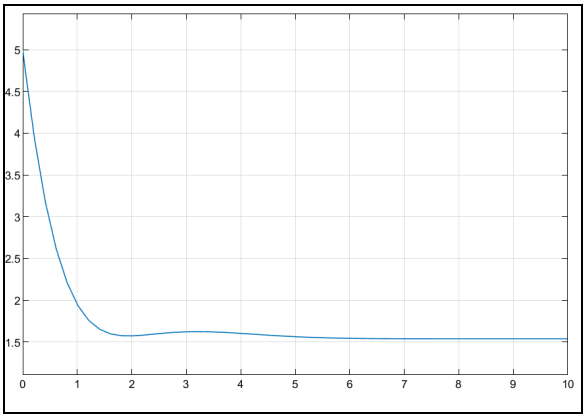


Fig E2.f Time-Domain response with increase input voltage of 5V

When the input voltage is raised to 5V, the output initially drops quickly as the voltage increases before stabilizing.

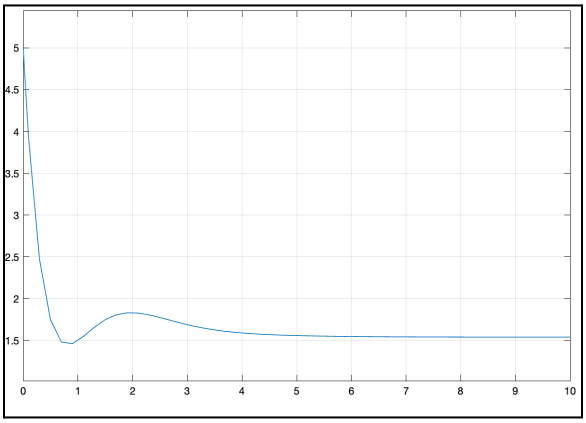


Fig E2.g Time-Domain response with increase reduce capacitance (1/20F)

The capacitance was set to 1/20F. Following that, the circuit's behavior completely changed. There was a higher oscillation effect and an increase in its transient response. It displayed a shorter settling time and higher overshoot. The capacitor shows the system's decreased capacity for energy storage; the drop in capacitance directly affects the system's stability and damping.

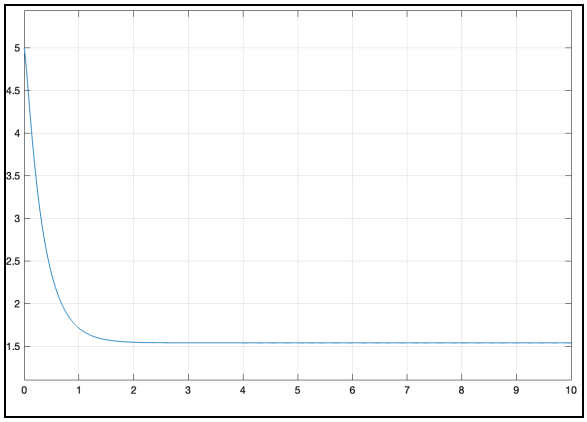


Fig E2.h Time-Domain response with reduced inductance (0.1H)

The inductances were lowered to 0.1 H, and the system responded faster. Since it's more sensitive to changes in the input, the system oscillates more before reaching a steady state. Furthermore, the circuit's total impedance decreased due to the reduced inductance, which increased the energy transfer between each component. This experiment further highlighted the importance of inductance in altering the circuit's transient and steady-state characteristics.

C. ***Experiment 3***: *Mathematical Modelling of Physical Systems*

i) Part A.1

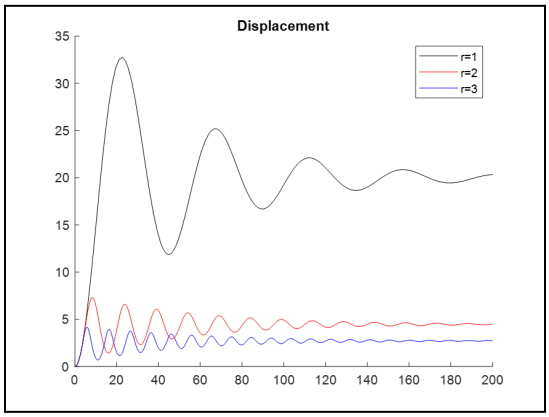


Fig E3.a Superimposed Displacement for varying values of r

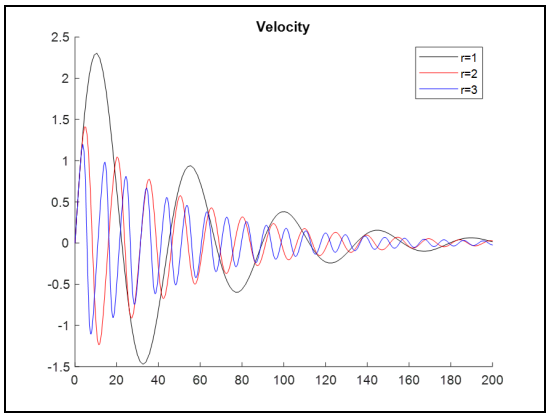


Fig E3.b Superimposed Velocity for varying values of r

ii) Part A.2

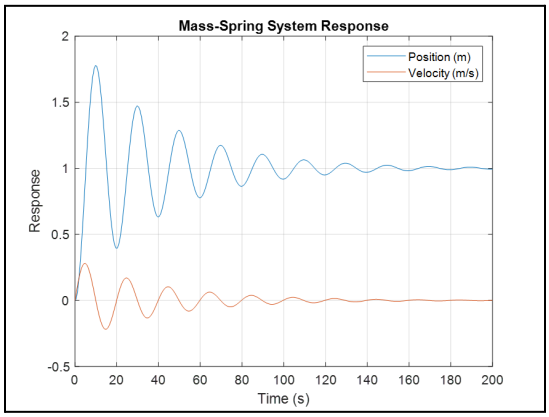


Fig E3.c Time Response of the Mass-Spring-Damper System

iii) Part B.1

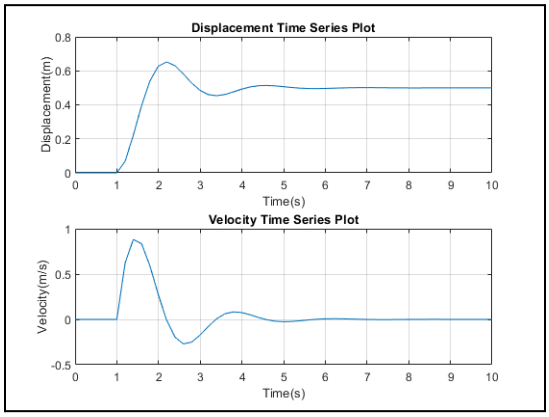


Fig E3.d Output Plots of the System

iv) Part B.2

a. Individual Plots (Scope Reading)

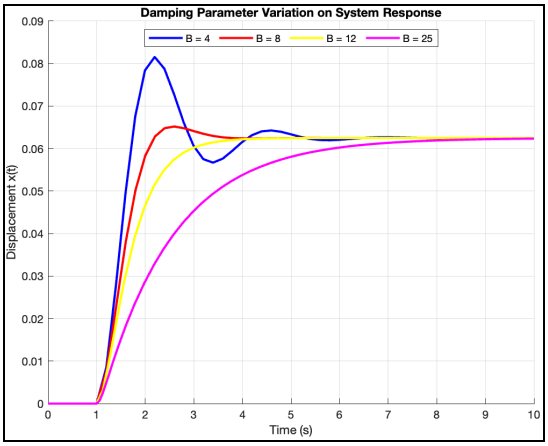


Fig E3.e Plot of varying B-Parameter

b. Individual Scope Reading

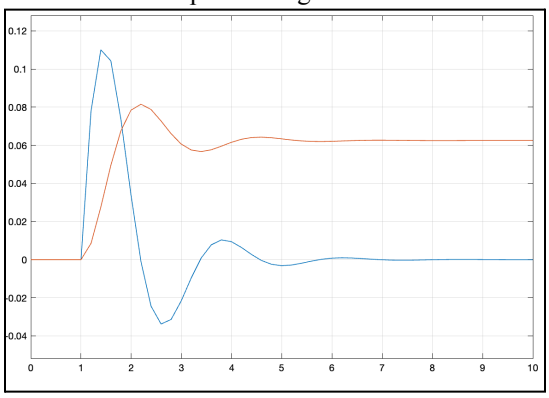


Fig E3.f Scope of reading at B=4

The system displays oscillatory behavior with a significant overshoot. The settling time is relatively long as the system oscillates before reaching stability.

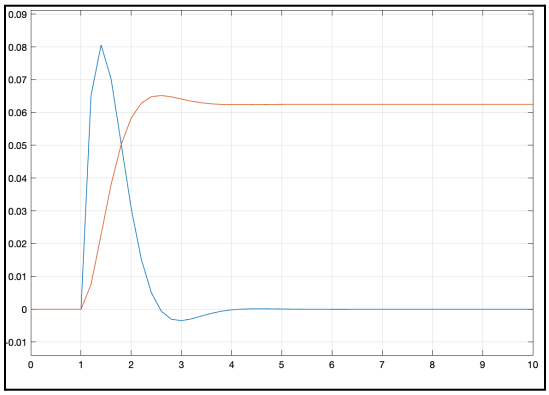


Fig E3.g Scope of reading at B = 8

The oscillations are smaller than those observed with B = 4, and the system stabilizes faster with less overshoot.

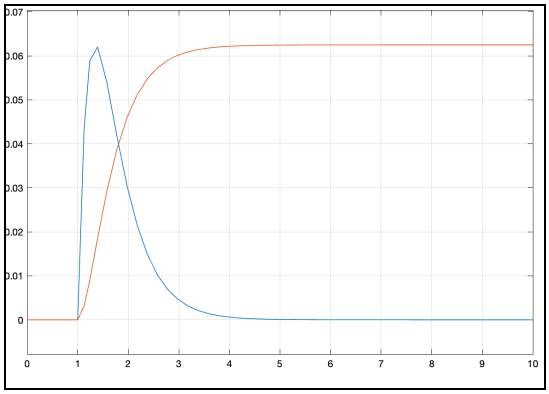


Fig E3.h Scope of reading at B = 12

The response is significantly smoothed, with minimal oscillations, and the system reaches stability much faster.

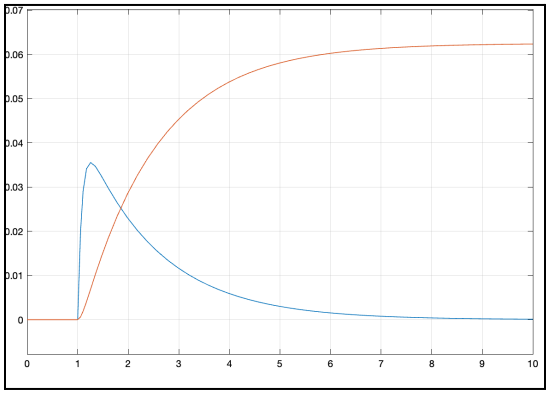


Fig E3.i Scope of reading at B = 25

The system exhibits a slow response without any oscillations. Furthermore, the settling time is longer, as the system takes longer to stabilize.

*D. **Experiment 4**: State Space Representation*

i) Part 1

The state equations from the previous section were arranged into their corresponding standard state-space format in matrix form and the output equation was considered $x_3(t)$as the output. As a final step, the state and output equations were combined into the complete state-space model:

State equation:

$$\dot{z} = Az + Bu$$

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \\ \dot{z}_5 \\ \dot{z}_6 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} f(t)$$

Output equation:

$$y = Cz + Du$$

$$y = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{bmatrix}$$

Fig. 1.6 State-Space Representation through manual solving

ii) Part 2

The solution follows a systematic approach to deriving the transfer function $\frac{X_3(s)}{F(s)}$ for a mechanical system. First, Newton's Second Law is applied, and the equations of motion are transformed into the Laplace domain. The force equation is expressed in terms of displacement variables $X_1(s)$, $X_2(s)$, and $X_3(s)$. Then, the relationships between these variables are established, leading to expressions for $X_2(s)$ and $X_3(s)$ in terms of each other. By substituting these relationships, $X_1(s)$ is expressed in terms of $X_3(s)$, and the force equation is further simplified. Finally, solving $\frac{X_3(s)}{F(s)}$ results in the transfer function:

$$\frac{X_3(s)}{F(s)} = \frac{1}{s^5+3s^4+5s^3+6s^2+4s+2}$$

Fig. 2.2 Simplified Transfer Function through manual solving

This transfer function characterizes the system's response to an applied force, helping in analyzing system dynamics and controller design.

iii) Part 3

```
>> Step3

z =

  A =
       x1   x2   x3   x4   x5   x6
   x1   0    1    0    0    0    0
   x2  -1   -1    0    1    0    0
   x3   0    0    0    1    0    0
   x4   0    1   -1   -1    1    0
```

```
   x5    0    0    0    0    0    1
   x6    0    0    1    0   -1   -1

  B =
        u1
   x1    0
   x2    1
   x3    0
   x4    0
   x5    0
   x6    0

  C =
        x1   x2   x3   x4   x5   x6
   y1    0    0    0    0    1    0

  D =
        u1
   y1    0

Continuous-time state-space model.
```

Fig. 3.2 Output State-Space Representation through MATLAB

iv) Part 4

```
trans_func =

                     s + 6.206e-17
  ----------------------------------------------------
  s^6 + 3 s^5 + 5 s^4 + 6 s^3 + 4 s^2 + 2 s - 6.402e-17

Continuous-time transfer function.
```

Fig. 4.2 Output Transfer Function through MATLAB

IV. Conclusion

This study provided valuable insights into control system behavior, stability, and response characteristics through software simulations, mathematical modeling, and state-space representation. Using MATLAB, Simulink, and Simscape, the group analyzed transfer functions, feedback mechanisms, and system parameter effects on electrical and mechanical systems.

Experiment 1 explored transfer functions and feedback, demonstrating the impact of open-loop and closed-loop configurations on stability. Experiment 2 validated MATLAB's effectiveness in modeling electrical circuits, highlighting the influence of capacitance and inductance. Experiment 3 examined mass-spring-damper systems, confirming the role of damping in oscillations and settling time. Experiment 4 focused on state-space representation, enabling systematic system analysis and transfer function derivation.

Overall, this study emphasized the importance of computational tools in modern control engineering, reinforcing key principles for future applications in electrical and mechanical system design.

References

[1] R. Davis, "Six Benefits of using a Control System," blog.boston-engineering.com, May 03, 2023. https://blog.boston-engineering.com/six-benefits-of-using-a-control-system

[2] M. Singh Dhaka and A. Rehalia, "Importance and Scope of Control Systems," International Journal of Advanced Engineering Research and Applications (IJA-ERA, no. 2, 2016, Accessed: Feb. 04, 2025. [Online]. Available: https://www.ijaera.org/manuscript/20160207007.pdf

[3] MathWorks, "What Is MATLAB?" MathWorks, 2025. [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html. [Accessed: 22-Mar-2025].

[4] MathWorks, "Simulink," MathWorks, 2025. [Online]. Available: https://www.mathworks.com/products/simulink.html. [Accessed: 22-Mar-2025].

[5] MathWorks, "Simscape," MathWorks, 2025. [Online]. Available: https://www.mathworks.com/products/simscape.html. [Accessed: 22-Mar-2025].

[6] "Transfer function model - MATLAB," www.mathworks.com.

[7] "Feedback connection of multiple models - MATLAB feedback," www.mathworks.com. https://www.mathworks.com/help/control/ref/inputoutputmodel.feedback.html

[8] "Zero-pole-gain model - MATLAB," www.mathworks.com. https://www.mathworks.com/help/control/ref/zpk.html