NAGARJUNA

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

VISION

To transform the students as leaders in Electronics & Communication Engineering to achieve professional excellence in the competitive world.

MISSION

M1: To create an environment for the students to have strong academic fundamentals and enable them to be life – long learners.

M2: To provide modern tools to the students in the field of electronics and communication to meet the real – world challenges.

M3: To develop communication skill, leadership qualities, team work and skills for continuing education among the students.

M4: To inculcate Ethics, Human values for solving societal problems and environmental protection.

M5: Validate engineering knowledge through innovative research projects to enhance their employability and entrepreneurship skills.

NAGARJUNA COLLEGE OF ENGINEERING &TECHNOLOGY

(An Autonomous College under VTU)

Venkatagiri post, Devanahalli, Bengaluru-562164

Department of Electronics & Communication Engineering

COLLEGE OF ENGINEERING AND TECHNOLOGY

Lab Manual

IV Semester

Course Name: Circuits and Controls

Course Code: 21ECI43

Name of the student: _____

USN: _____

Section:_____

Batch:_____

Experiment 1:

VERIFICATION OF SUPERPOSITION THEOREM

Aim: To verify the superposition theorem for the given circuit.

Apparatus Required:

Sl.No.	Apparatus	Range	Quantity
1	RPS (regulated power supply)	(0-30V)	2
2	Ammeter	(0-10mA)	1
3	Resistors	1kΩ, 330Ω, 220Ω	1 each
4	Bread Board	--	1
5	Wires	--	Required

Statement:

Superposition theorem states that in a linear bilateral network containing more than one source, the current flowing through the branch is equal to the algebraic sum of all the currents flowing through that branch when sources are considered one at a time and replacing other sources by their respective internal resistances.

Procedure:

1. Give the connections as per the diagram.

2. Set a particular voltage value using RPS1 and RPS2 & note down the ammeter reading

3. Set the same voltage as in circuit 1 using RPS1 alone and disconnect RPS2 and short circuit the terminals and note the ammeter reading.

4. Repeat the same procedure with RPS2 and note down the ammeter reading.

5. Verify superposition theorem.

Circuit Diagram:

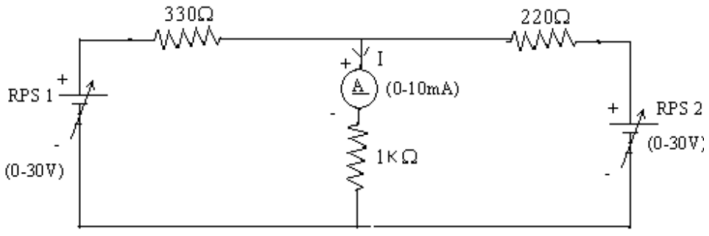


Fig.1

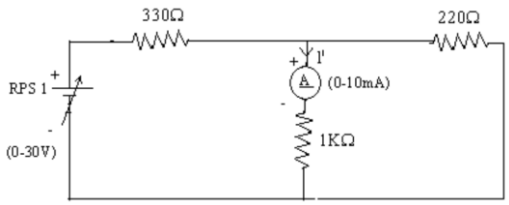


Fig.2

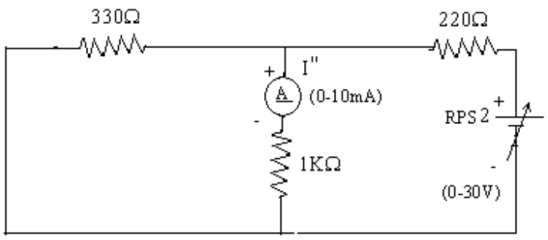


Fig.3

Tabular Column

	RPS1 in V		RPS2 in V		Current in mA	
	Theoretical	**Practical**	**Theoretical**	**Practical**	**Theoretical**	**Practical**
Fig.1					I=	I=
Fig.2					I'=	I'=
Fig.3					I"=	I"=
	Total Current in the current				I= I'+ I"=_____	I= I'+ I"=_____

Result:

Thus Superposition theorem was been verified both theoretically and practically.

Experiment 2:

VERIFICATION OF THEVENIN'S THEOREM

Aim:

To verify Thevenin's theorem and to calculate the load current for the given circuit.

Apparatus Required:

Sl.No.	Apparatus	Range	Quantity
1	RPS (regulated power supply)	(0-30V)	2
2	Ammeter	(0-10mA)	1
3	Resistors	1KΩ, 330Ω	3,1
4	Bread Board	--	Required
5	DRB	--	1

Statement:

Any linear bilateral, active two terminal networks can be replaced by a equivalent voltage source (V_{TH}). Thevenin's voltage or VOC in series with looking back resistance R_{TH}.

Procedure:

1. Connections are given as per the circuit diagram.

2. Set a particular value of voltage using RPS and note down the corresponding ammeter readings.

To find VTH

3. Remove the load resistance and measure the open circuit voltage using multimeter (V_{TH}).

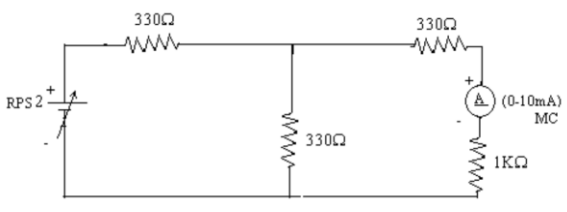
To find RTH

4. To find the Thevenin's resistance, remove the RPS and short circuit it and find the R_{TH} using multimeter.

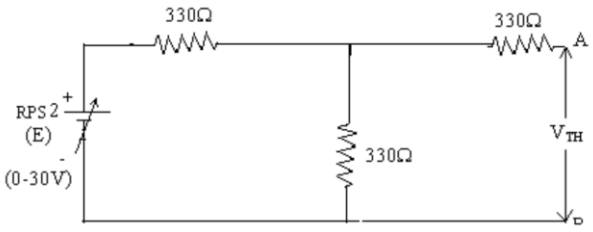
5. Give the connections for equivalent circuit and set VTH and RTH and note the corresponding ammeter reading.

6. Verify Thevenin's theorem.

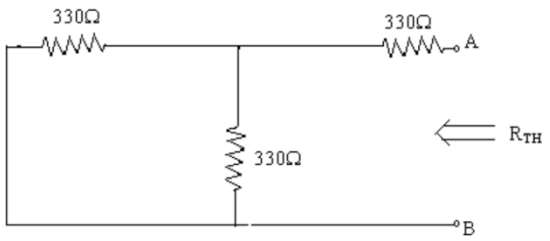
Circuit diagram



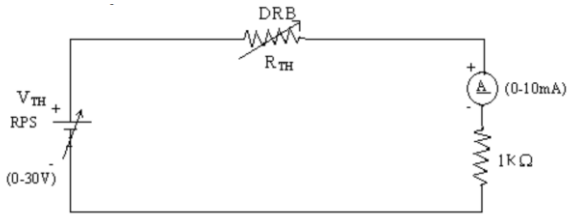
To find load current



To find V_{TH}



To find R_{TH}



Thevenin's Equivalent circuit

Tabular Column:

Parameters		**Theoretical**	**Practical**
RPS in V			
V_{TH} in V			
R_{TH} in Ohms			
Load current (I_L)	**Original circuit**		
	Thevenin's Equivalent circuit		

Result:

Thus Thevenin's theorem was verified both practically and theoretically

Experiment 3:

VERIFICATION OF NORTON'S THEOREM

Aim:

To verify Norton's theorem for the given circuit and to determine the load current for the given circuit.

Apparatus Required:

Sl.No.	Apparatus	Range	Quantity
1	Ammeter	(0-10mA) MC (0-30mA) MC	1 1
2	Resistors	330, 1KΩ	3,1
3	RPS	(0-30V)	2
4	Bread Board	--	1
5	Wires	--	Required

Statement:

Any linear, bilateral, active two terminal network can be replaced by an equivalent current source (IN) in parallel with Norton's resistance (RN)

Procedure:

1. Connections are given as per circuit diagram.

2. Set a particular value in RPS and note down the ammeter readings in the original circuit.

To Find I_N:

3. Remove the load resistance and short circuit the terminals.

4. For the same RPS voltage note down the ammeter readings.

To Find R_N:

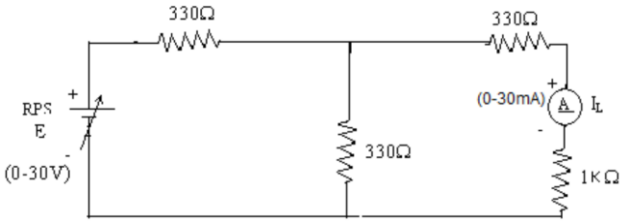
5. Remove RPS and short circuit the terminal and remove the load and note down the resistance across the two terminals.

Equivalent Circuit:

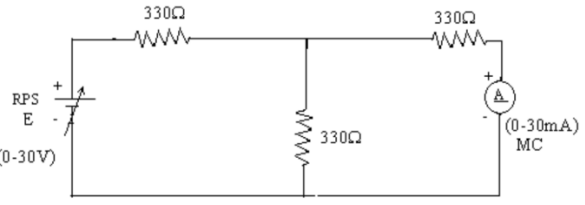
6. Set IN and RN and note down the ammeter readings.

7. Verify Norton's theorem.

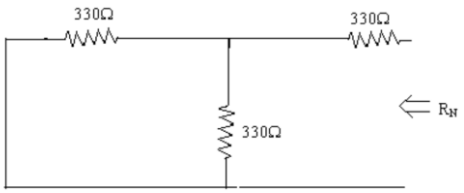
Circuit Diagram



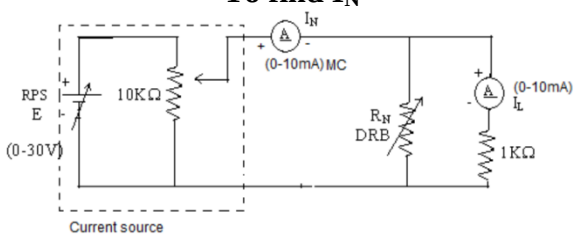
To find load current



To find I_N



To find R_N



Norton's equivalent circuit

Tabular Column:

Parameters		**Theoretical**	**Practical**
RPS in V			
I_N in A			
R_N in Ohms			
Load current (I_L)	**Original circuit**		
	Norton's Equivalent circuit		

Result:

Thus Norton's theorem was verified both practically and theoretically

Experiment 4:

DETERMINATION OF TIME RESPONSE SPECIFICATION OF A SECOND ORDER UNDER DAMPED SYSTEM, FOR DIFFERENT DAMPING FACTORS.

In this code, you can adjust the values of wn (natural frequency) and zeta (damping ratio) according to your system. The code calculates the step response of the system and determines the time response specifications such as time to peak, peak value, rise time, and settling time.

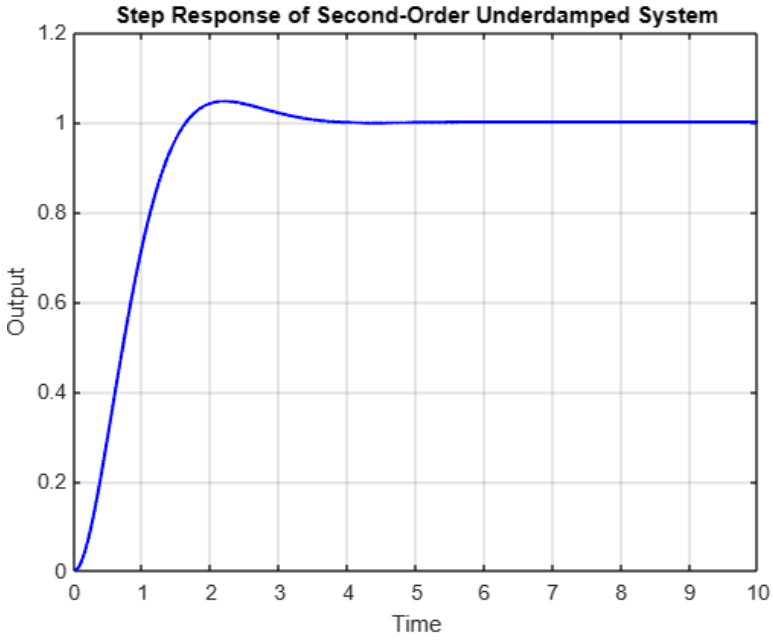
Matlab Code:

```
clc;
clear all;
close all;

% System parameters
wn = 2;      % Natural frequency
zeta = 0.7;  % Damping ratio

% Transfer function of the second-order system
num = wn^2;
den = [1 2*zeta*wn wn^2];
sys = tf(num, den);

% Step response of the system
t = 0:0.01:10;   % Time vector
[y, ~] = step(sys, t);

% Plotting the step response
plot(t, y, 'b', 'LineWidth', 1.5);
grid on;
xlabel('Time');
ylabel('Output');
title('Step Response of Second-Order Underdamped System');

% Determining time response specifications
[~, peak_idx] = max(y);
peak_time = t(peak_idx);     % Time to peak
peak_value = y(peak_idx);    % Peak value

% Time to rise from 10% to 90% of the peak value
rise_idx = find(y >= 0.1*peak_value & y <= 0.9*peak_value);
rise_time = t(rise_idx(end)) - t(rise_idx(1));

% Time to settle within 2% of the final value
settle_idx = find(y >= 0.98*y(end) & y <= 1.02*y(end));
settle_time = t(settle_idx(end)) - t(settle_idx(1));

% Displaying the time response specifications
disp('Time Response Specifications:');
```

```
disp(['Time to peak: ' num2str(peak_time)]);
disp(['Peak value: ' num2str(peak_value)]);
disp(['Rise time: ' num2str(rise_time)]);
disp(['Settling time: ' num2str(settle_time)]);
```

Result:

```
Time Response Specifications:
Time to peak: 2.2
Peak value: 1.046
Rise time: 1.16
Settling time: 8.44
```



Experiment 5:

DETERMINATION OF FREQUENCY RESPONSE OF A SECOND ORDER SYSTEM

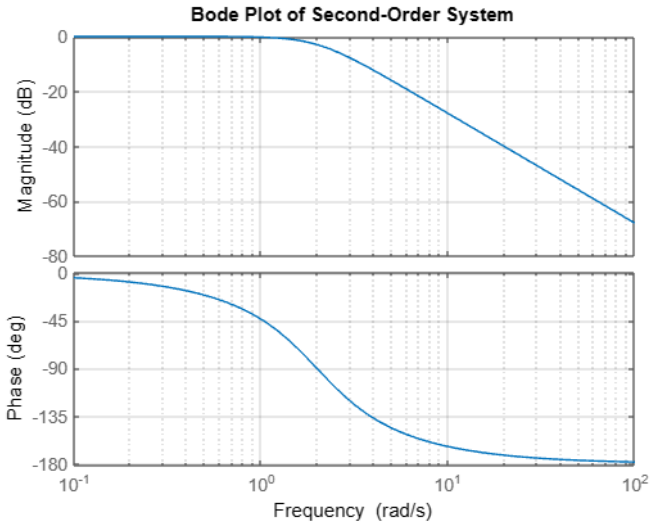
In this code, you can adjust the values of wn (natural frequency) and zeta (damping ratio) according to your system. The code calculates the frequency response of the system using the Bode plot. It generates the magnitude and phase response for a range of frequencies using the bode function. Then, it plots the magnitude response and phase response on logarithmic scales using the semilogx function.

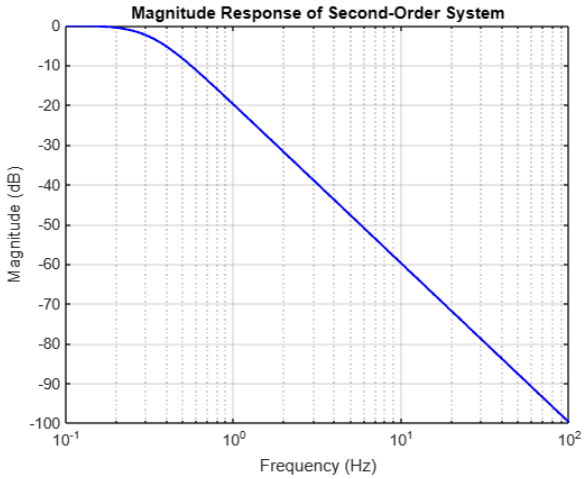
Matlab Code:

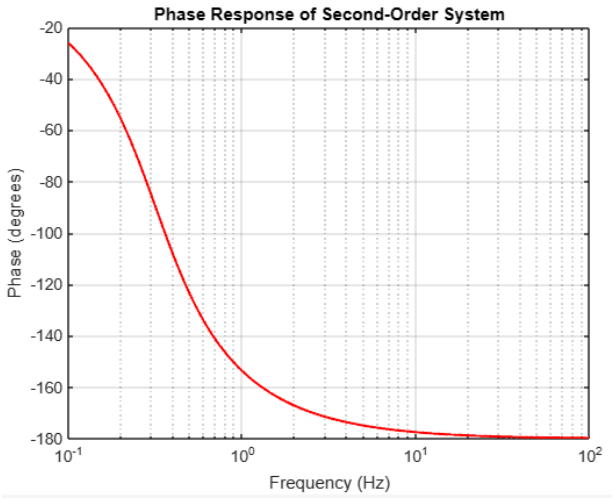
```
clc;
clear all;
close all;

% System parameters
wn = 2;      % Natural frequency
zeta = 0.7;  % Damping ratio


% Transfer function of the second-order system
num = wn^2;
den = [1 2*zeta*wn wn^2];
sys = tf(num, den);

% Frequency vector
f = logspace(-1, 2, 1000);  % Logarithmic scale from 0.1 to 100

% Bode plot of the system
figure;
bode(sys, {0.1, 100});
grid on;
title('Bode Plot of Second-Order System');

% Magnitude and phase response
[mag, phase] = bode(sys, 2*pi*f);  % Convert frequency to rad/s

% Plotting magnitude response
figure;
semilogx(f, 20*log10(squeeze(mag)), 'b', 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Response of Second-Order System');

% Plotting phase response
```

```
figure;
semilogx(f, squeeze(phase), 'r', 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Phase Response of Second-Order System');
```

Result:







Experiment 6:

DETERMINATIONS OF FREQUENCY RESPONSE OF A LEAD LAG COMPENSATOR

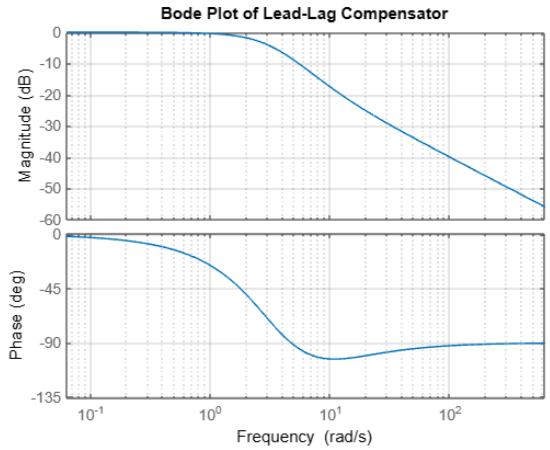
In this code, you can adjust the values of Kp (proportional gain), alpha (lead-lag parameter), T1 (lead time constant), and T2 (lag time constant) according to your compensator design. The code calculates the frequency response of the lead-lag compensator using the Bode plot. It generates the magnitude and phase response for a range of frequencies using the bode function. Then, it plots the magnitude response and phase response on logarithmic scales using the semilogx function.

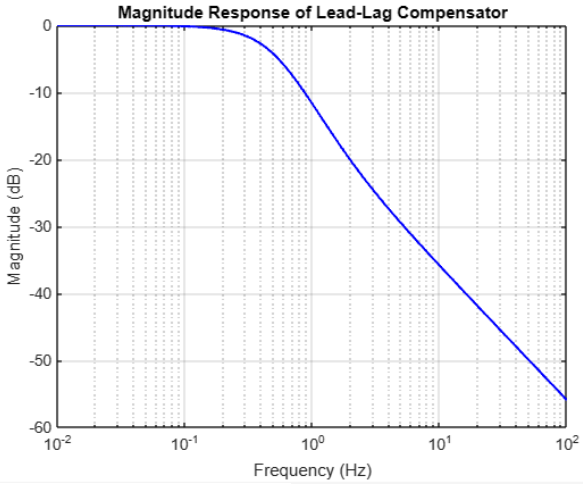
Matlab Code:

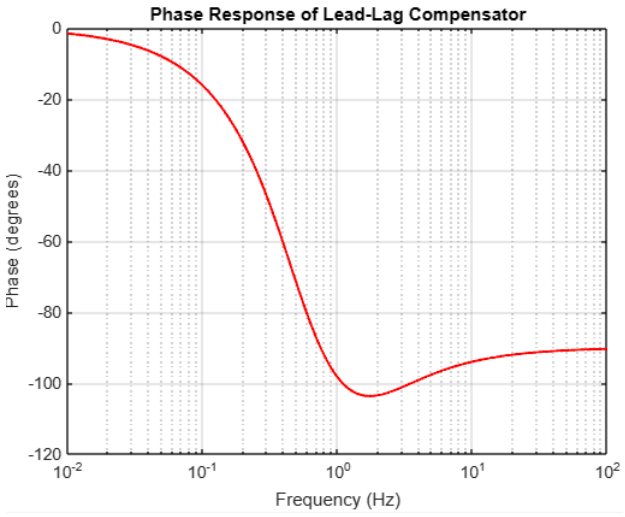
```
clc;
clear all;
close all;

% Lead-Lag Compensator parameters
Kp = 1;          % Proportional gain
alpha = 0.5;     % Lead-Lag parameter
T1 = 0.1;        % Lead time constant
T2 = 1;          % Lag time constant

% Transfer function of the Lead-Lag compensator
num = [Kp*T1 Kp];
den = [T1*T2 alpha*(T1+T2) 1];
sys = tf(num, den);

% Frequency vector
f = logspace(-2, 2, 1000);  % Logarithmic scale from 0.01 to 100

% Bode plot of the compensator
figure;
bode(sys, 2*pi*f);   % Convert frequency to rad/s
grid on;
title('Bode Plot of Lead-Lag Compensator');

% Magnitude and phase response
[mag, phase] = bode(sys, 2*pi*f);  % Convert frequency to rad/s

% Plotting magnitude response
figure;
semilogx(f, 20*log10(squeeze(mag)), 'b', 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
```

```
title('Magnitude Response of Lead-Lag Compensator');

% Plotting phase response
figure;
semilogx(f, squeeze(phase), 'r', 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Phase (degrees)');
title('Phase Response of Lead-Lag Compensator');
```

Result:







Experiment 7:

USING SUITABLE SIMULATION PACKAGE STUDY OF SPEED CONTROL OF DC MOTOR USING

i) Armature control ii) Field control

i. Armature control

In this code, you can adjust the parameters of the DC motor (resistance, inductance, moment of inertia, and damping coefficient, motor constant and back emf constant) according to your motor specifications. The code creates a transfer function for the DC motor based on the parameters.

The speed control design uses a PID controller with proportional gain (Kp), integral gain (Ki), and derivative gain (Kd). The pid function is used to create the PID controller, and it is connected with the plant using the feedback function.

A step reference signal is generated, and the closed-loop system is simulated using the step function. The resulting speed response is plotted, and the steady-state error is calculated and displayed.

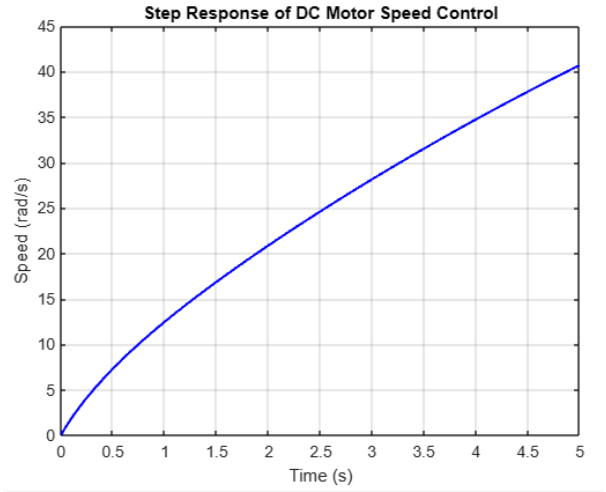
Matlab Code:

```
clc;
clear all;
close all;

% DC Motor Parameters
R = 1;       % Armature resistance (ohms)
L = 0.5;     % Armature inductance (H)
J = 0.01;    % Moment of inertia (kg.m^2)
b = 0.1;     % Damping coefficient (N.m.s)
Km = 0.01;   % Motor constant (N.m/A)
Kb = 0.01;   % Back emf constant (V.s/rad)

% Transfer function of the DC motor
num = Km;
den = [J*L (J*R + b*L) (b*R + Km^2)];
sys = tf(num, den);

% Speed Control Design
Kp = 1;      % Proportional gain
Ki = 1;      % Integral gain
Kd = 0.1;    % Derivative gain
Ts = 0.01;   % Sampling time

% Create a PID controller
controller = pid(Kp, Ki, Kd);

% Connect the controller with the plant
sys_cl = feedback(controller*sys, 1);
```

```
% Generate a step reference signal
t = 0:Ts:5;
ref = 100*ones(size(t));

% Simulate the closed-loop system
[y, t] = step(ref*sys_cl, t);

% Plot the step response
plot(t, squeeze(y),'b', 'LineWidth', 1.5);
grid on;
xlabel('Time (s)');
ylabel('Speed (rad/s)');
title('Step Response of DC Motor Speed Control');

% Display the steady-state error
ss_error = abs(y(end) - ref(end));
disp(['Steady-State Error: ' num2str(ss_error)]);
```

Result:

```
Steady-State Error: 59.3098
```



ii. Field control

In this code, you can adjust the parameters of the DC motor (resistance, inductance, moment of inertia, damping coefficient, motor constant, back emf constant, and field constant) according to your motor specifications. The code creates a transfer function for the DC motor based on the parameters.
The speed control design uses a PID controller with proportional gain (Kp), integral gain (Ki), and derivative gain (Kd). The pid function is used to create the PID controller, and it is connected with the plant using the feedback function.
A step reference signal is generated, and the closed-loop system is simulated using the step function. The resulting speed response is plotted, and the steady-state error is calculated and displayed.

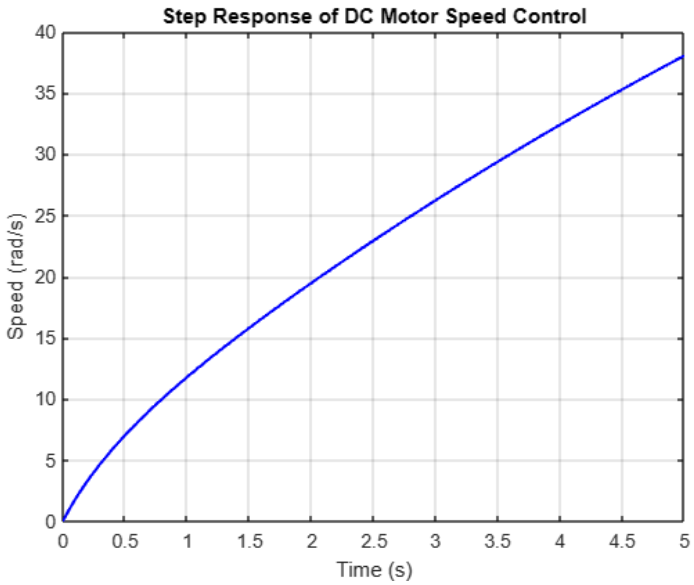
Matlab Code:

```
clc;
clear all;
close all;

% DC Motor Parameters
R = 1;       % Armature resistance (ohms)
L = 0.5;     % Armature inductance (H)
J = 0.01;    % Moment of inertia (kg.m^2)
b = 0.1;     % Damping coefficient (N.m.s)
Km = 0.01;   % Motor constant (N.m/A)
Kb = 0.01;   % Back emf constant (V.s/rad)
Kf = 0.1;    % Field constant (V.s)

% Transfer function of the DC motor
num = Km;
den = [J*L (J*R + b*L + J*Kf) (b*R + Km^2 + b*Kf)];
sys = tf(num, den);

% Speed Control Design
Kp = 1;       % Proportional gain
Ki = 1;       % Integral gain
Kd = 0.1;     % Derivative gain
Ts = 0.01;    % Sampling time

% Create a PID controller
controller = pid(Kp, Ki, Kd);

% Connect the controller with the plant
sys_cl = feedback(controller*sys, 1);
```

```
% Generate a step reference signal
t = 0:Ts:5;
ref = 100*ones(size(t));

% Simulate the closed-loop system
[y, t] = step(ref*sys_cl, t);

% Plot the step response
plot(t, y, 'b', 'LineWidth', 1.5);
grid on;
xlabel('Time (s)');
ylabel('Speed (rad/s)');
title('Step Response of DC Motor Speed Control');

% Display the steady-state error
ss_error = abs(y(end) - ref(end));
disp(['Steady-State Error: ' num2str(ss_error)]);
```

Result:

```
Steady-State Error: 61.965
```



Experiment 8:

USING SUITABLE SIMULATION PACKAGE, OBTAIN THE TIME RESPONSE FROM STATE MODEL OF A SYSTEM.

In this code, a state space model is defined using the ss function, specifying the state matrix A, input matrix B, output matrix C, and feedthrough matrix D.
Method 1 uses the initial function to calculate the time response of the system with an initial condition. It requires specifying the initial state vector x0 and the time vector t. The function returns the output y, time t, and state trajectory x. The code plots the output and state responses.
Method 2 uses the step function to calculate the time response of the system with a step input. It requires specifying the time vector t. The function returns the output y, time t, and state trajectory x. The code plots the output and state responses**.**

Matlab Code:

```
clc;
clear all;
close all;

% State Space Model
A = [0 1; -2 -3];   % State matrix
B = [0; 1];         % Input matrix
C = [1 0];          % Output matrix
D = 0;              % Feedthrough matrix

% Create state space model
sys = ss(A, B, C, D);

% Method 1: Calculate the time response using 'initial' function
x0 = [1; 0];          % Initial state vector
t = 0:0.01:5;         % Time vector
[y, t, x] = initial(sys, x0, t);  % Calculate time response

% Plot the time response
subplot(2,1,1);
plot(t, y, 'b', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Output');
title('Time Response - Initial Condition');
grid on;
subplot(2,1,2);
plot(t, x(:, 1), 'r', 'LineWidth', 1.5);
```

```
hold on;
plot(t, x(:, 2), 'g', 'LineWidth', 1.5);
xlabel('Time');
ylabel('State');
title('State Response - Initial Condition');
legend('x1', 'x2');
grid on;
```

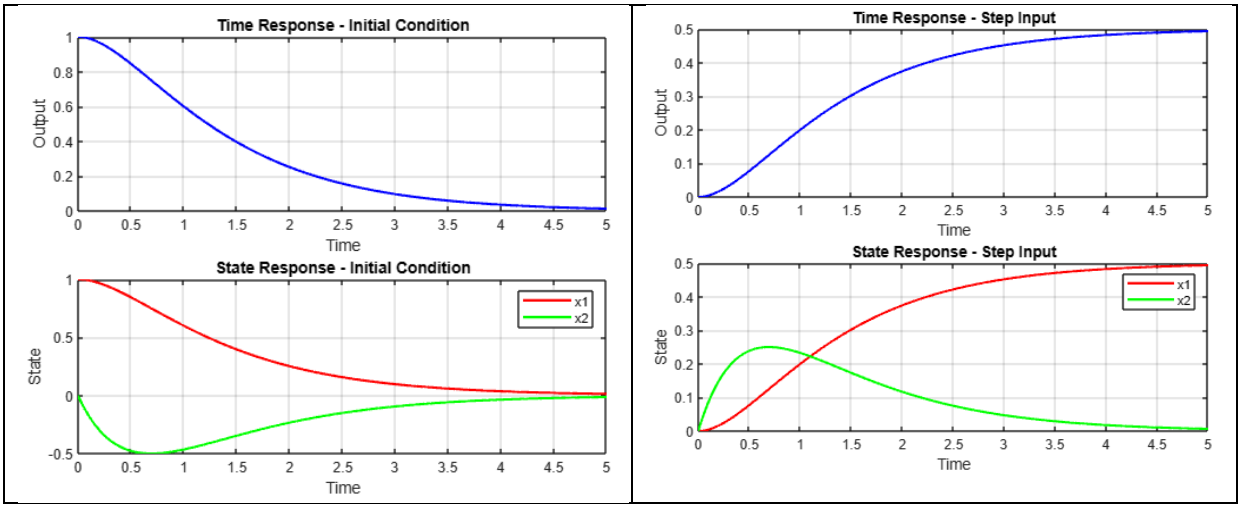
% Method 2: Calculate the time response using 'step' function

```
t = 0:0.01:5;         % Time vector
u = ones(size(t));    % Step input
[y, t, x] = step(sys, t);  % Calculate time response
```

% Plot the time response

```
figure;
subplot(2,1,1);
plot(t, y, 'b', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Output');
title('Time Response - Step Input');
grid on;
subplot(2,1,2);
plot(t, x(:, 1), 'r', 'LineWidth', 1.5);
hold on;
plot(t, x(:, 2), 'g', 'LineWidth', 1.5);
xlabel('Time');
ylabel('State');
title('State Response - Step Input');
legend('x1', 'x2');
grid on;
```

Result:



Experiment 9:

IMPLEMENTATION OF PI, PD CONTROLLERS

i. PI Controller

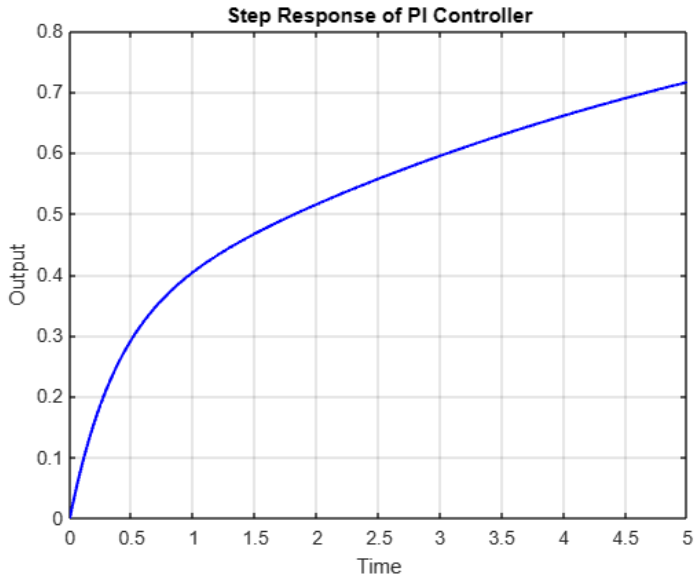
- In this code, you first define the transfer function of your plant system using the tf function. The num and den variables represent the numerator and denominator coefficients of the transfer function, respectively.
- Next, you specify the parameters of the PI controller, namely the proportional gain Kp and the integral gain Ki.
- Using the pid function, you create the PI controller by passing the Kp and Ki values as arguments.
- The feedback function is then used to connect the PI controller with the plant. The resulting closed-loop transfer function sys_cl represents the entire control system.
- You define a time vector t over which the step response will be evaluated.
- A step reference signal ref is created as a vector of ones.
- The step function is used to simulate the closed-loop system's response to the step reference signal. The output y and time vector t are obtained.
- Finally, the step response is plotted using the plot function.

Matlab Code:

```
clc;
clear all;
close all;

% Plant Transfer Function
num = [1];      % Numerator coefficients
den = [1 2];    % Denominator coefficients
plant = tf(num, den);   % Create transfer function

% PI Controller Parameters
Kp = 1;     % Proportional gain
Ki = 0.5;   % Integral gain

% Create PI controller
controller = pid(Kp, Ki);

% Connect the controller with the plant
sys_cl = feedback(controller * plant, 1);

% Time vector
t = 0:0.01:5;

% Step reference signal
ref = ones(size(t));
```

```
% Simulate the closed-loop system
[y, t] = step(ref * sys_cl, t);

% Plot the step response
plot(t, squeeze(y), 'b', 'LineWidth', 1.5);
grid on;
xlabel('Time');
ylabel('Output');
title('Step Response of PI Controller');
```

Result:



ii. PD Controller

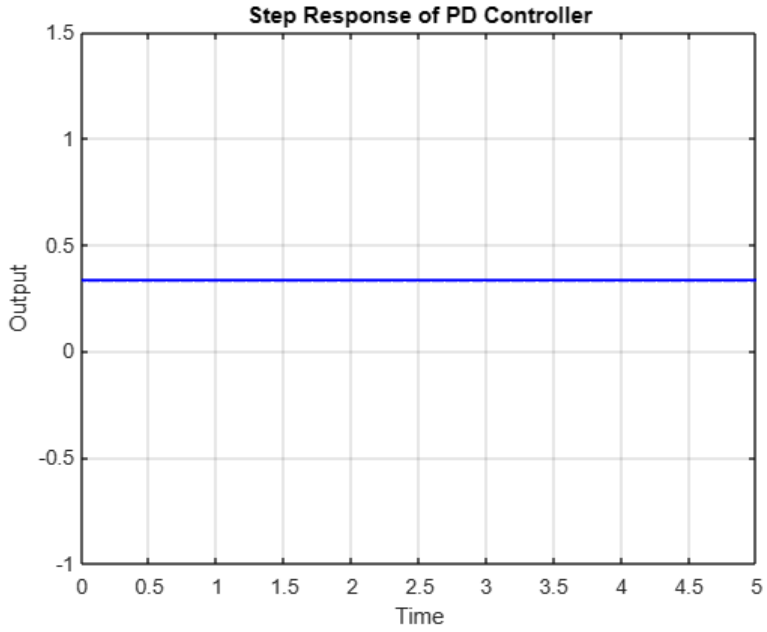
- In this code, you first define the transfer function of your plant system using the tf function. The num and den variables represent the numerator and denominator coefficients of the transfer function, respectively.
- Next, you specify the parameters of the PD controller, namely the proportional gain Kp and the derivative gain Kd. Note that the integral gain is set to 0.
- Using the pid function, you create the PD controller by passing the Kp and Kd values as arguments.
- The feedback function is then used to connect the PD controller with the plant. The resulting closed-loop transfer function sys_cl represents the entire control system.
- You define a time vector t over which the step response will be evaluated.
- A step reference signal ref is created as a vector of ones.
- The step function is used to simulate the closed-loop system's response to the step reference signal. The output y and time vector t are obtained.
- Finally, the step response is plotted using the plot function

Matlab Code:

```
clc;
clear all;
close all;

% Plant Transfer Function
num = [1];      % Numerator coefficients
den = [1 2];    % Denominator coefficients
plant = tf(num, den);   % Create transfer function

% PD Controller Parameters
Kp = 1;     % Proportional gain
Kd = 0.5;   % Derivative gain

% Create PD controller
controller = pid(Kp, 0, Kd);

% Connect the controller with the plant
sys_cl = feedback(controller * plant, 1);

% Time vector
t = 0:0.01:5;

% Step reference signal
ref = ones(size(t));

% Simulate the closed-loop system
[y, t] = step(ref * sys_cl, t);
```

```
% Plot the step response
plot(t, squeeze(y), 'b', 'LineWidth', 1.5);
grid on;
xlabel('Time');
ylabel('Output');
title('Step Response of PD Controller');
```

Result:



Experiment 10:

Implement a PID Controller and hence realize an Error Detector.

In this code, the PID controller parameters (proportional, integral, and derivative gains) are specified. The desired setpoint and initial conditions are also defined. The code then performs a loop over the time steps, where the error between the setpoint and the current value is calculated. The proportional, integral, and derivative terms are computed based on the error and the gains. The PID output is obtained by summing the three terms. Finally, the current value is updated based on the output, and the process is repeated for each time step.

Matlab Code:

```
clc;
clear all;
close all;

% PID Controller parameters
Kp = 1.0;   % Proportional gain
Ki = 0.5;   % Integral gain
Kd = 0.2;   % Derivative gain

% Desired setpoint and initial conditions
setpoint = 10.0;    % Desired value
initial_value = 0;  % Initial value

% Time parameters
dt = 0.1;       % Time step
simulation_time = 10.0;   % Total simulation time
t = 0:dt:simulation_time;   % Time vector

% Initialize variables
error = zeros(size(t));
integral = zeros(size(t));
derivative = zeros(size(t));
output = zeros(size(t));
current_value = initial_value;

% PID Controller loop
```

```
for i = 2:length(t)
    % Error calculation
    error(i) = setpoint - current_value;

    % Proportional term
    p = Kp * error(i);

    % Integral term
    integral(i) = integral(i-1) + Ki * error(i) * dt;
    i_term = integral(i);

    % Derivative term
    derivative(i) = (error(i) - error(i-1)) / dt;
    d_term = Kd * derivative(i);

    % PID output
    output(i) = p + i_term + d_term;

    % Update current value (e.g., using a system model)
    % current_value = system_update(output(i));

    % In this example, we assume a simple integrator model
    current_value = current_value + output(i) * dt;
end

% Plotting the results
figure;
subplot(2,1,1);
plot(t, output, 'b', 'LineWidth', 1.5);
title('PID Controller Output');
xlabel('Time');
ylabel('Output');
grid on;
```

```
subplot(2,1,2);
plot(t, error, 'r', 'LineWidth', 1.5);
title('Error');
xlabel('Time');
ylabel('Error');
grid on;
```

Result: