Netaji Subhas University of Technology

Fourth Semester

ICE-1

Batch 2022-26

Engineering Analysis and Design

(ICICC10)

Practical Lab File

Submitted To
Mrs.Anuradha Tomar

Submitted By
Devansh Behl
2022UIC3582
ICE-1

Index

MATLAB Experiments

Experiment-1

Aim

To simulate the output waveform of the RLC series circuit using MATLAB.

Software/Hardware Used

Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

The RLC (Resistor-Inductor-Capacitor) circuit is a fundamental circuit widely used in digital electronics and various other applications. In Simulink, the RLC circuit is composed of three components: an inductor (L), a resistor (R), and a capacitor (C). These components are interconnected in series, with the inductor connected across the voltage source and the resistor and capacitor connected to ground.

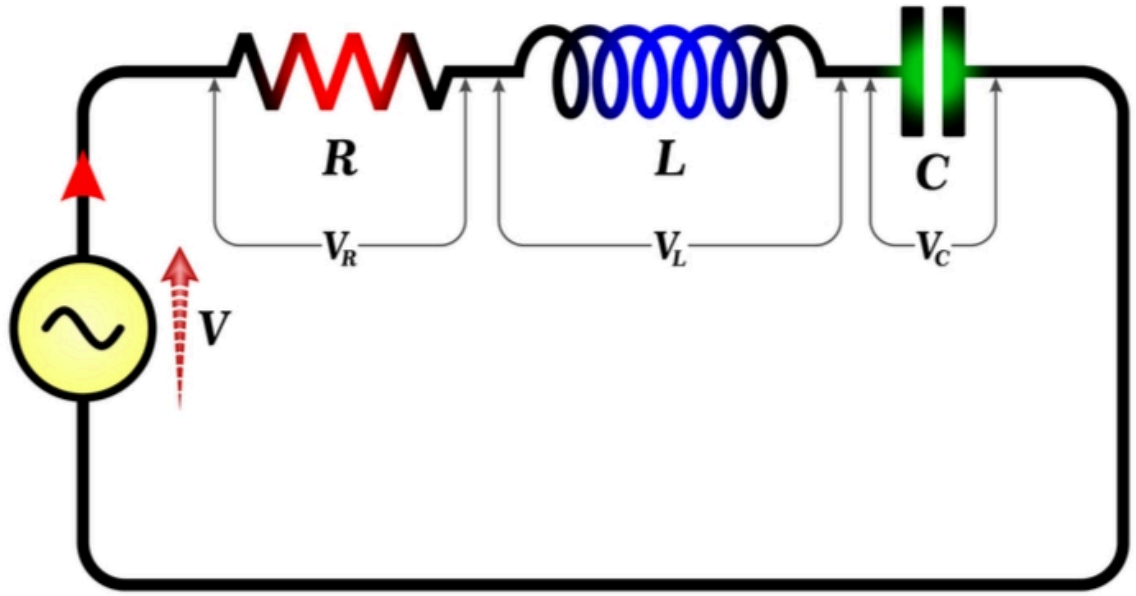
This circuit configuration allows the RLC circuit to store and release electrical energy, making it versatile for various applications. In practical scenarios, RLC circuits can model a wide range of systems, including lamps, electric motors, and other loads commonly encountered in electronic circuits.

The specific values assigned to the components in this RLC circuit example are 10nF for capacitance, 100mH for inductance, and 1k for resistance. These parameters define the behavior and characteristics of the circuit, influencing its performance under different conditions.

The RLC circuit exhibits oscillatory behavior, characterized by the periodic exchange of energy between the inductor and capacitor. The frequency of oscillation, given as 1kHz, and the time period, given as 10s, determine the rate at which energy is transferred within the circuit.

In the given scenario, with no current flowing through the circuit (0A), the voltage across the capacitor can be calculated using the formula $V = I * t / C$, where V is voltage, I is current, t is time, and C is capacitance. Since the current is zero, the voltage across the capacitor will be zero as well.

Understanding the behavior of RLC circuits is essential for designing and analyzing electronic systems, as they play a crucial role in energy storage, signal filtering, and frequency response. By simulating RLC circuits in tools like Simulink, engineers can explore their behavior, optimize parameters, and ensure the efficient operation of electronic devices and systems.



(Fig-1.1-Image of RLC Circuit)

The advantages of this Simulink RLC circuit are:

-It can be used to measure the electric power

-It is a very basic circuit that is quite easy to understand

-It can be used to find the RLC circuit's transient response

Procedure

• Open MATLAB and create a new Simulink model.

• From the Simulink library browser, drag the "AC voltage source",” Voltage measurement”, “Current measurement” and “RLC circuit” block to the model.

• Double-click the blocks and set the resistance, capacitance, and AC voltage source values.

• From the "Sinks" library, drag the "Scope" block to the model and connect it as shown in the figure.

• Save the model and run the simulation.

Precautions-

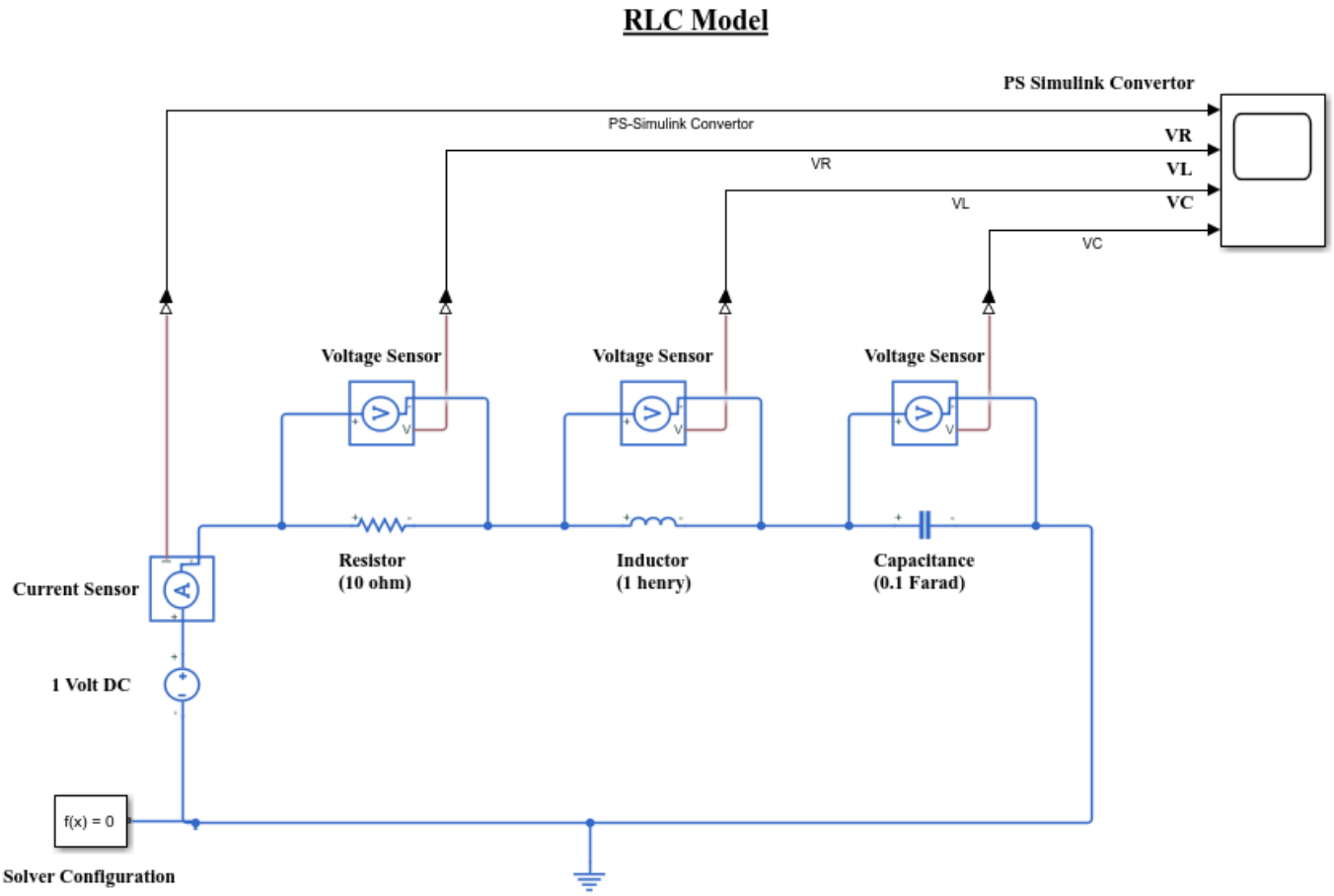
• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

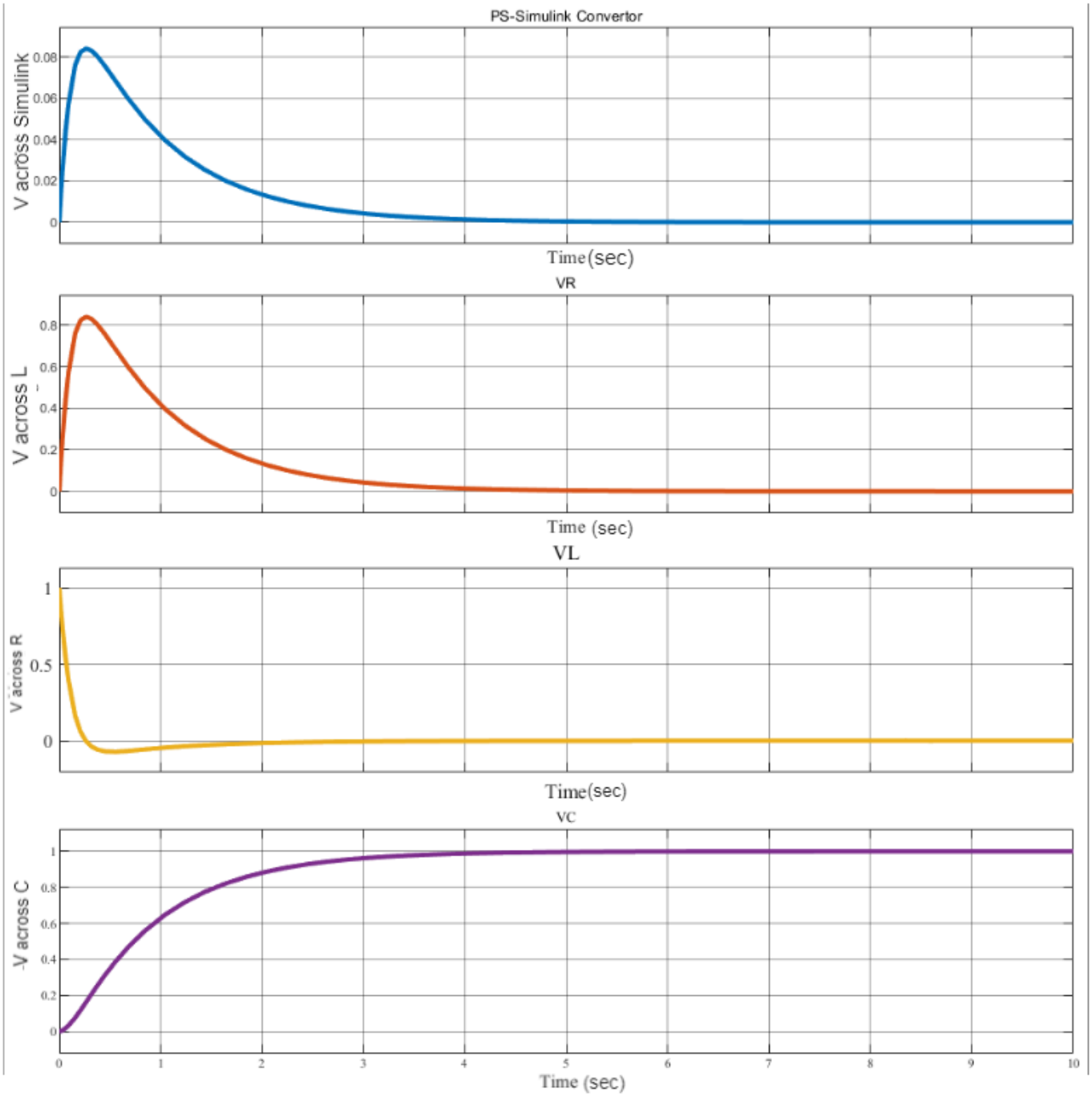
• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit



(Fig-1.2-Simulation circuit of RLC Circuit)

Output Graph



(Fig-1.3-Output Graph of RLC Circuit)

Experiment-2

Aim

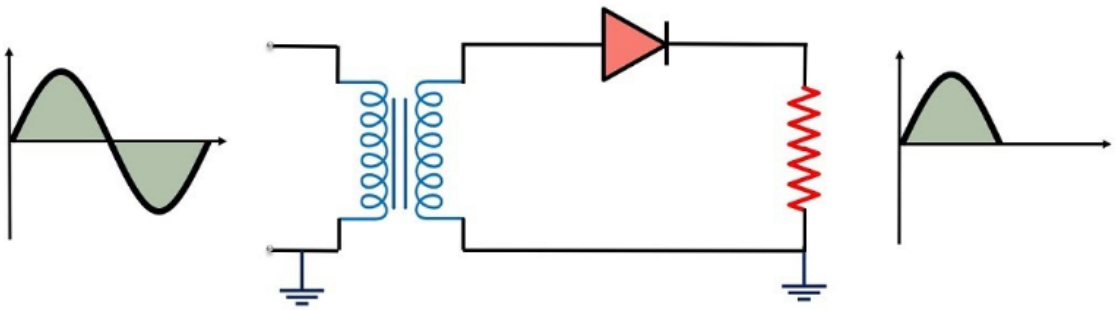
To Simulate the waveform of a half-wave controlled and uncontrolled rectifier using MATLAB.

Software/Hardware Used

Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

A half-wave rectifier circuit converts alternating current (AC) voltage to direct current (DC) by allowing only one half-cycle of the AC waveform to pass through while blocking the other half. This basic rectifier employs just one diode, making it simple and cost-effective. The components of a half-wave rectifier include a diode, which permits current flow in one direction, a voltage source supplying the AC input, and a resistive load to receive the rectified DC output. During the positive half-cycle of the AC input, the diode conducts, enabling current to flow through the load. Conversely, during the negative half-cycle, the diode blocks current. As a result, the output across the load exhibits pulsating DC voltage, with peaks occurring only during the positive half-cycles of the input waveform. Although efficient for certain applications, half-wave rectifiers are less utilized than full-wave rectifiers due to their lower efficiency and higher ripple voltage.



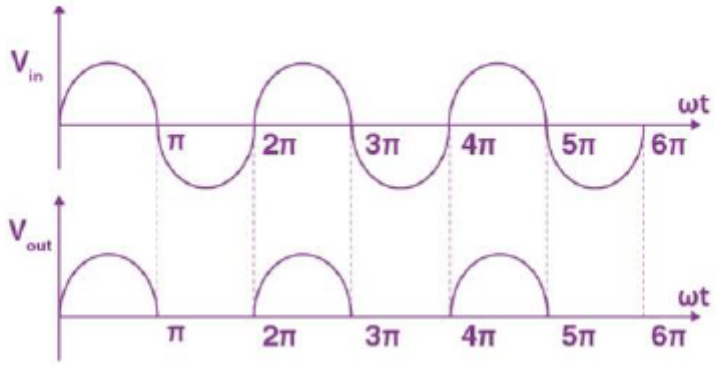
(Fig-2.1-Half Wave Rectifier Diagram)

A half-wave rectifier circuit consists of three main components as follows:

1. A diode
2. A Voltage source
3. A resistive load

Half Wave Rectifier Waveform

The half wave rectifier waveform before and after rectification is shown below in the figure.



(Fig-2.2-Half Wave Rectifier Waveform)

Procedure

• Open a new Simulink model and save it with a suitable name.

• Add a Sine Wave block from the Simulink Sources library to the model. Double-click on the block to set the parameters of the input waveform such as amplitude, frequency, and phase.

• Add a Simscape Electrical Specialized Power Systems Bridge Rectifier block from the Simscape Electrical library to the model. Double-click on the block to set the circuit parameters such as the load resistance, filter capacitor, and diode parameters.

• Connect the output of the Sine Wave block to the input of the Bridge Rectifier block.

• Add a Scope block from the Simulink Sinks library to the model to visualize the output waveform. Connect the output of the Bridge Rectifier block to the input of the Scope block.

• Run the simulation and observe the output waveform on the Scope block. You can also use the Scope block's measurement features to measure the DC component and ripple voltage of the output waveform.

Precautions-

• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

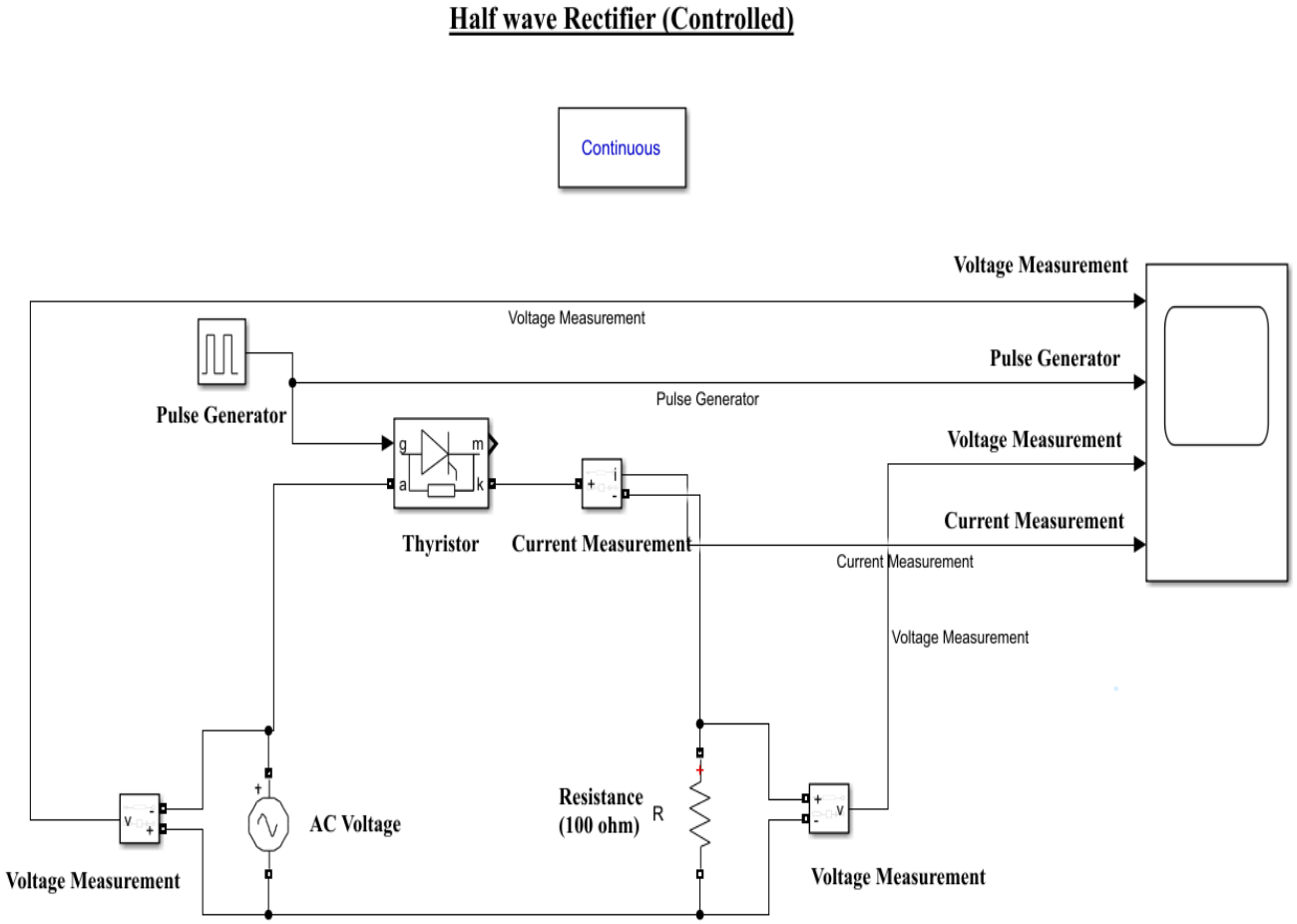
• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

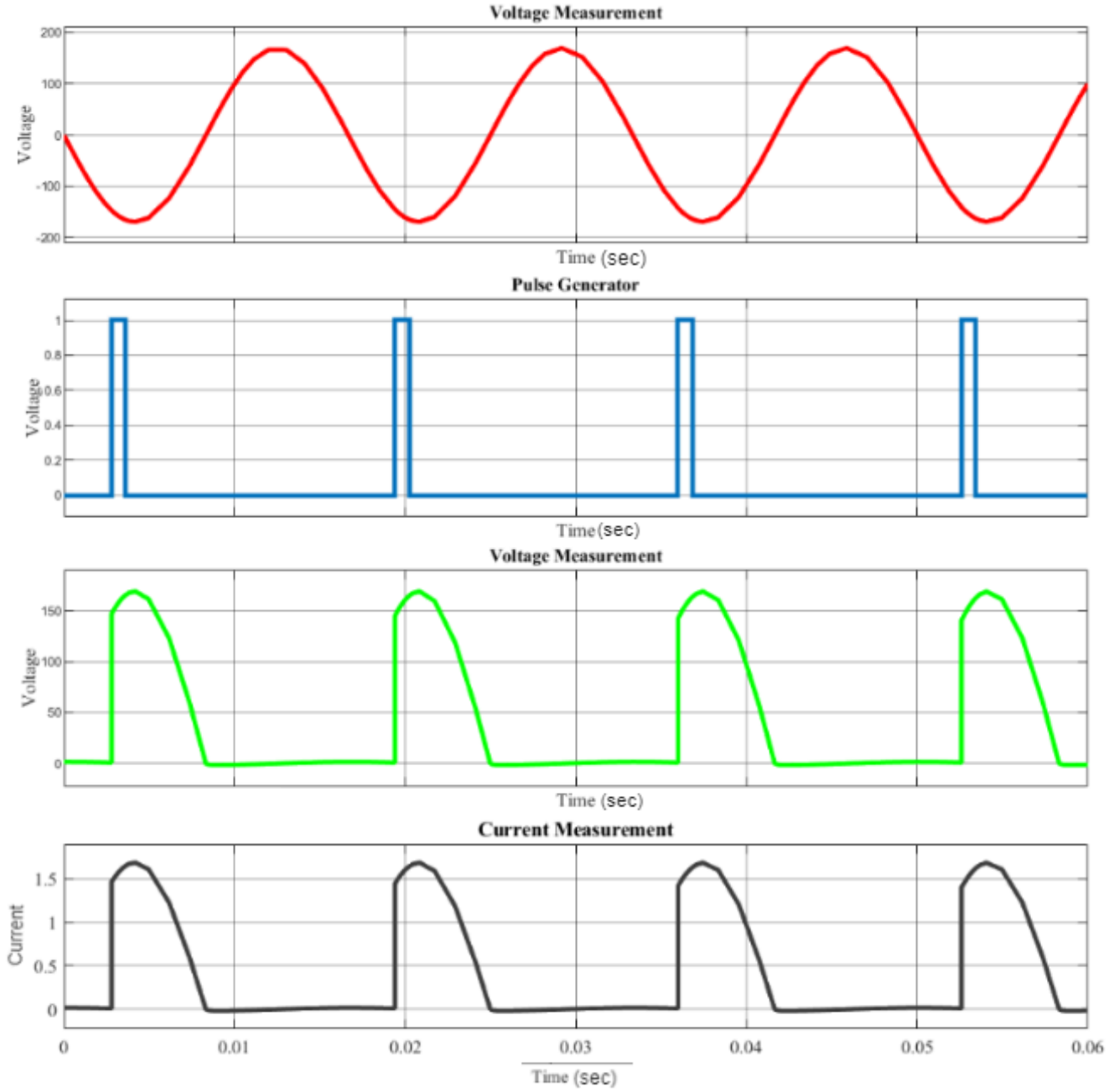
Simulation Circuit

(a) Controlled Rectifier



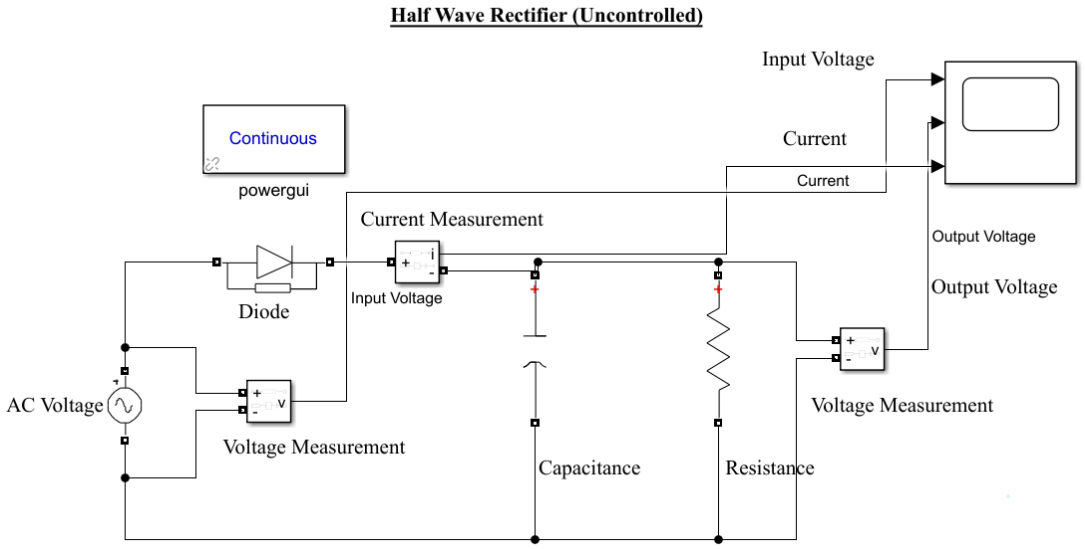
(Fig-2.3-Controlled Half Wave Rectifier Simulation Circuit)

Output Graph



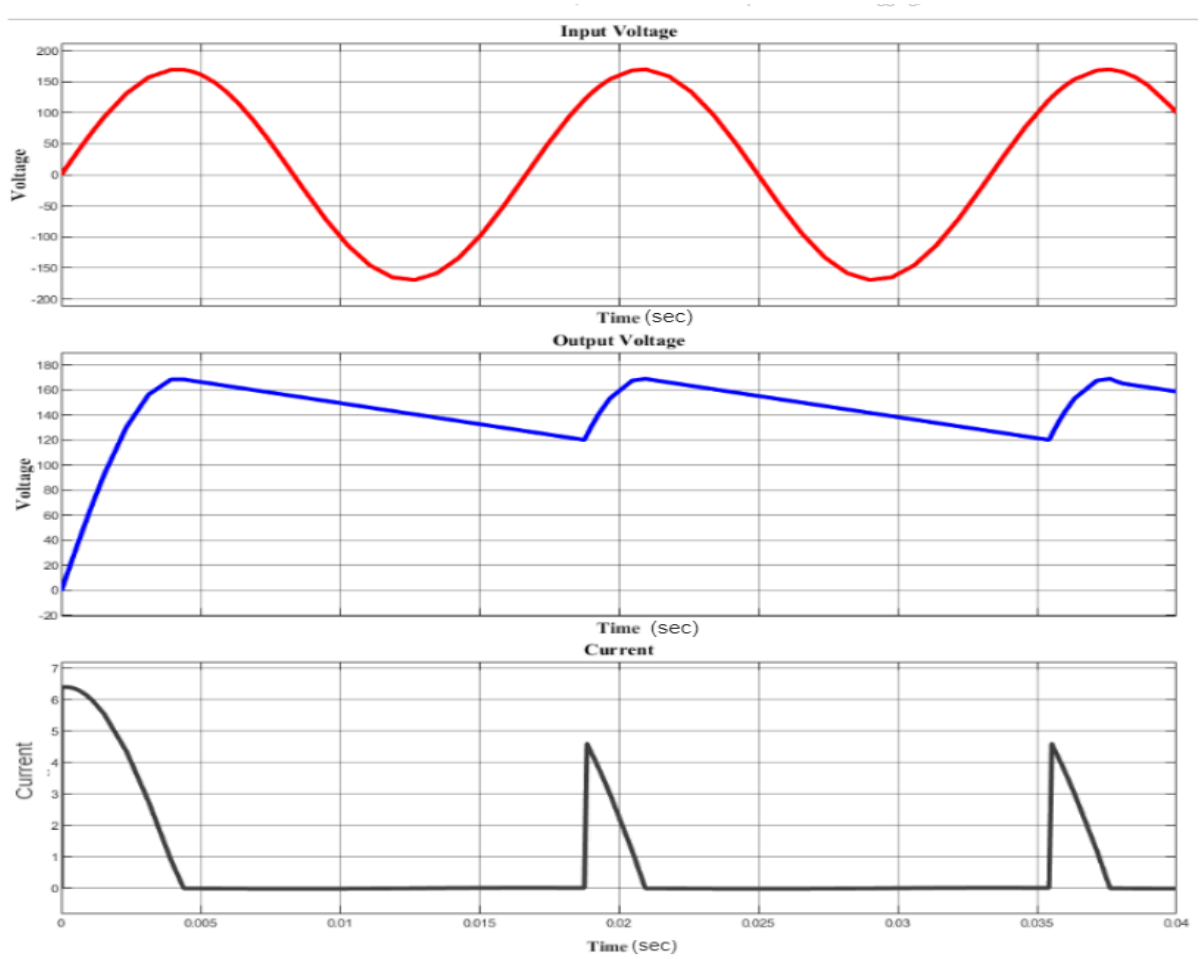
(Fig-2.4-Controlled Half Wave Rectifier Simulation Waveform)

(b) Uncontrolled Rectifier



(Fig-2.5-Uncontrolled Half Wave Rectifier Simulation Circuit)

Output Graph



(Fig-2.6-Uncontrolled Half Wave Rectifier Simulation Waveform)

Experiment-3

Aim

To Simulate the waveform of full wave controlled and uncontrolled rectifier using MATLAB.

Software/Hardware Used

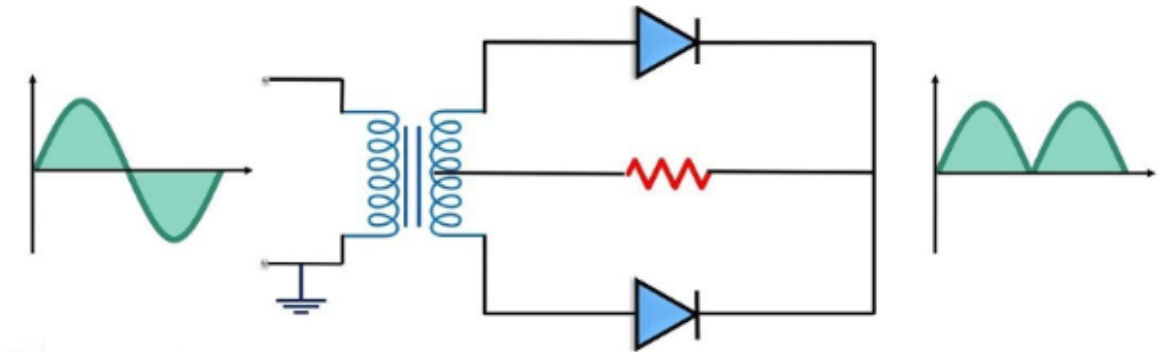
Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

A full wave rectifier is a crucial electronic circuit employed in converting alternating current (AC) voltage into direct current (DC) voltage. Its primary function is to allow current flow in only one direction, either positive or negative, during each half cycle of the AC voltage, resulting in a unidirectional flow of current. Unlike half-wave rectifiers, which utilize only one half of the AC waveform, full-wave rectifiers harness both positive and negative cycles, thereby doubling the efficiency and producing a smoother DC output.

Among full-wave rectifiers, the most prevalent type is the bridge rectifier. This configuration employs four diodes arranged in a bridge pattern. During the positive half cycle of the AC voltage, two diodes conduct, enabling current to flow through the load in one direction. Similarly, during the negative half cycle, the other two diodes conduct, facilitating current flow in the opposite direction. This alternating conduction of diodes ensures that the load receives a continuous supply of current, resulting in a more efficient rectification process compared to half-wave rectifiers.

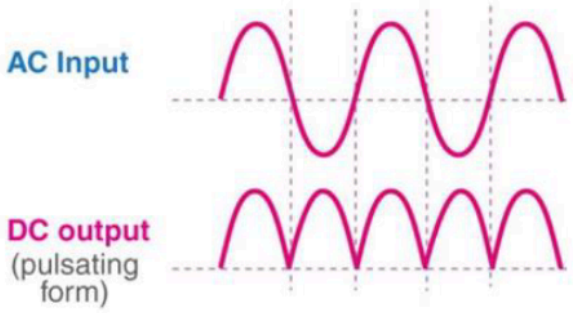
The output of a full wave rectifier exhibits a pulsating DC waveform, containing both positive and negative voltage peaks. However, to obtain a smoother DC output suitable for most applications, additional filtering is required. This is typically achieved using a filter circuit comprising capacitors or inductors. These components help reduce the ripple voltage present in the output, ensuring a more stable and constant DC voltage supply.



(Fig-3.1-Full Wave Rectifier Diagram)

Full wave rectifiers find extensive application in various fields, especially in power supplies for electronic devices. They are commonly used in devices such as laptops, mobile phones, televisions, and other electronic gadgets that require a steady DC voltage from an AC power source. Additionally, full-wave rectifiers are employed in industrial applications, such as motor control systems, lighting systems, and battery charging circuits, where efficient conversion of AC to DC voltage is essential for proper operation.

Full Wave Rectifier Waveform



(Fig-3.2-Full Wave Rectifier Waveform)

In summary, full wave rectifiers, particularly bridge rectifiers, play a pivotal role in converting AC voltage to DC voltage efficiently. By utilizing both halves of the AC waveform, they offer higher efficiency and smoother output compared to half-wave rectifiers. With their versatility and widespread application, full-wave rectifiers serve as indispensable components in modern electronic systems, ensuring reliable and stable DC power supplies for various devices and equipment.

Procedure:

• Open a new Simulink model and save it with a suitable name.

• Add a Sinusoidal signal source from the Simulink Sources library to the model. Double-click on the block to set the parameters of the input waveform such as amplitude, frequency, and phase.

• Add a bridge rectifier circuit from the Simscape library to the model. Double-click on the block to set the circuit parameters such as the load resistance, filter capacitor, and diode parameters.

• Connect the output of the sinusoidal signal source block to the input of the bridge rectifier circuit block.

• Add a scope from the Simulink Sinks library to the model to visualize the output waveform.Connect the output of the bridge rectifier circuit block to the input of the scope block.

• Run the simulation and observe the output waveform on the scope block. You can also use the Scope block's measurement features to measure the DC component and ripple voltage of the output waveform.

Precautions-

• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

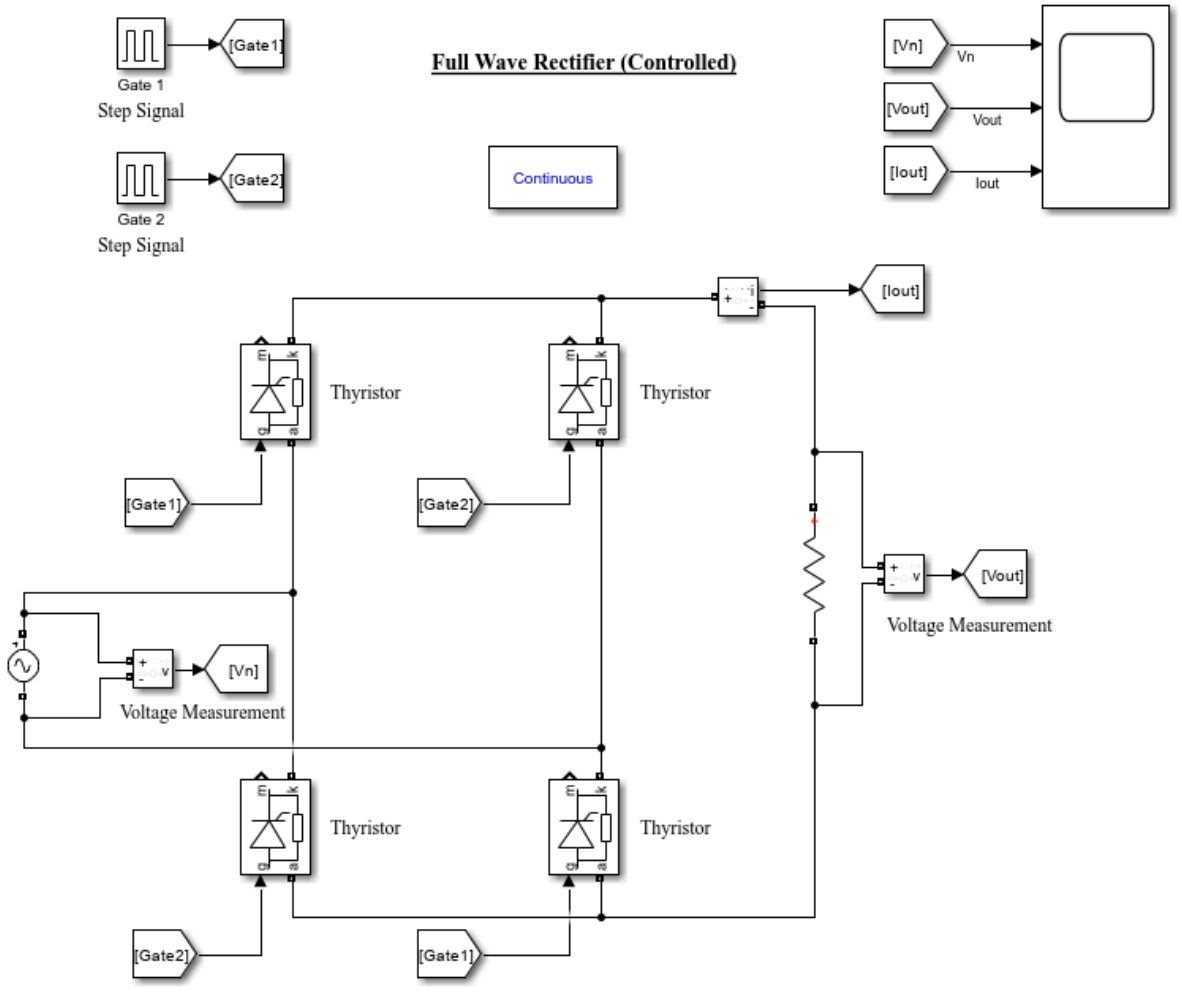
• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

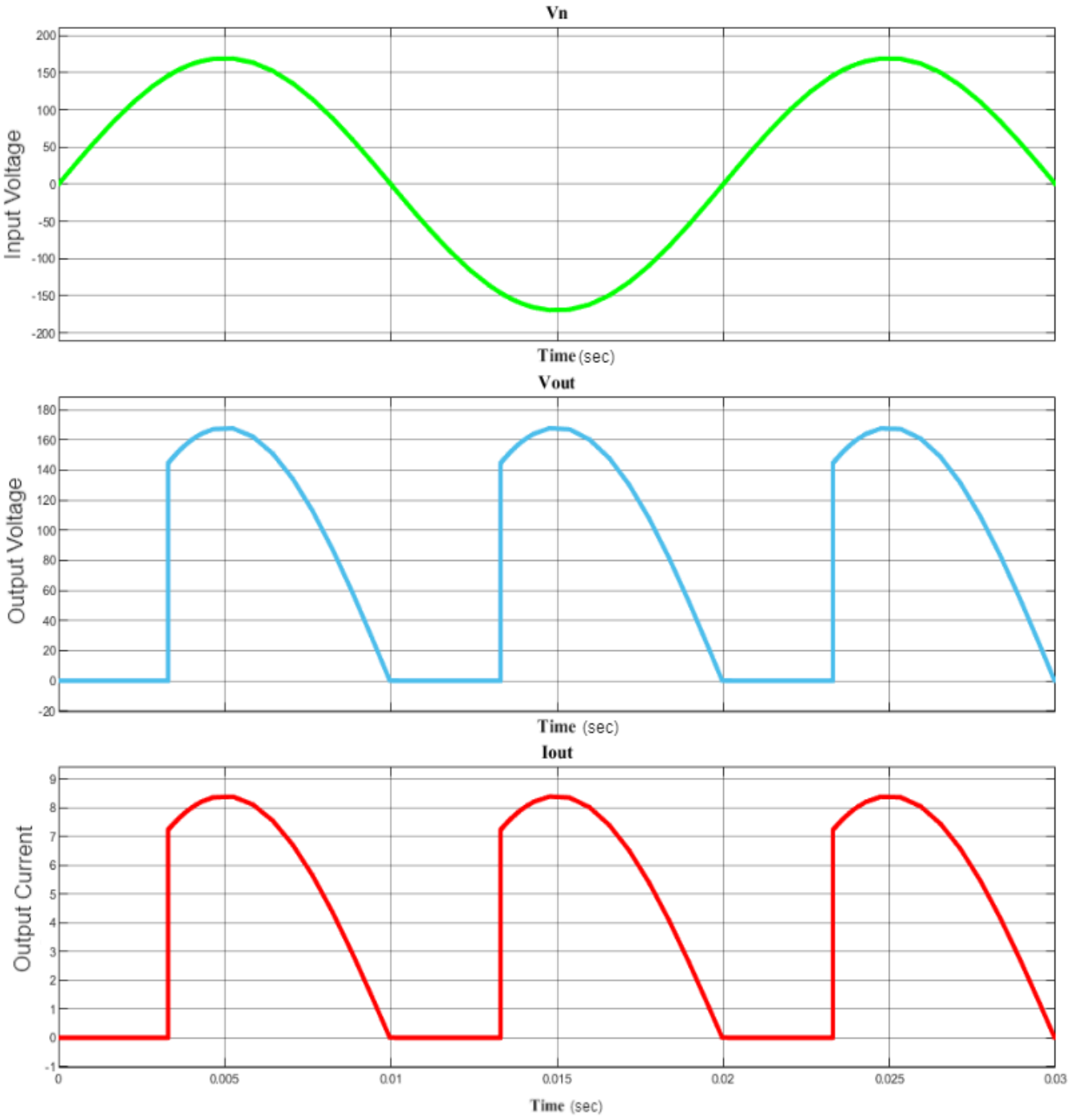
Simulation Circuit

(a)Controlled Rectifier



(Fig-3.3-Full Wave Controlled Rectifier Circuit)

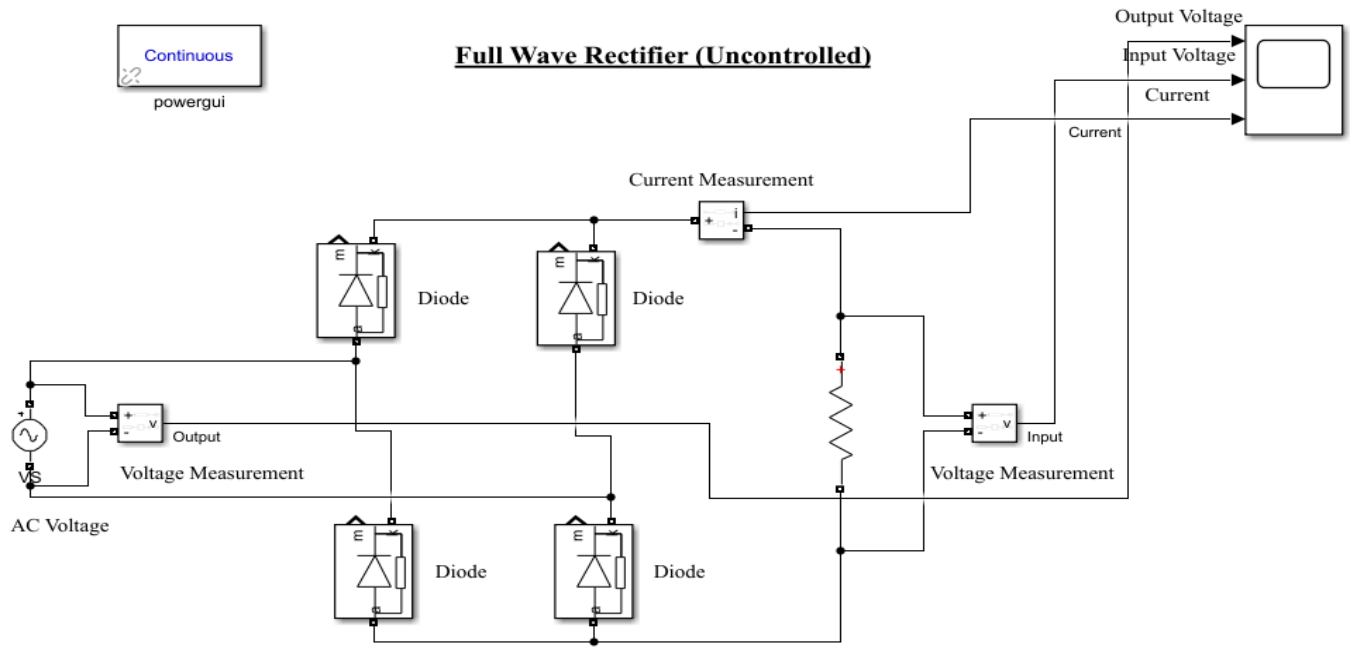
Output Graph



(Fig-3.4-Full Wave Controlled Rectifier Waveform)

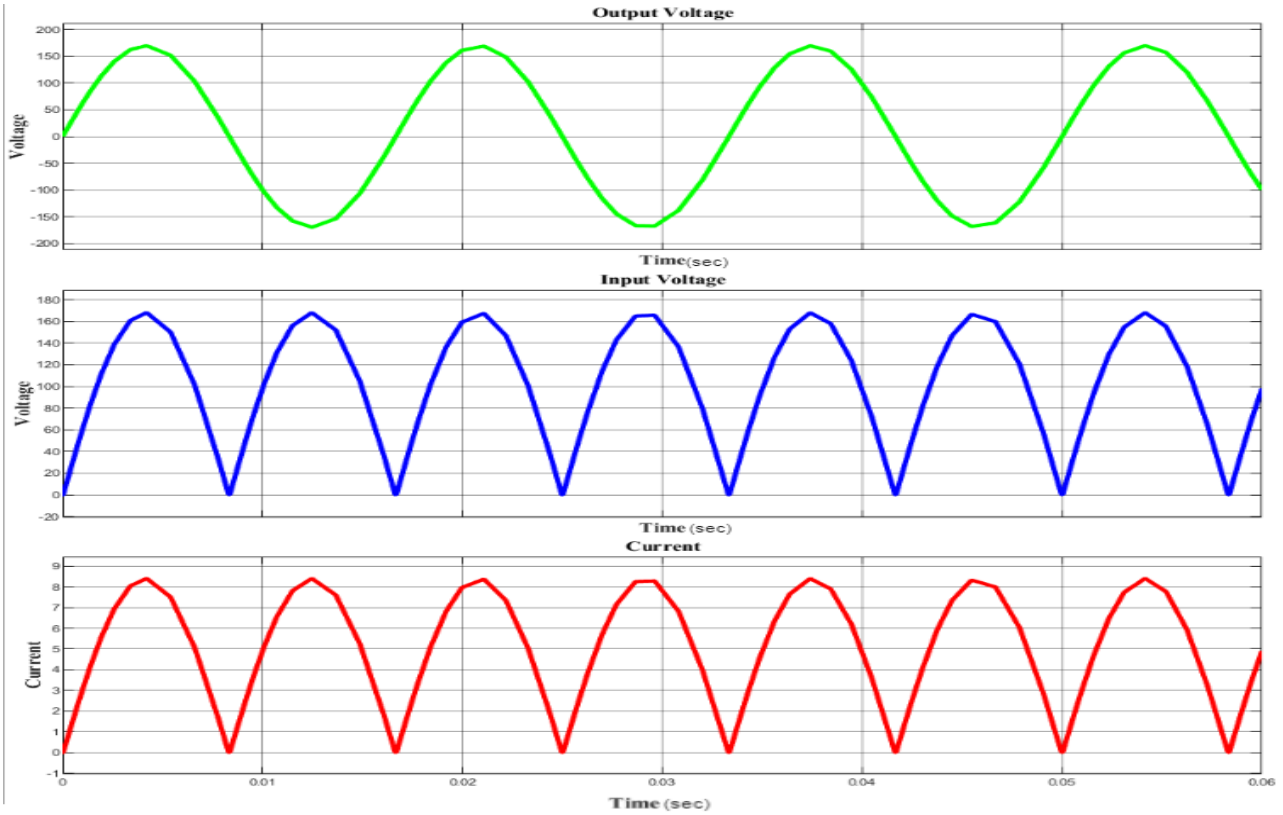
Simulation Circuit

(a)Uncontrolled Rectifier



(Fig-3.5-Full Wave Uncontrolled Rectifier Circuit)

Output Graph



(Fig-3.6-Full Wave Uncontrolled Rectifier Waveform)

Experiment-4

Aim

To develop mathematical and Simulink models for a Mass-Spring-Damper system and observe the step response

Software/Hardware Used

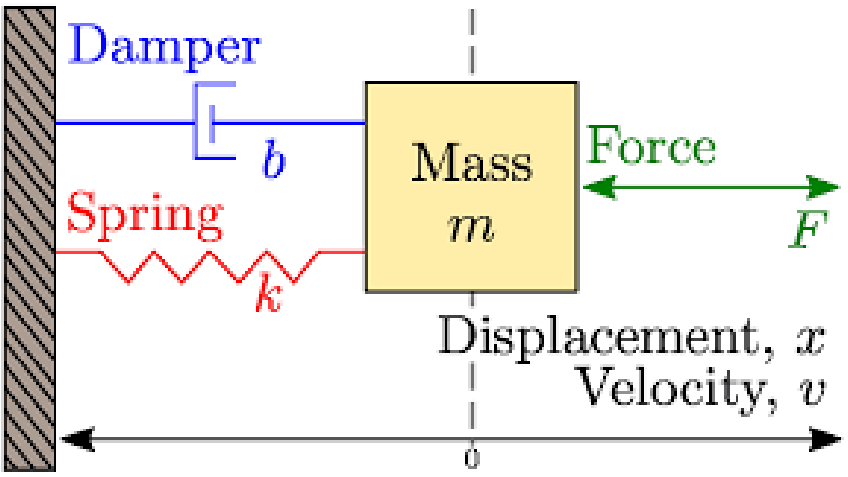
Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

The mass-spring-damper model is a fundamental mechanical system used to describe the dynamic behavior of objects subjected to external forces. It comprises discrete mass nodes distributed within an object and interconnected through a network of springs and dampers. Each mass node represents a localized portion of the object, while the springs and dampers simulate the mechanical connections between these nodes.

In this model, the masses represent the inertial properties of the object, with their mass values determining how they respond to applied forces. The springs mimic the elastic properties of the material, providing resistance to deformation and restoring forces when displaced from equilibrium. The dampers, on the other hand, introduce a damping effect that dissipates energy and reduces oscillations within the system.

Deriving the equations of motion for this model is usually done by examining the sum of forces on the mass:



(Fig-4.1-Mass Spring Damper Diagram)

The mass-spring-damper model is particularly suitable for simulating objects with complex material properties, such as nonlinearity and viscoelasticity. Nonlinearity refers to materials whose mechanical response deviates from linear behavior, often exhibiting phenomena like hysteresis or nonlinear stiffness. Viscoelastic materials display both viscous (damping) and elastic (spring-like) characteristics, causing time-dependent deformation under applied loads.

$$\Sigma F = -kx - c\dot{x} + F_{external} = m\ddot{x}$$

By rearranging this equation, we can derive the standard form:[3]

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = u \text{ where } \omega_n = \sqrt{\frac{k}{m}}; \quad \zeta = \frac{c}{2m\omega_n}; \quad u = \frac{F_{\text{external}}}{m}$$

ω_n is the undamped natural frequency and ζ is the damping ratio. The homogeneous equation for the mass spring system is:

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0$$

This has the solution:

$$x = Ae^{-\omega_n t\left(\zeta+\sqrt{\zeta^2-1}\right)} + Be^{-\omega_n t\left(\zeta-\sqrt{\zeta^2-1}\right)}$$

If $\zeta < 1$ then $\zeta^2 - 1$ is negative, meaning the square root will be negative the solution will have an oscillatory component.

By incorporating these elements into the model, engineers can accurately simulate the dynamic behavior of objects subjected to various loading conditions. This allows for the prediction of how the object will respond to external forces, enabling the optimization of design parameters and the assessment of structural integrity.

Furthermore, the mass-spring-damper model finds widespread application across numerous fields, including mechanical engineering, civil engineering, robotics, and biomechanics. It serves as the foundation for more complex dynamic simulations, such as finite element analysis (FEA) and multi-body dynamics (MBD), providing valuable insights into the behavior of real-world systems and aiding in the development of innovative technologies.

Procedure

• Open MATLAB and navigate to the Simulink Library Browser.

• In the Simulink Library Browser, search for the "Simulink" library and expand it.

• From the Simulink library, drag and drop the "Scope" block onto the blank Simulink model.

• From the "Sources" library, drag and drop the "Step" block onto the Simulink model.

• From the "Continuous" library, drag and drop the "Transfer Fcn" block onto the Simulink model.

• Double-click the "Transfer Function" block to open its parameters and set the numerator to "1" and the denominator to "[m, b, k]" where m, b, and k are the mass, damping coefficient, and spring constant, respectively.

• Connect the output of the "Step" block to the input of the "Transfer Fcn" block.

• Connect the output of the "Transfer Fcn" block to the input of the "Scope" block.

• Double-click the "Scope" block to open its parameters and set the "Number of input ports" to "1" and the "Number of traces" to "1".

• Save the Simulink model and run the simulation.

Precautions-

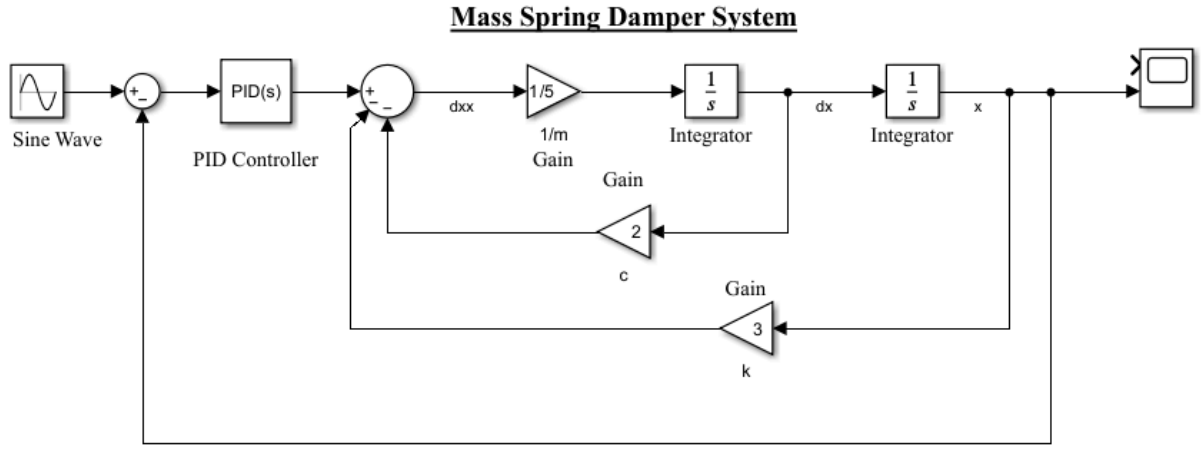
• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

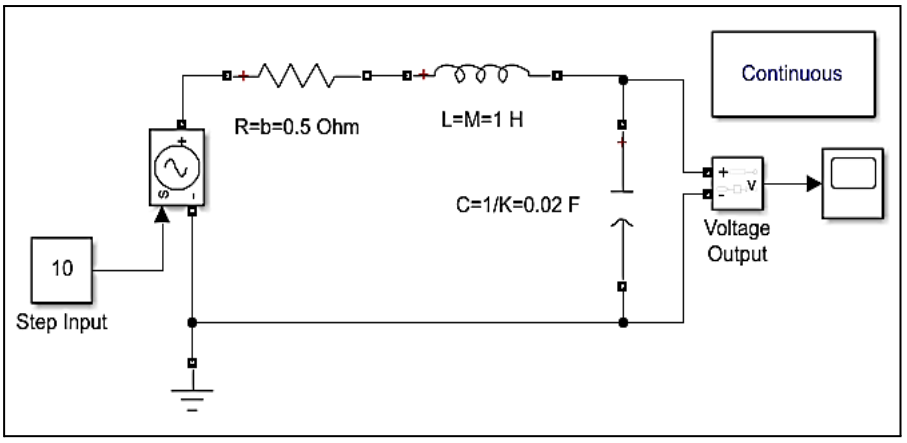
• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit

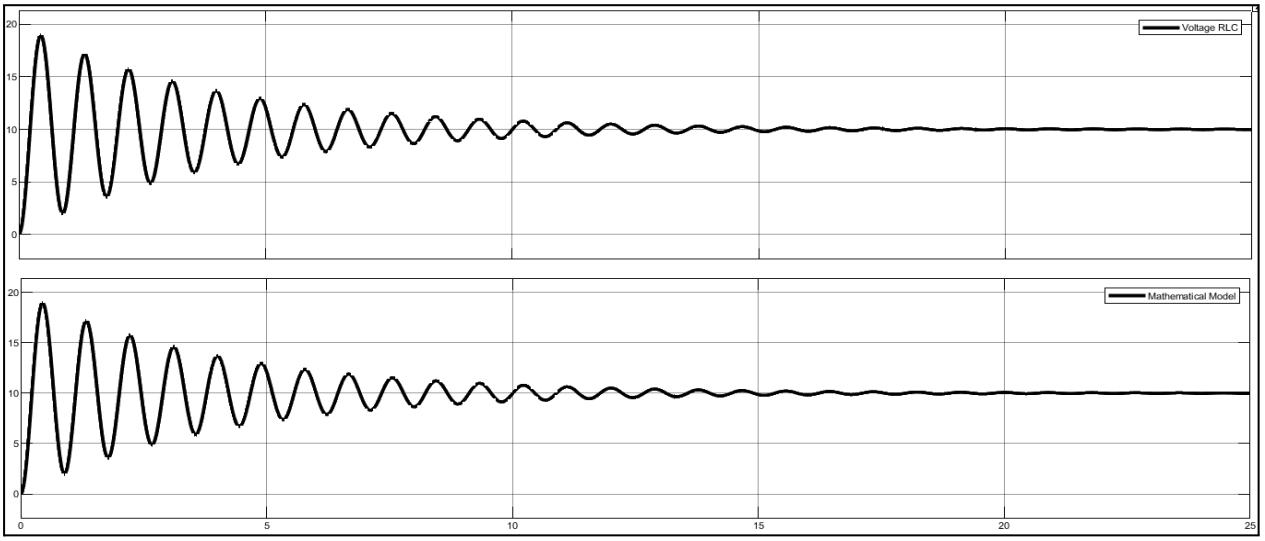


(Fig-4.2-Mass Spring Damper Simulation Circuit)



(Fig-4.3-Mass Spring Damper Analogous Diagram)

Output Graph



(Fig-4.4-Mass Spring Damper Waveform)

Experiment-5

Aim

To simulate the working of the liquid level system using MATLAB.

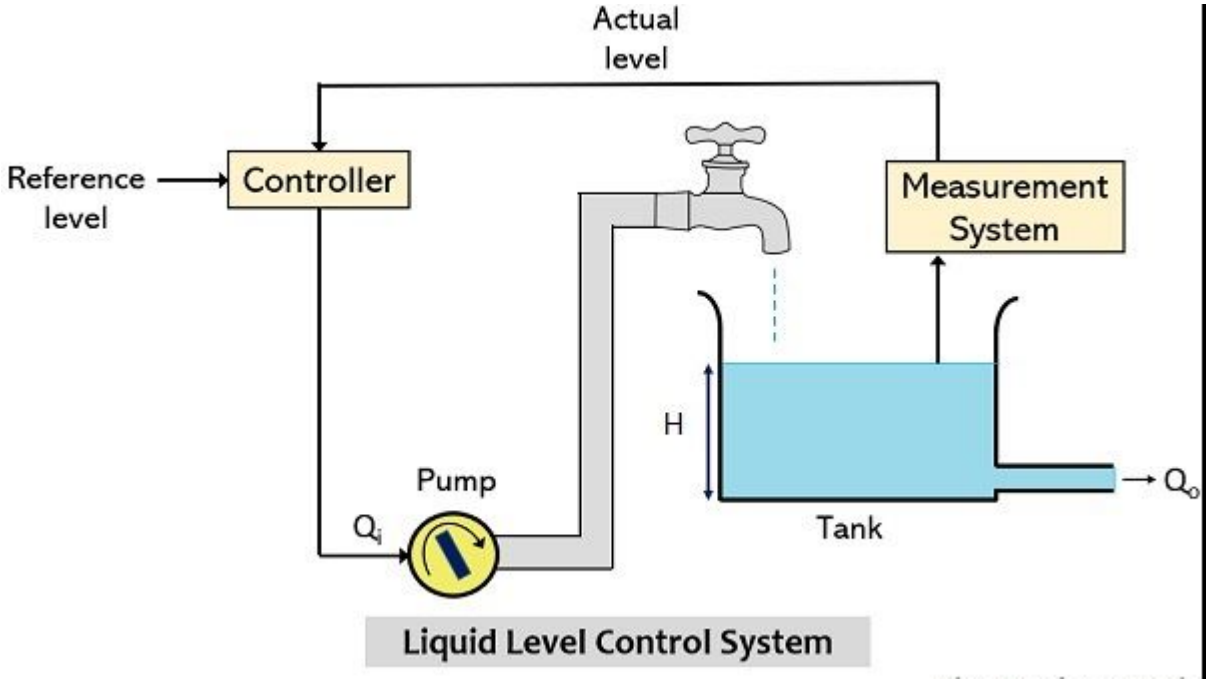
Software/Hardware Used

Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

A liquid level system is a crucial component in industrial processes, responsible for measuring and controlling the level of liquid within a container or tank. These systems are ubiquitous in industries such as chemical processing, water treatment, and food production, where precise control of liquid levels is essential for ensuring product quality, safety, and process efficiency.

At its core, a liquid level system consists of sensors for measuring the level of liquid, actuators for controlling flow rates or valves, and control algorithms to regulate the liquid level within desired parameters. By accurately monitoring and adjusting liquid levels, these systems help maintain optimal operating conditions and prevent overflows or shortages that could lead to costly disruptions or hazards.



(Fig-5.1-Liquid Level Control system Diagram)

MATLAB provides a powerful platform for simulating and analyzing the behavior of liquid level systems. With its Simulink toolbox, engineers can create dynamic models that mimic the real-world behavior of these systems, allowing for virtual experimentation and optimization before implementing changes in actual industrial settings.

In a simulation using MATLAB's Simulink, engineers can develop a detailed model of the liquid-level system, incorporating factors such as tank dimensions, liquid properties, sensor characteristics, and control strategies. By defining the system parameters and control algorithms within the Simulink environment, engineers can simulate various scenarios to analyze system responses under different operating conditions.

Furthermore, MATLAB's extensive computational capabilities enable engineers to perform complex analyses and optimizations to enhance the performance of liquid level systems. This may involve tuning control parameters, optimizing sensor placement, or evaluating the impact of disturbances on system behavior.

Overall, MATLAB's Simulink toolbox provides a comprehensive platform for designing, simulating, and analyzing liquid level systems, empowering engineers to develop robust control strategies and ensure the smooth operation of industrial processes reliant on precise liquid level control.

Procedure:

• Define parameters such as tank dimensions, liquid properties, and sensor characteristics.

• Build a Simulink model including components like sources, sensors, actuators, and controllers.

• Use mathematical models to represent tank dynamics, considering factors like inflow, outflow, disturbances, and control actions.

• Implement control algorithms such as PID, fuzzy logic, or MPC to regulate liquid level.

• Run simulations to observe system behavior, analyze results, and validate control strategies.

• Iterate on the design, adjust parameters, and optimize control algorithms to improve performance.

• Validate the model against real-world data, verify accuracy, and make adjustments as necessary.

• Document the model, methodology, simulation results, and conclusions in a comprehensive report.

Precautions-

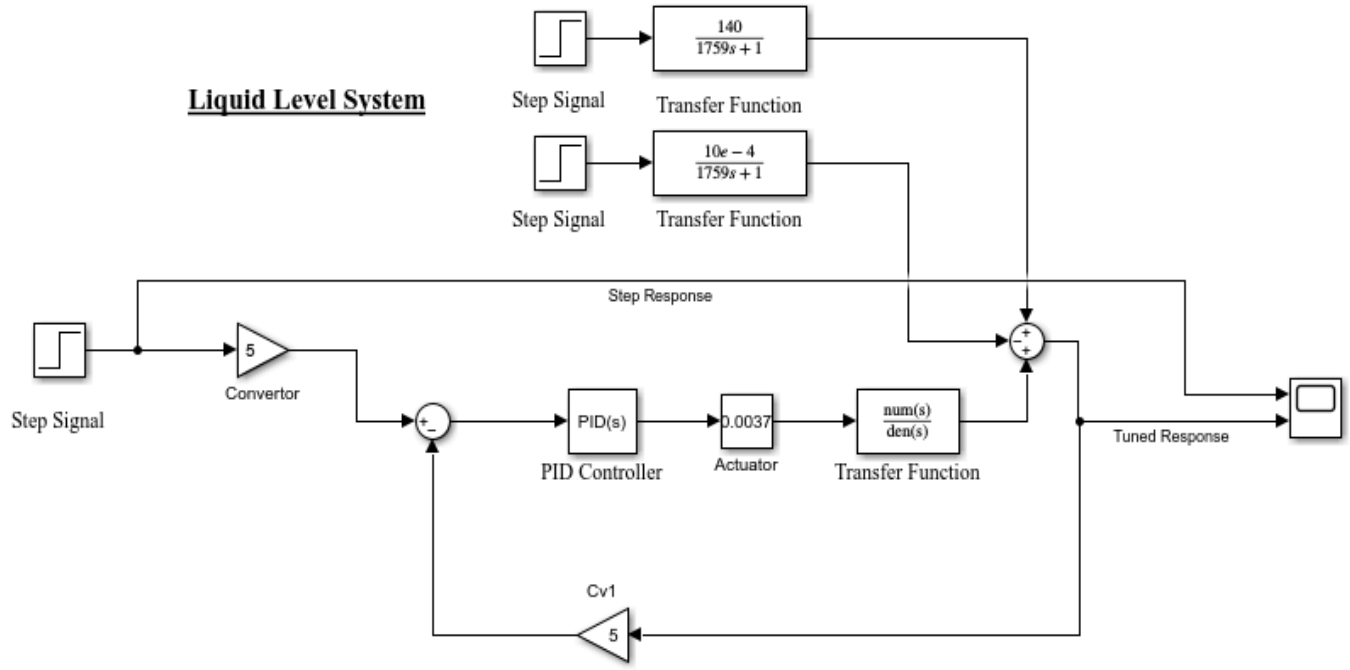
• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

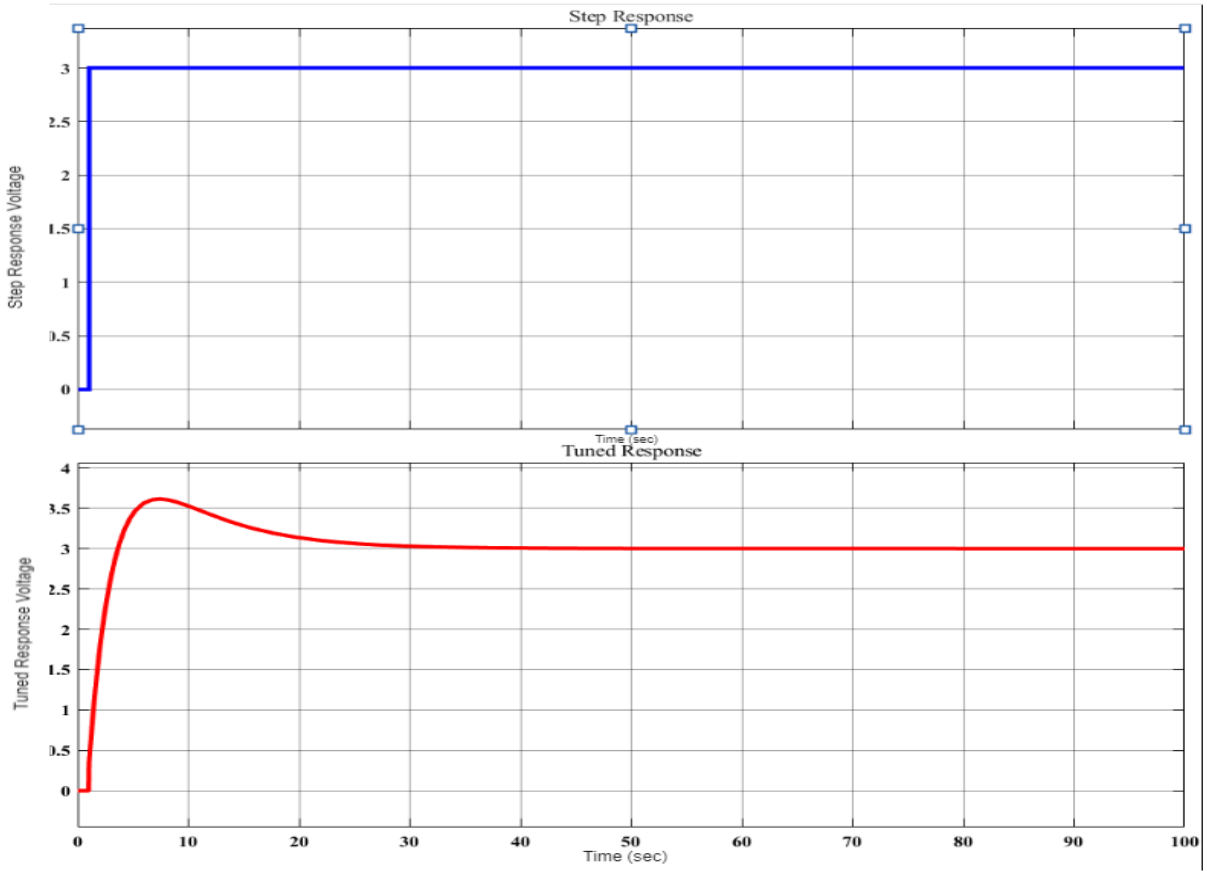
• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit



(Fig-5.2-Liquid Level Control system Circuit)

Output Graph



(Fig-5.3-Liquid Level Control system Waveform)

Experiment-6

Aim

To simulate the working of P, PI, and PID controllers for different linear systems using MATLAB.

Software/Hardware Used

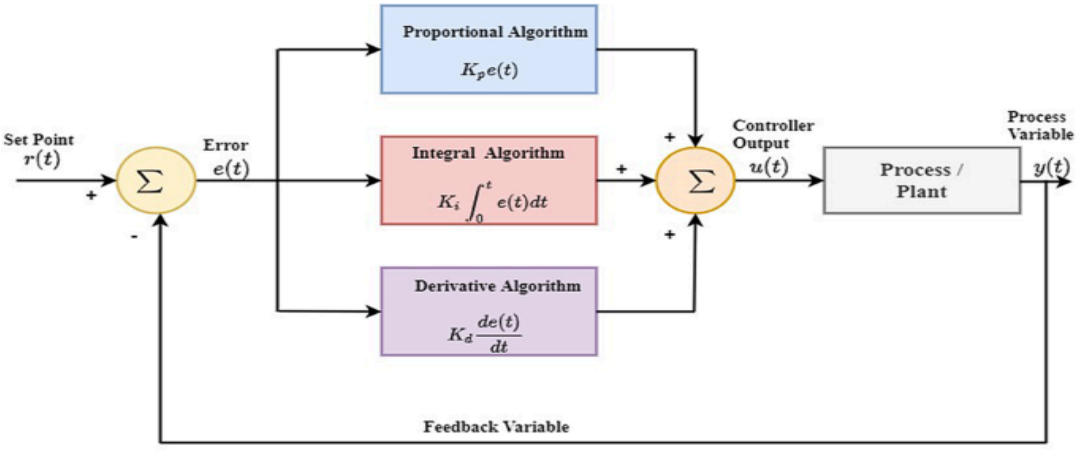
Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

Simulating the working of P (Proportional), PI (Proportional-Integral), and PID (Proportional-Integral-Derivative) controllers for various linear systems using MATLAB offers valuable insights into control system dynamics and performance. These controllers are vital components in engineering and automation, enabling precise regulation of system behavior in response to external inputs or disturbances.

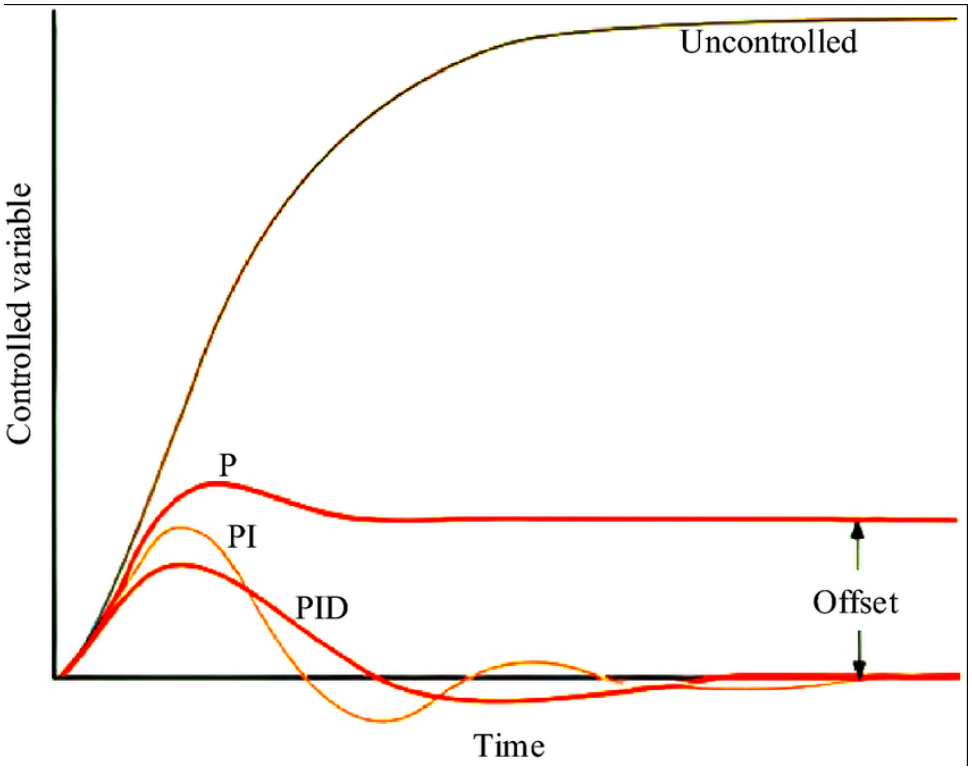
In a P controller, the control effort is directly proportional to the error between the desired setpoint and the actual system output. This proportional action helps reduce steady-state error but may lead to oscillations in the system response. Incorporating integral action in a PI controller allows for the accumulation of error over time, ensuring the elimination of steady-state error. The PI controller provides improved steady-state performance compared to the P controller but may still exhibit sluggish response to sudden changes.

Introducing derivative action in a PID controller adds a predictive element, enabling the controller to anticipate future changes in the system's behavior based on the rate of change of the error. The derivative action helps dampen oscillations and improve transient response, leading to faster settling times and reduced overshoot.



(Fig-6.1-Linear P, PI,PID Diagram)

MATLAB provides a comprehensive platform for simulating and analyzing the performance of P, PI, and PID controllers across different linear systems. Engineers can utilize MATLAB's Control System Toolbox to design and implement these controllers, considering factors such as system dynamics, stability, and desired performance specifications.



(Fig-6.2-Linear P, PI,PID Waveform)

By simulating the behavior of P, PI, and PID controllers in MATLAB, engineers can evaluate their effectiveness in regulating system responses under varying operating conditions. They can analyze key performance metrics such as rise time, settling time, overshoot, and steady-state error to assess controller performance and fine-tune controller parameters for optimal system performance.

Furthermore, MATLAB offers tools for tuning controller gains automatically or through manual methods like Ziegler-Nichols or trial-and-error, facilitating the optimization of controller parameters to meet specific design requirements. Overall, simulating P, PI, and PID controllers using MATLAB empowers engineers to design robust control systems capable of achieving desired performance objectives across a wide range of linear systems.

Procedure:

- Obtain transfer function or state-space models.
- Define gains (Kp, Ki, Kd) for P, PI, and PID controllers.
- Use Control System Toolbox for controller design.
- Simulate system responses under various conditions.
- Fine-tune controller gains for desired performance.
- Compare controller performance metrics.
- Document designs, simulations, and analysis for each controller type.

Precautions-

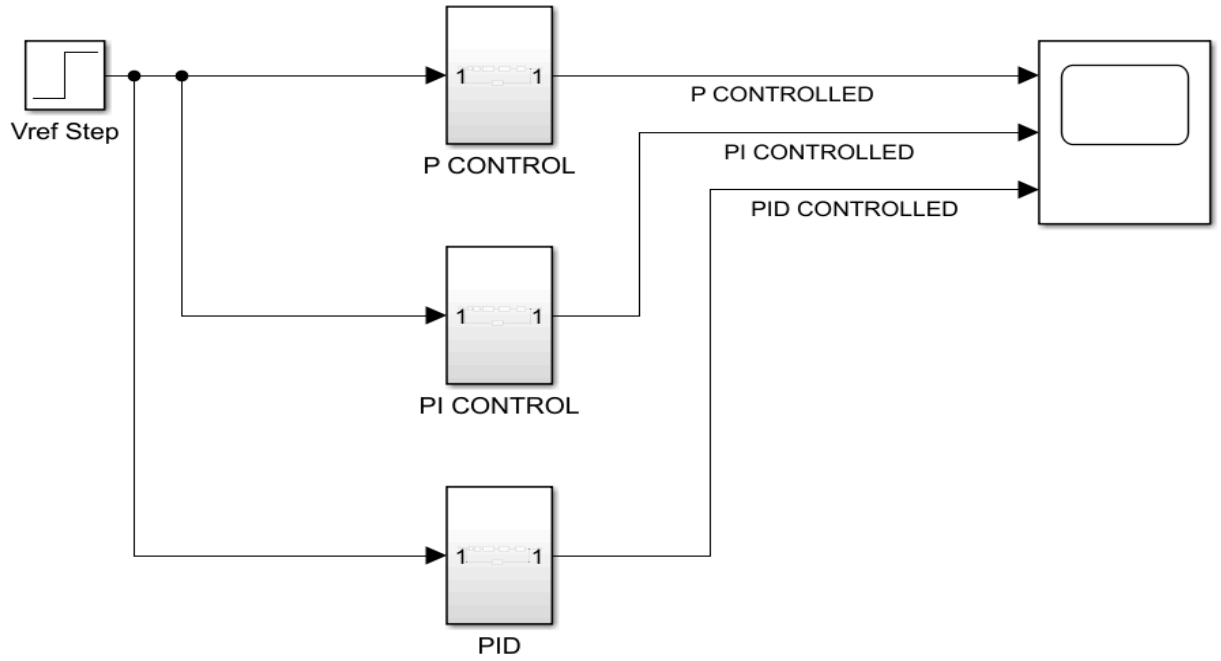
• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

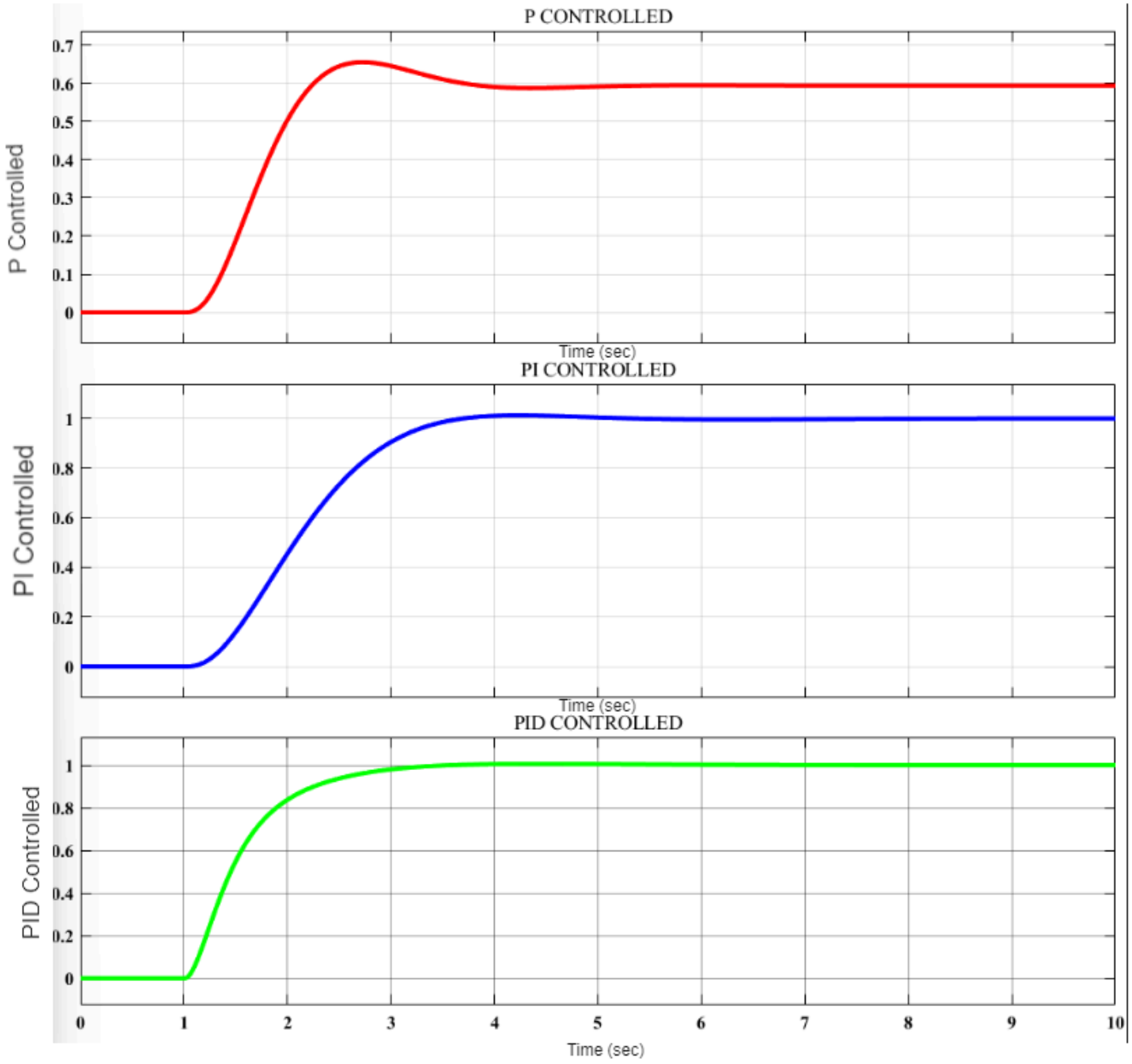
• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit



(Fig-6.3-Linear P, PI,PID Simulation Circuit)

Output Graph



(Fig-6.4-Linear P, PI,PID Simulation Output Waveform)

Experiment-7

Aim

To simulate the working of P, PI, and PID controllers for different Non-linear systems using MATLAB.

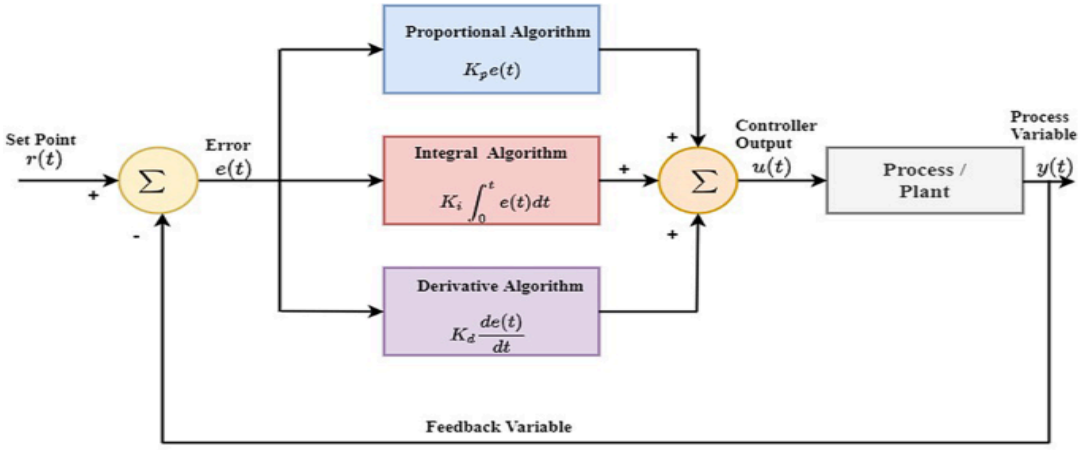
Software/Hardware Used

Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

Simulating the operation of P (Proportional), PI (Proportional-Integral), and PID (Proportional-Integral-Derivative) controllers for various nonlinear systems using MATLAB offers valuable insights into control system behavior and performance under complex conditions. Nonlinear systems exhibit behavior that cannot be described by linear relationships between inputs and outputs, making their analysis and control challenging yet essential in many engineering applications.

In nonlinear systems, the relationship between input and output variables may change based on the operating conditions, system parameters, or external disturbances. As a result, traditional linear control techniques may not suffice for effectively regulating system behavior. However, P, PI, and PID controllers remain versatile tools for addressing nonlinearities and achieving desired control objectives.



(Fig-7.1-Non-Linear P, PI,PID Diagram)

MATLAB provides powerful tools and libraries for simulating and analyzing nonlinear systems and implementing control algorithms. Engineers can utilize MATLAB's Control System Toolbox to design and tune P, PI, and PID controllers for nonlinear systems, considering factors such as system dynamics, stability, and desired performance specifications.

Simulating the operation of P, PI, and PID controllers in MATLAB for nonlinear systems involves modeling the nonlinear behavior of the system and integrating the controller into the simulation environment. Engineers can employ various techniques such as state-space

models, transfer functions, or numerical methods to represent the nonlinear dynamics accurately.

By simulating the behavior of P, PI, and PID controllers for nonlinear systems in MATLAB, engineers can evaluate their effectiveness in regulating system responses under different operating conditions and nonlinearities. They can analyze key performance metrics such as transient response, stability, robustness, and tracking accuracy to assess controller performance and optimize controller parameters accordingly.

Furthermore, MATLAB offers advanced control design tools, such as nonlinear model predictive control (NMPC) or adaptive control, which can be utilized to address specific challenges posed by nonlinear systems. These tools enable engineers to develop robust control strategies capable of handling nonlinearities and uncertainties effectively.

Overall, simulating P, PI, and PID controllers for nonlinear systems using MATLAB empowers engineers to design and implement effective control solutions for a wide range of complex engineering applications, ensuring stable and reliable system performance in the face of nonlinear dynamics.

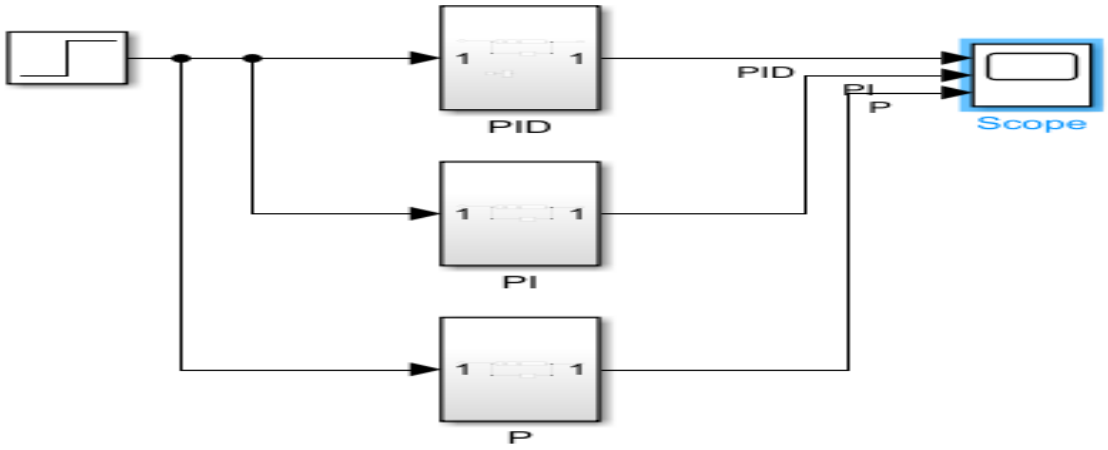
Procedure:

- Obtain transfer function or state-space models.
- Define gains (Kp, Ki, Kd) for P, PI, and PID controllers.
- Use Control System Toolbox for controller design.
- Simulate system responses under various conditions.
- Fine-tune controller gains for desired performance.
- Compare controller performance metrics.
- Document designs, simulations, and analysis for each controller type.

Precautions-

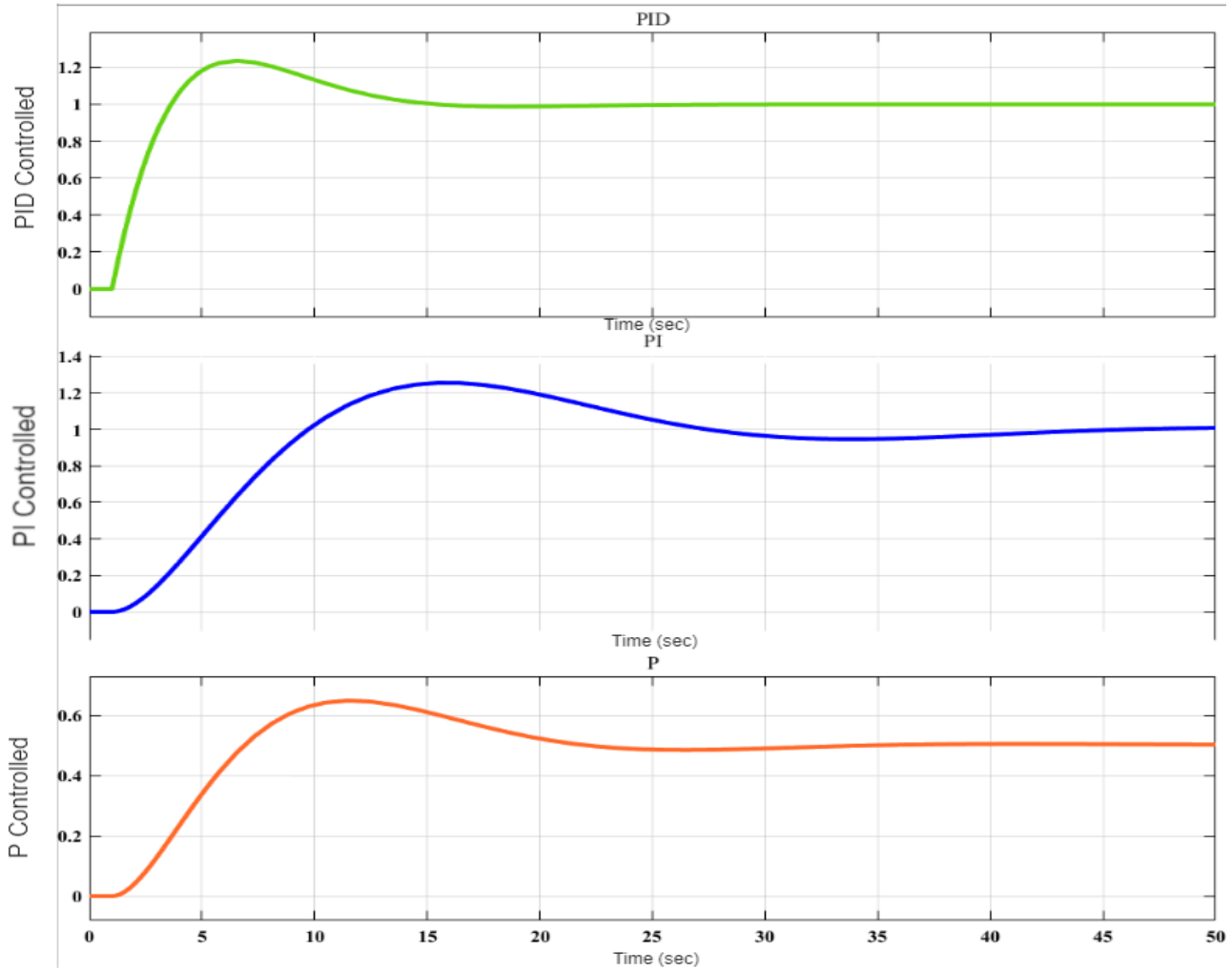
- Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.
- Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.
- Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.
- Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit



(Fig-7.2-Non-Linear P, PI,PID Simulation Circuit)

Output Graph



(Fig-7.3-Non-Linear P, PI,PID Waveform)

Experiment-8

Aim

To observe the VI characteristics of solar PV systems at different radiation and temperature using MATLAB.

Software/Hardware Used

Dell Latitude 5420, MATLAB (version R2023b), Simulink Toolbox (version 11.1)

Theory

MATLAB facilitates the comprehensive analysis of solar photovoltaic (PV) system Voltage-Current (VI) characteristics, offering valuable insights into the performance of these systems under varying environmental conditions. Solar PV systems convert sunlight into electricity through the photovoltaic effect, where solar cells generate electric current when exposed to light. Understanding how changes in radiation and temperature impact the behavior of solar cells is essential for optimizing the efficiency and reliability of solar energy utilization.

With MATLAB, engineers and researchers can create sophisticated models to simulate the behavior of solar cells under different environmental conditions. These models incorporate physical equations and empirical data to accurately represent the electrical characteristics of solar cells, including their VI curves. By inputting varying levels of solar radiation and temperature, MATLAB enables users to observe how these factors affect the output voltage and current of the solar cells.

Analyzing the VI characteristics of solar PV systems allows stakeholders to assess system performance, identify potential issues, and optimize system design and operation. For example, changes in radiation levels throughout the day or variations in temperature can significantly impact the output power of solar panels. MATLAB simulations can help predict these effects and guide decisions related to system sizing, orientation, shading mitigation strategies, and the selection of appropriate components.

Moreover, MATLAB's extensive computational capabilities enable engineers to conduct parametric studies and sensitivity analyses, exploring the influence of different environmental variables on system performance. By systematically varying parameters such as solar irradiance, temperature, and cell configuration, MATLAB users can gain valuable insights into the robustness and reliability of solar PV systems under diverse operating conditions.

Overall, MATLAB serves as a powerful tool for analyzing Solar PV system VI characteristics, providing a versatile platform for studying the impact of environmental factors on system performance. Through accurate simulation and analysis, MATLAB helps optimize solar energy utilization, contributing to the advancement of sustainable energy technologies and the transition towards cleaner and more efficient power generation systems.

Procedure:

• Develop a MATLAB Simulink model representing the solar PV system, including solar panels, sensors for irradiance and temperature, and a load.

• Define parameters for radiation and temperature, considering different levels to observe their effects on the VI characteristics.

• Configure MATLAB environment with necessary toolboxes such as SimPowerSystems for simulating power electronic systems.

• Run simulations in MATLAB to observe the VI characteristics of the solar PV system under varying radiation and temperature conditions.

• Analyze simulation results to observe changes in the voltage-current (VI) characteristics corresponding to different levels of radiation and temperature.

• Generate plots or graphs to visualize the VI characteristics at different radiation and temperature levels, facilitating comparative analysis.

• Evaluate system performance based on observed VI characteristics, and optimize parameters or control strategies to improve solar PV system efficiency and reliability.

Precautions-

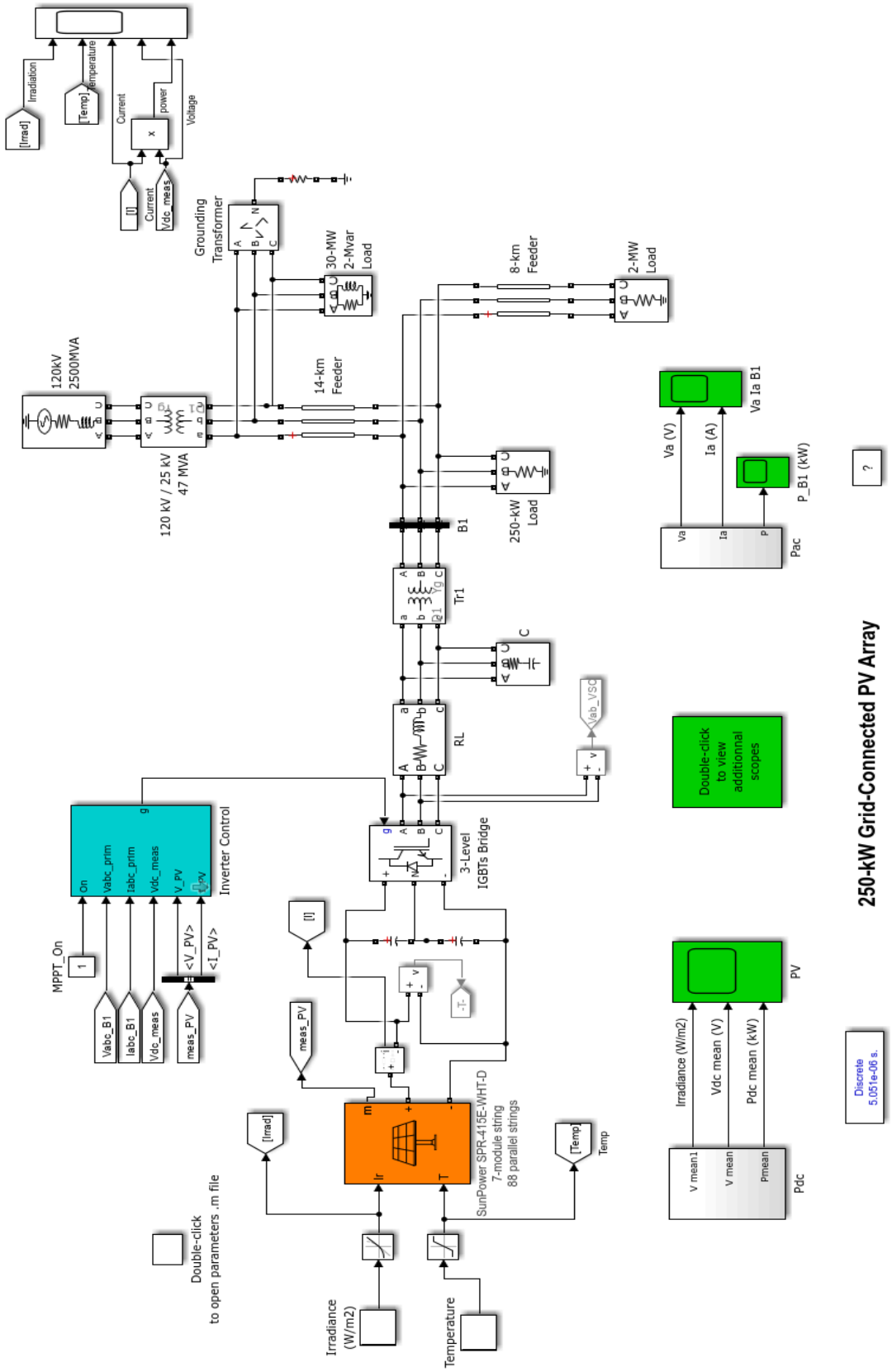
• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

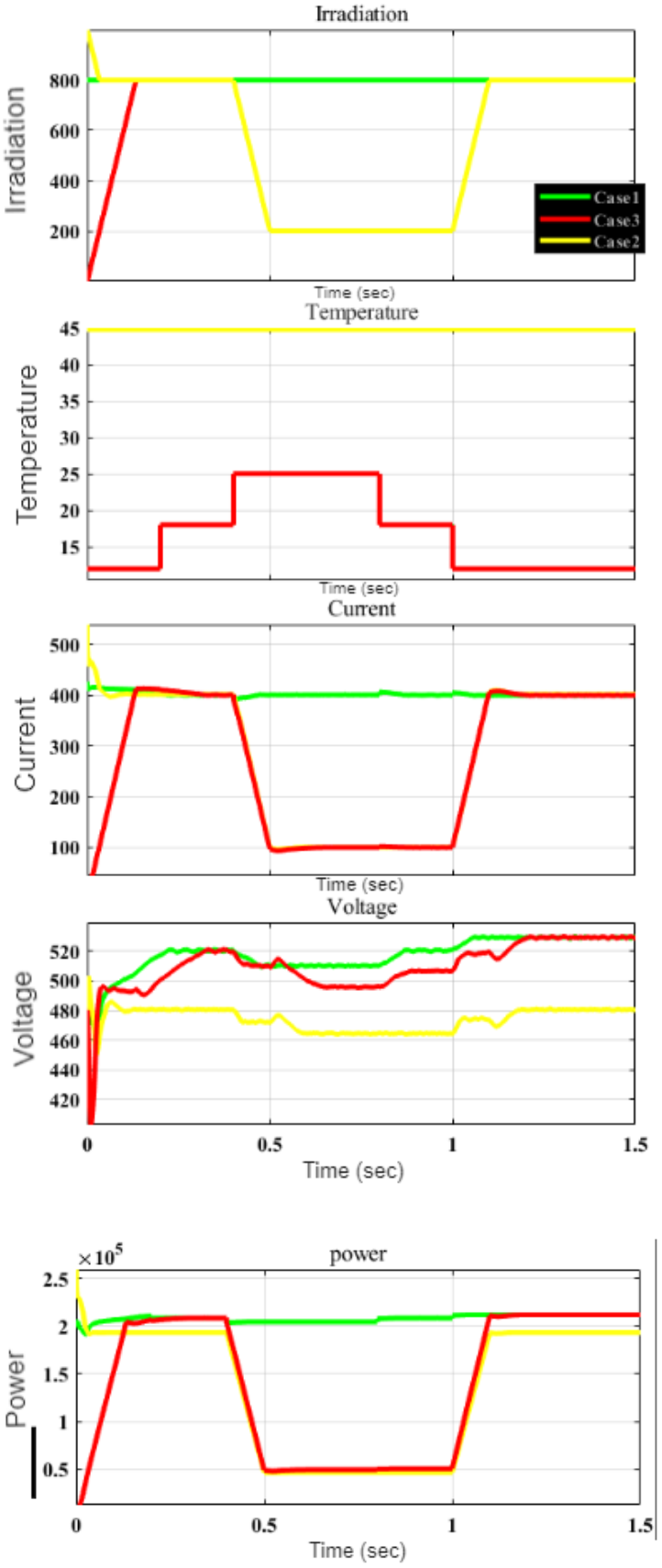
• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit



(Fig-8.1-PV System Simulation Circuit)

Output Graph



(Fig-8.2-PV System Waveform)

LT-SPICE Experiments

Experiment-1

Aim

To study the half and full wave rectifier in LTSPICE.

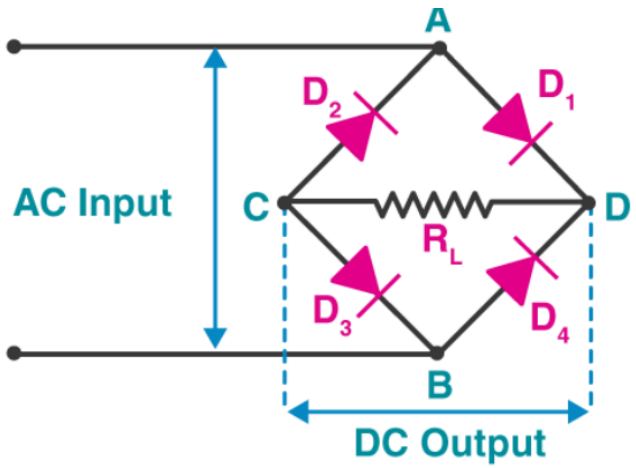
Software/Hardware Used

Dell Latitude 5420, LTSPICE(Version 17.1.8)

Theory

1) Full wave rectifier:

A full wave rectifier is defined as a rectifier that converts the complete cycle of alternating current into pulsating DC. Unlike half wave rectifiers that utilize only the half wave of the input AC cycle, full wave rectifiers utilize the full cycle. The lower efficiency of the half wave rectifier can be overcome by the full wave rectifier. The circuit of the full wave rectifier can be constructed in two ways. The first method uses a centre tapped transformer and two diodes. This arrangement is known as a centre tapped full wave rectifier. The second method uses a standard transformer with four diodes arranged as a bridge. This is known as a bridge rectifier. The circuit of the full wave rectifier consists of a step-down transformer and two diodes that are connected and centre tapped. The output voltage is obtained across the connected load resistor. The construction of a bridge rectifier is shown in the figure below. The bridge rectifier circuit is made of four diodes D1, D2, D3, D4, and a load resistor RL. The four diodes are connected in a closed-loop configuration to efficiently convert the alternating current (AC) into Direct Current (DC). The main advantage of this configuration is the absence of the expensive centre-tapped transformer. Therefore, the size and cost are reduced.



(Fig-1.1-Full Wave Rectifier Diagram)

The input signal is applied across terminals A and B, and the output DC signal is obtained across the load resistor RL connected between terminals C and D. The four diodes are arranged in such a way that only two diodes conduct electricity during each half cycle. D1 and D3 are

pairs that conduct electric current during the positive half cycle. Likewise, diodes D2 and D4 conduct electric current during a negative half cycle.

2) Half Wave rectifier

Half-wave rectifiers transform AC voltage to DC voltage. A half wave rectifier circuit uses only one diode for the transformation. A half wave rectifier is defined as a type of rectifier that allows only one-half cycle of an AC voltage waveform to pass while blocking the other half cycle. A half-wave rectifier is the simplest form of the rectifier and requires only one diode for the construction of a half wave rectifier circuit. A half wave rectifier circuit consists of three main components as follows:

● A diode

● A transformer

● A resistive load

Procedure

- Open LTspice and create a new schematic file.
- Connect the components as shown in the diagram, for both full-wave and half wave rectifiers.
- Click on the "Run" button in the toolbar, or select "Simulate" from the menu bar
- Put the pointer at the desired location, to get the desired output.

Precautions-

• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

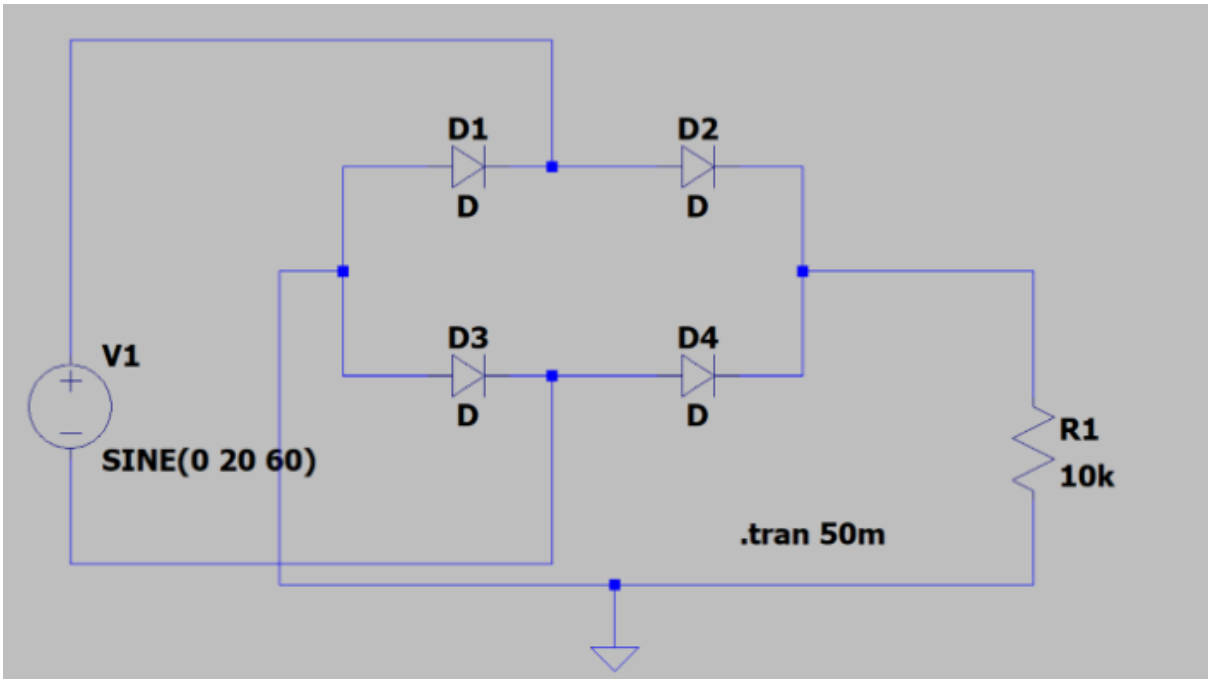
• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

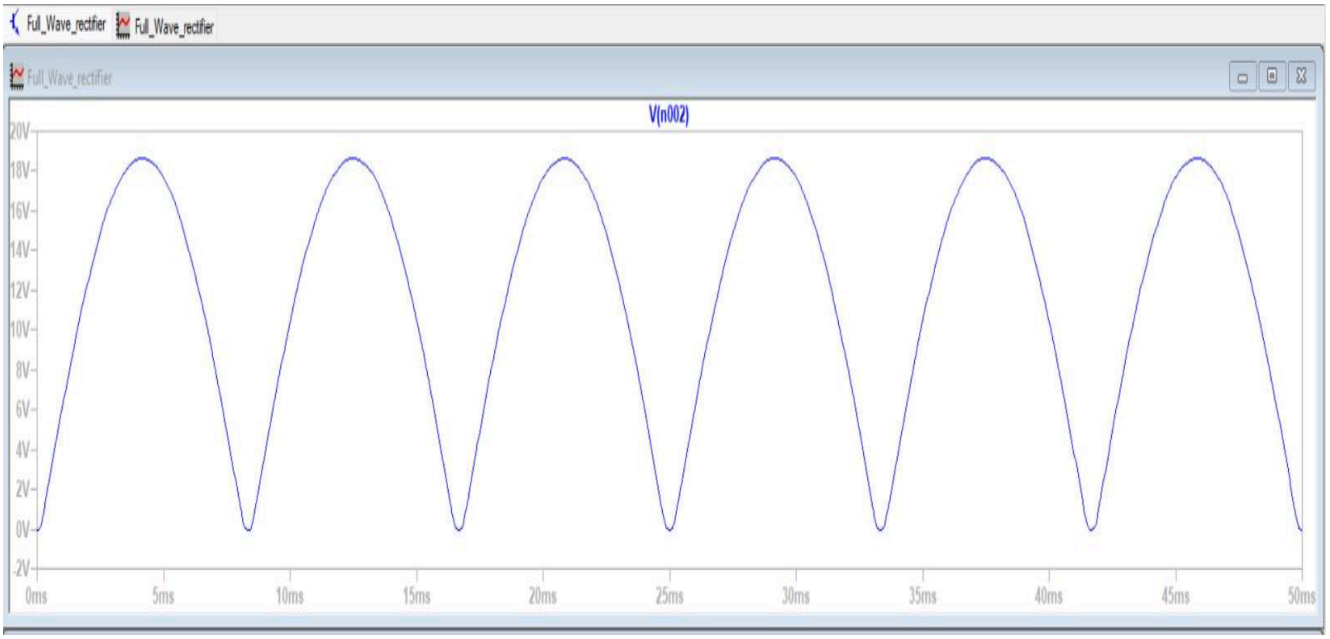
Simulation Circuit

1) Full wave rectifier:



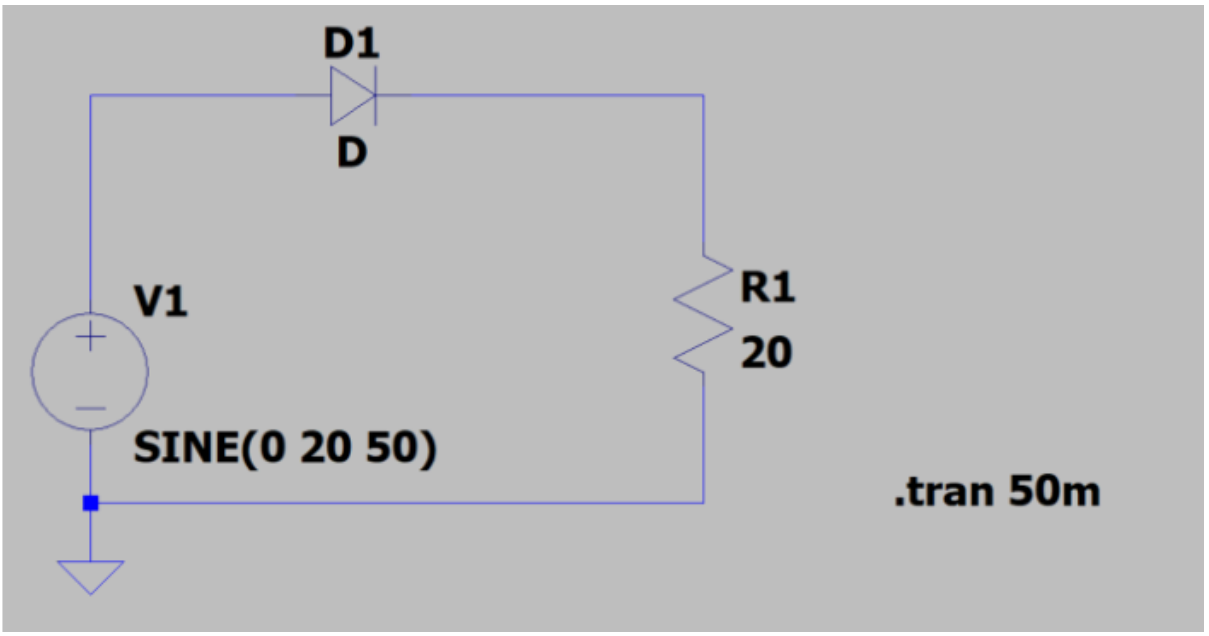
(Fig-1.2-Full Wave Rectifier Simulation Circuit)

Output Graph



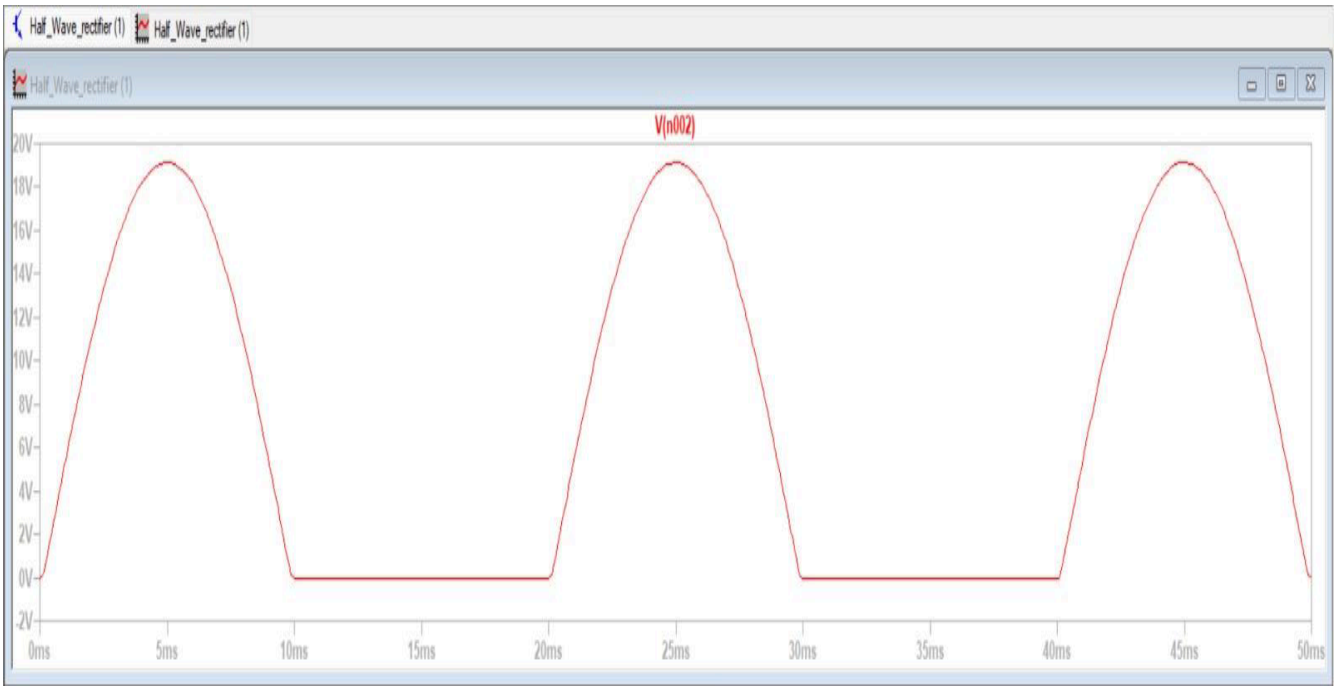
(Fig-1.3-Full Wave Rectifier Waveform)

2) Half wave rectifier:



(Fig-1.2-Half Wave Rectifier Simulation Circuit)

Output Graph



(Fig-1.3-Half Wave Rectifier Waveform)

Experiment-2

Aim

To study the working of inverting and non-inverting Op-amp.

Software/Hardware Used

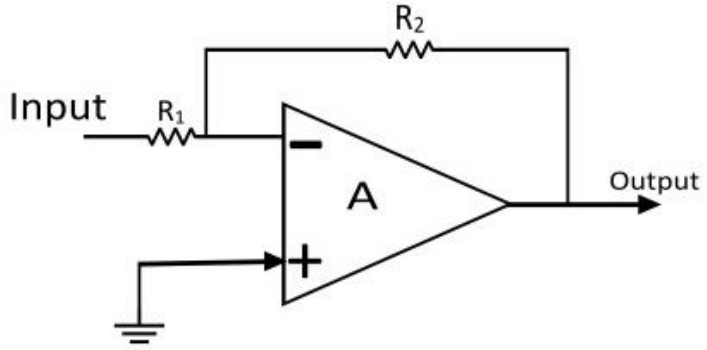
Dell Latitude 5420, LTSPICE(Version 17.1.8)

Theory:

An operational amplifier is a three-terminal device consisting of two high impedance input terminals, one is called the inverting input denoted by a negative sign and the other is the non-inverting input denoted with a positive sign. The third terminal is the output of the Op-Amp.

1) Inverting Operational Amplifier:

In the inverting operational amplifier circuit, the signal is applied at the inverting input and the non-inverting input is connected to the ground. In this type of amplifier, the output is 180^0 out of phase to the input, i.e. when positive signal is applied to circuit, the output of the circuit will be negative. By assuming the Op-Amp is ideal, then the concept of virtual short can be applied at the input terminals of the Op-Amp. So that voltage at the inverting terminal is equal to the voltage at non-inverting terminal.



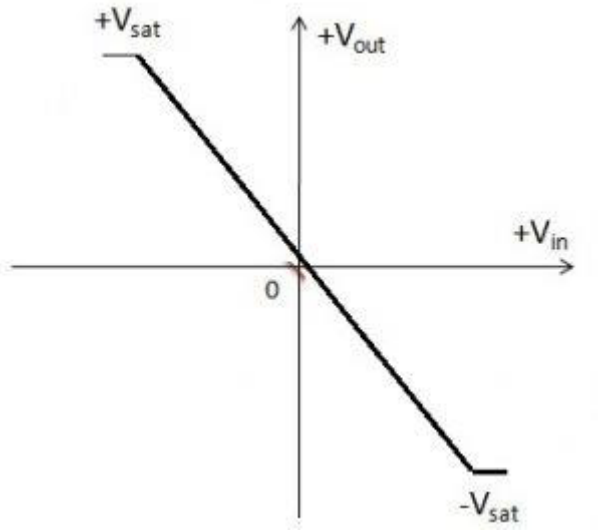
(Fig-2.1-Inverting Operational Amplifier Diagram)

Applying KCL at inverting node of Op-Amp:

$$\frac{0 - V_{in}}{R_1} + \frac{0 - V_{out}}{R_2} = 0$$

$$\text{Voltage Gain}(A_v) = \frac{V_{out}}{V_{in}} = -\frac{R_2}{R_1}$$

The voltage characteristics of inverting amplifier are shown in the below graph. It can be noted that once the input signal is positive like Vin, then the output voltage like Vout is negative. In addition, the output voltage will be changed linearly once the input voltage is applied.

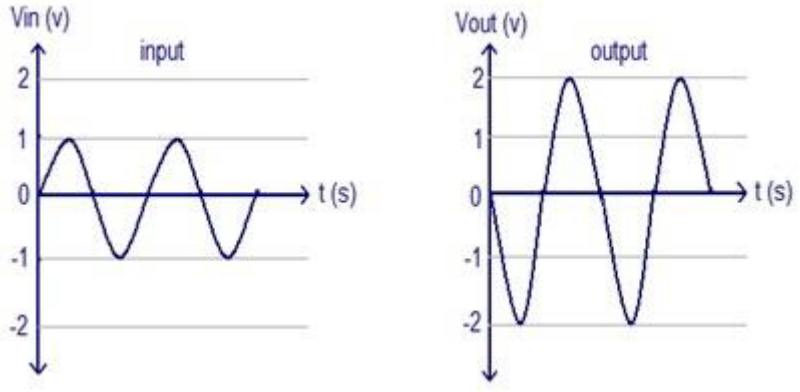


(Fig-2.2-Inverting Operational Amplifier Voltage Characteristics)

This characteristic will saturate otherwise the output will become constant, once the amplitude of the input signal goes ahead of both the applied power supplies to the op-amp.

+VCC = + VSAT & –VCC = -VSAT

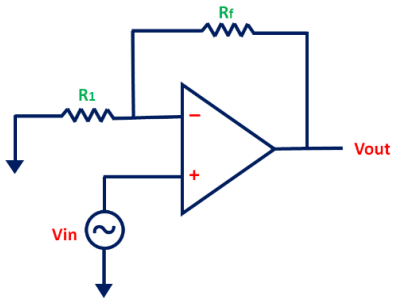
The inverting op-amps input & output waveforms are shown below. The following waveforms can be drawn by assuming the gain of the amplifier & the sine wave is an input signal. From the following waveforms, it is very clear that the output is double in magnitude as compared to the input like Vout = Av * Vin & phase is reverse to the input.



(Fig-2.3-Inverting Operational Amplifier Input and Output Waveforms)

2) Non-Inverting Operational Amplifier:

When the signal is applied at the non-inverting input, the resulting circuit is known as Non-Inverting Op-Amp. In this amplifier the output is exactly in phase with the input i.e. when a positive voltage is applied to the circuit, the output will also be positive. By assuming the Op-Amp is ideal, then concept of virtual short can be applied i.e. the voltage at the inverting and non-inverting terminal is equal.



(Fig-2.4-Non-Inverting Operational Amplifier Diagram)

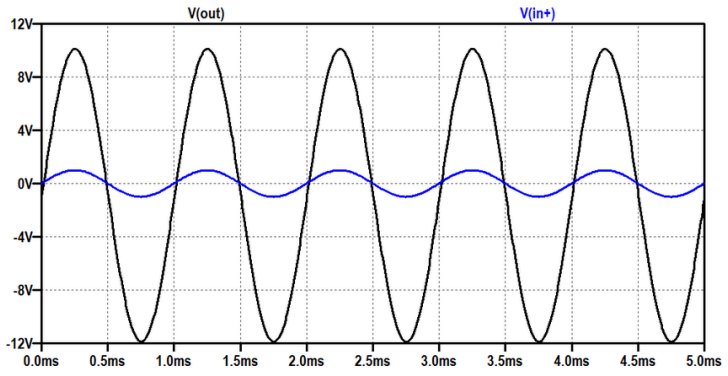
Applying KCL at the inverting node of op-amp:

Non Inverting Op-Amp Waveforms:

$$\frac{V_{in} - V_{out}}{R_2} + \frac{V_0 - 0}{R_1} = 0$$

$$\text{Voltage Gain}(A_v) = \frac{V_{out}}{V_{in}} = 1 + \frac{R_2}{R_1}$$

The input & output waveforms of non-inverting op-amp waveforms are shown below. The signal which needs to change is given to the +ve terminal of the op-amp whereas the –ve terminal is connected to GND with the help of an R1 resistor. The input (Vin) & output (Vout) voltages are within phase through each other, so their phase difference is 0 degrees or 360 degrees.



(Fig-2.5-Non-Inverting Operational Amplifier Input and Output Waveforms)

Procedure:

• Open LTspice and create a new schematic file.

• Connect the components as shown in the diagram, for both Inverting and Non Inverting Amplifier.

• Click on the "Run" button in the toolbar, or select "Simulate" from the menu bar

• Put the pointer at the desired location, to get the desired output.

Precautions-

• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

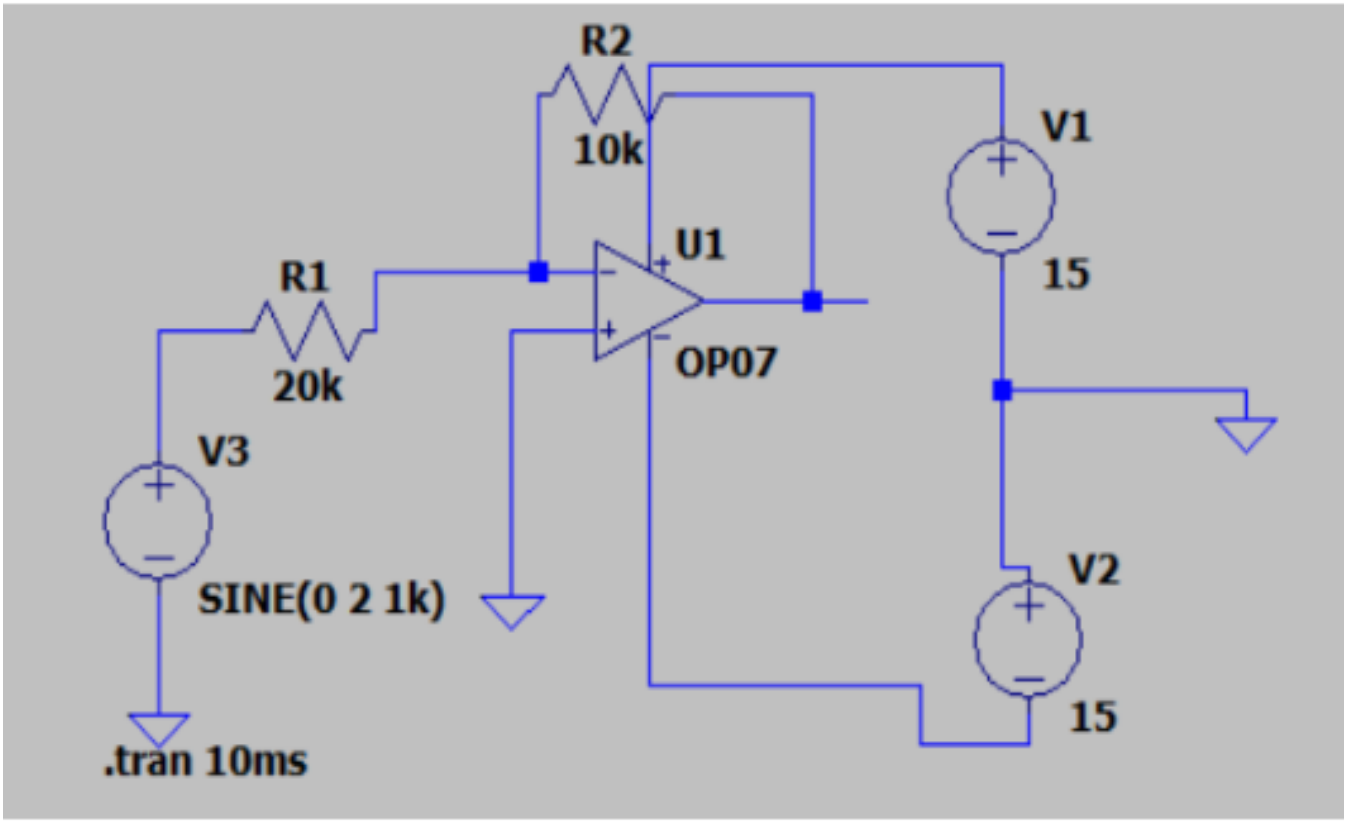
• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

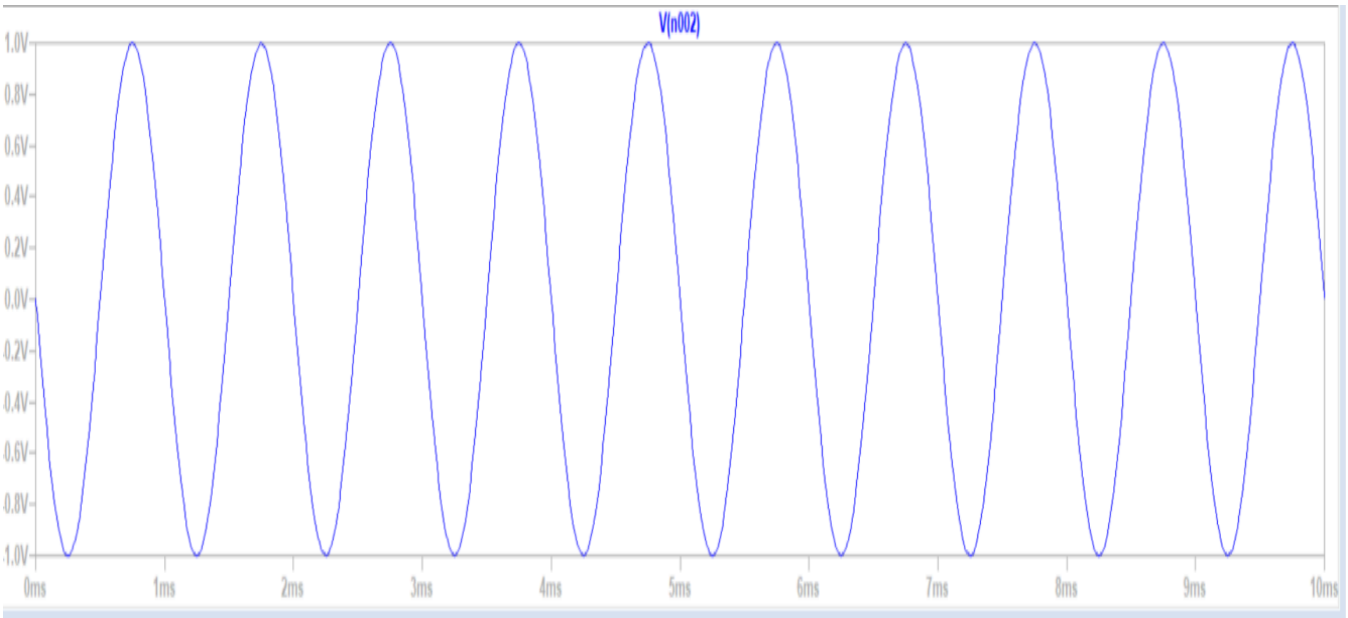
Simulation Circuit

(A) Inverting Op-Amp



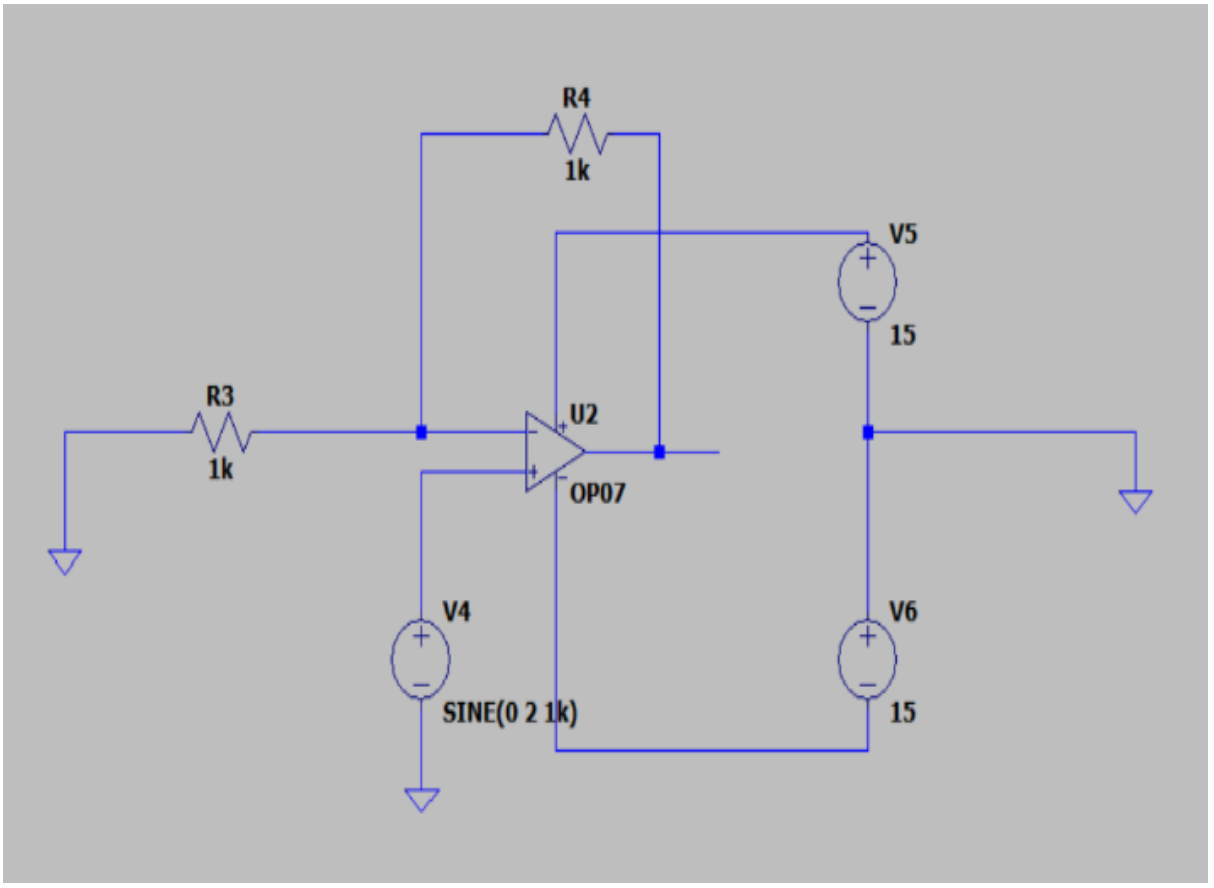
(Fig-2.6-Inverting Operational Amplifier Simulation Circuit)

Output Graph



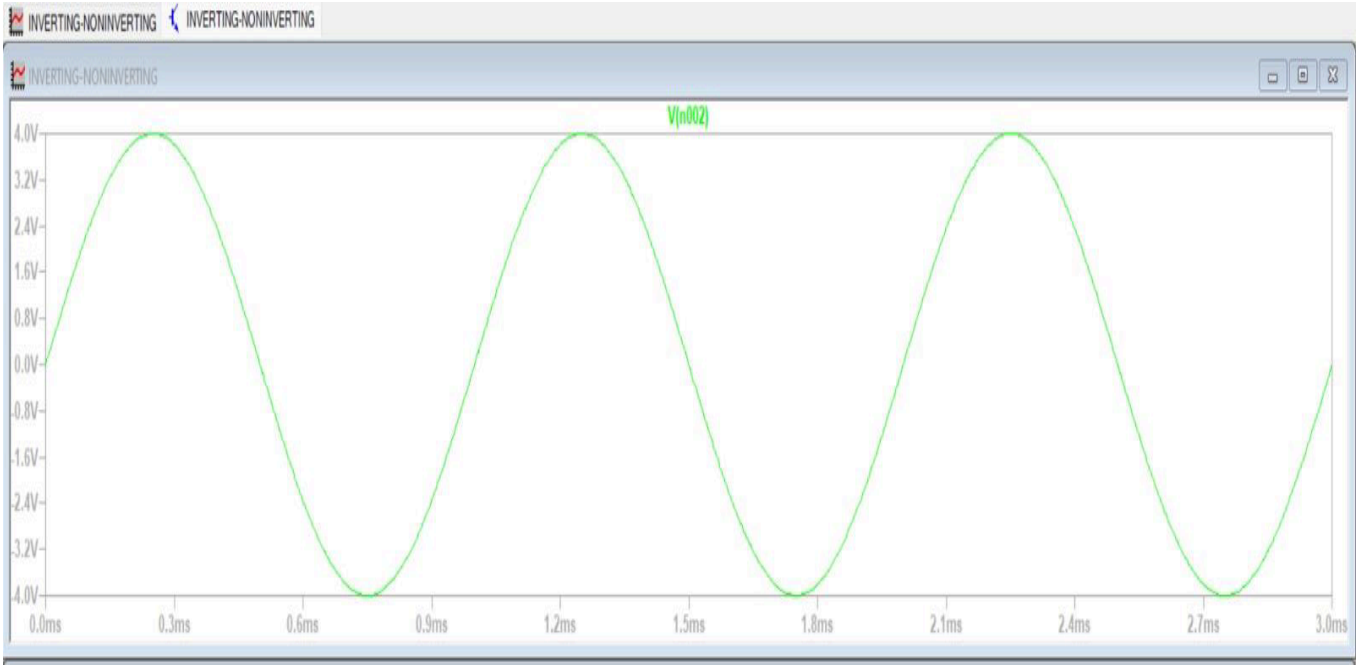
(Fig-2.7-Inverting Operational Amplifier Input and Output Waveforms)

(B) Non-Inverting Op-Amp



(Fig-2.8-Non-Inverting Operational Amplifier Simulation Circuit)

Output Graph



(Fig-2.9-Non-Inverting Operational Amplifier Input and Output Waveforms)

Experiment-3

Aim

To study the working of linear transformer circuits.

Software/Hardware Used

Dell Latitude 5420, LTSPICE(Version 17.1.8)

Theory:

Studying the working of a linear transformer circuit using LTspice involves analyzing the behavior of transformers in electronic circuits. Transformers are crucial components that transfer electrical energy between two or more circuits through electromagnetic induction. They are commonly used in power supplies, amplifiers, and various electronic devices to step up or step down voltage levels while maintaining the frequency of the alternating current (AC) signal.

LTspice is a powerful simulation software widely used for electronic circuit design and analysis. It allows engineers and researchers to model and simulate the behavior of complex electrical circuits accurately. By utilizing LTspice, users can study the performance of transformer circuits under different operating conditions without the need for physical prototyping.

To study the working of a linear transformer circuit using LTspice, one would typically begin by constructing a circuit schematic that includes the transformer, along with other components such as voltage sources, resistors, capacitors, and loads. The circuit schematic represents the physical connections and interactions between the components in the circuit.

Next, users would define the properties of the transformer in the LTspice simulation, including the turns ratio, core material, winding resistance, leakage inductance, and magnetizing inductance. These parameters determine how the transformer behaves in the circuit and influence its performance characteristics.

Once the circuit schematic and transformer properties are defined, users can run simulations in LTspice to analyze the behavior of the transformer circuit. This may involve applying different input voltage waveforms, varying load conditions, or testing the circuit's response to transient events.

During simulation, LTspice generates waveforms and calculates key performance metrics such as voltage, current, power, efficiency, and frequency response. Users can visualize these results through plots and graphs to gain insights into the transformer's behavior and performance under different scenarios.

Procedure:

• Open LTspice and create a new schematic file.

• Connect the components as shown in the diagram, for Linear Transformer Cicuit

• Click on the "Run" button in the toolbar, or select "Simulate" from the menu bar

• Put the pointer at the desired location, to get the desired output.

Precautions-

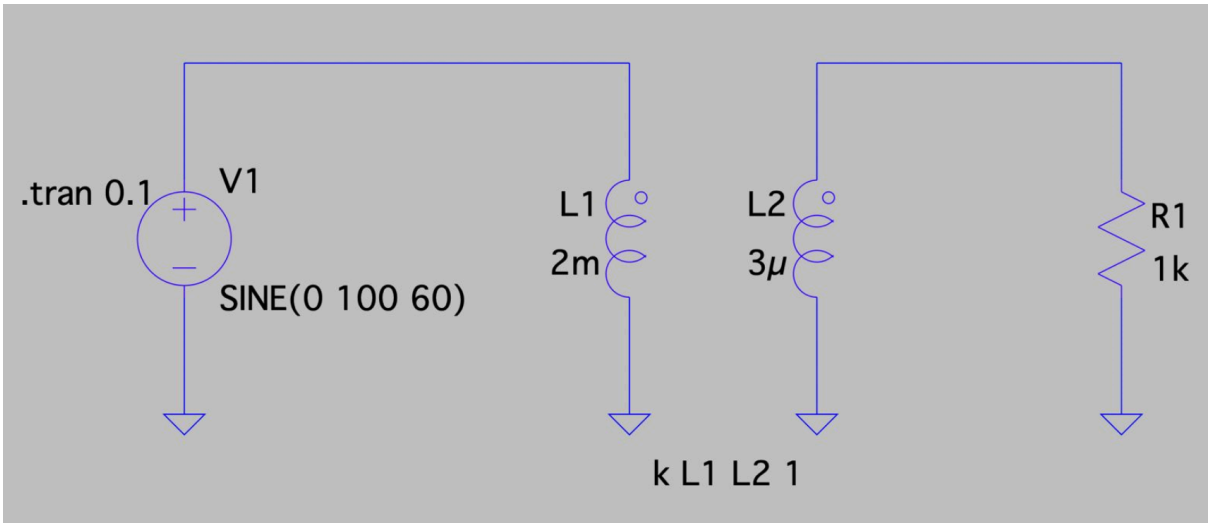
• Ensure that the circuit is properly modelled and simulated according to the specifications and operating conditions.

• Choose appropriate component values for the circuit to avoid damage or overloading to the circuit or the simulation program.

• Double-check the connections and node names in the schematic to avoid errors or incomplete circuits.

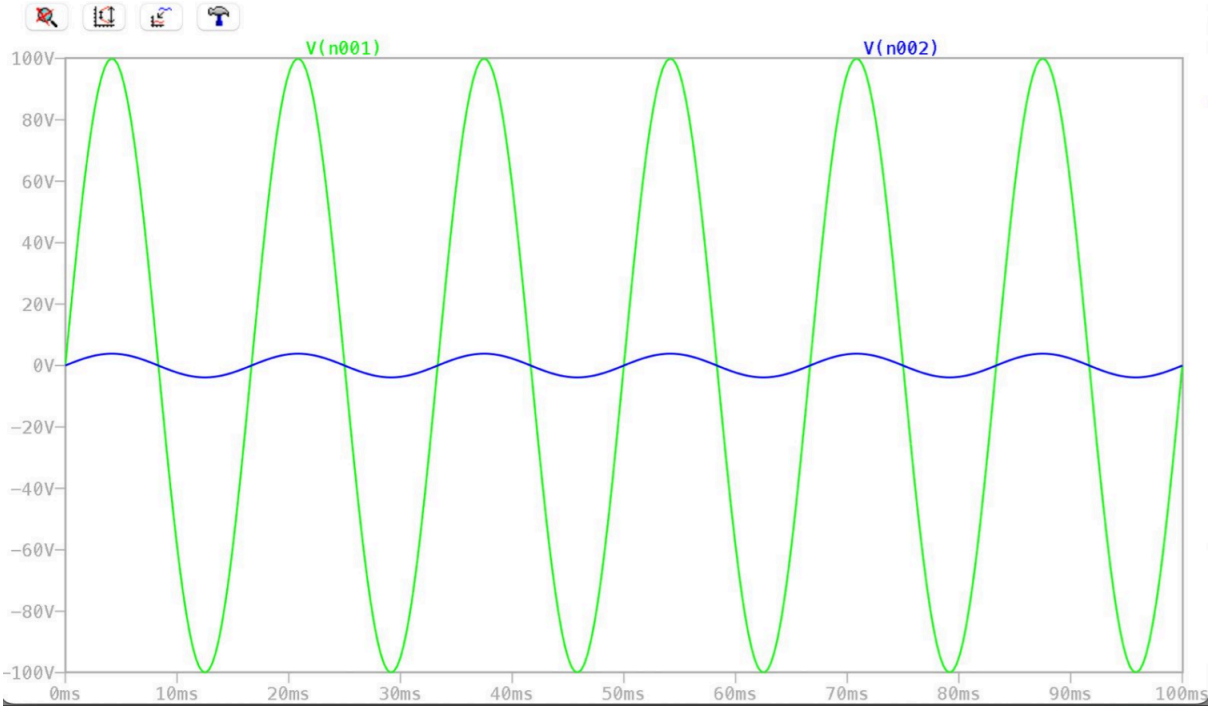
• Use appropriate signal sources and apply safe voltage and current levels to the circuit to avoid damaging the components or simulation program.

Simulation Circuit:



(Fig-3.1-Linear Transformer Simulation Circuit)

Output Graph:



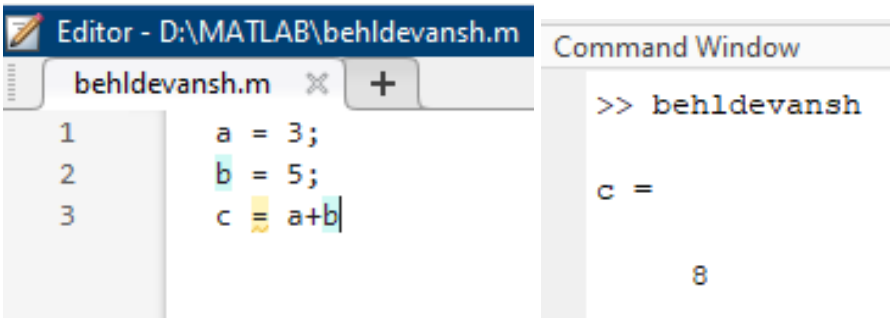
(Fig-3.2-Linear Transformer Simulation Circuit Waveform)

MATLAB Assignment-1

Part 1. Write your first Matlab program

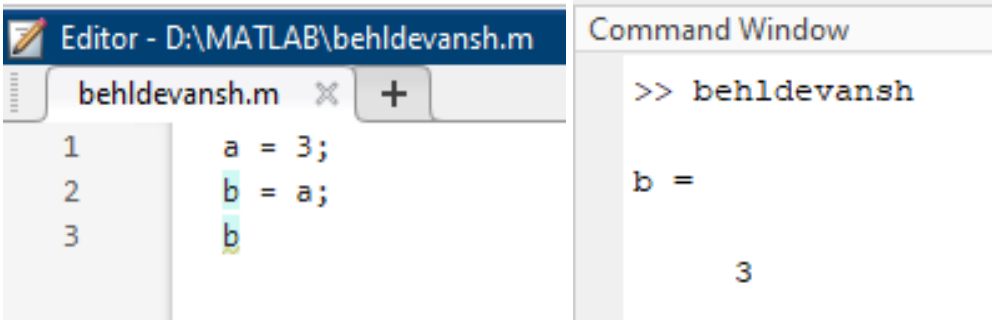
Q-1 Write your first Matlab program

Ans-



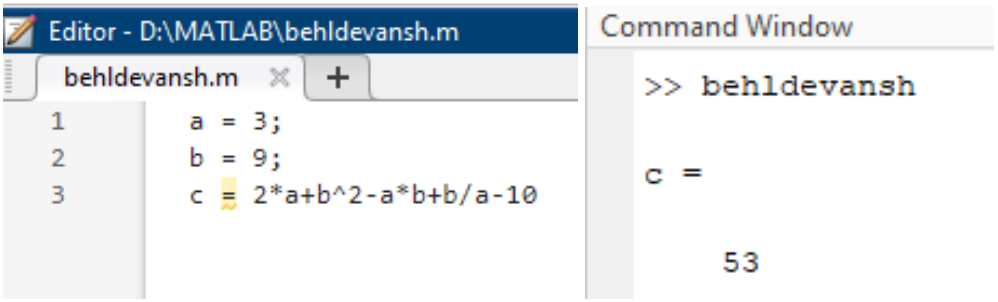
Q-2 The meaning of "a = b"

Ans-



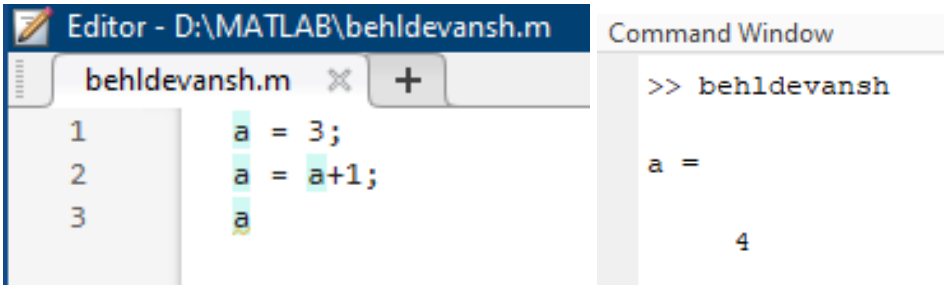
Q-3 Basic math operations

Ans-



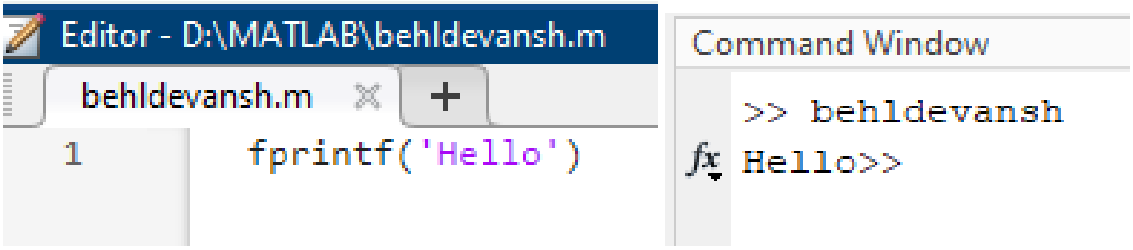
Q-4 The meaning of "a = b", continued

Ans-



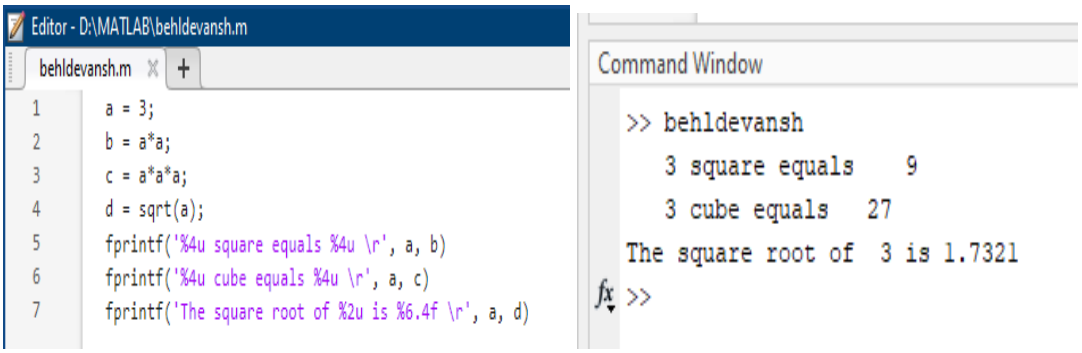
Q-5 Formatted output

Ans-



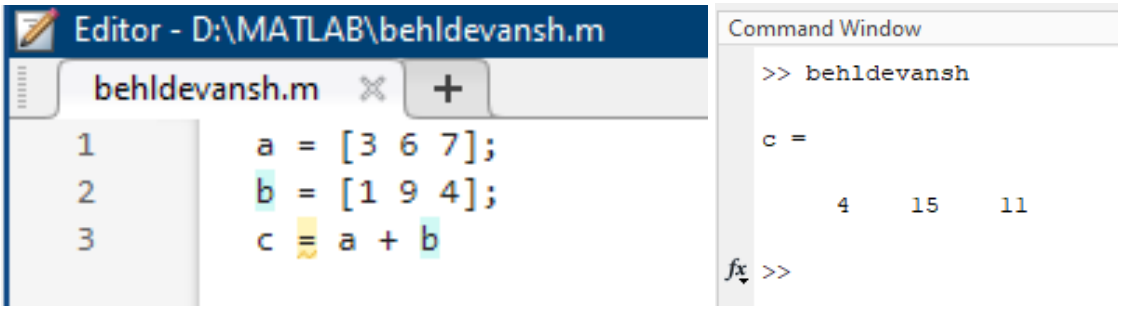
Q-6 Formatted output

Ans-



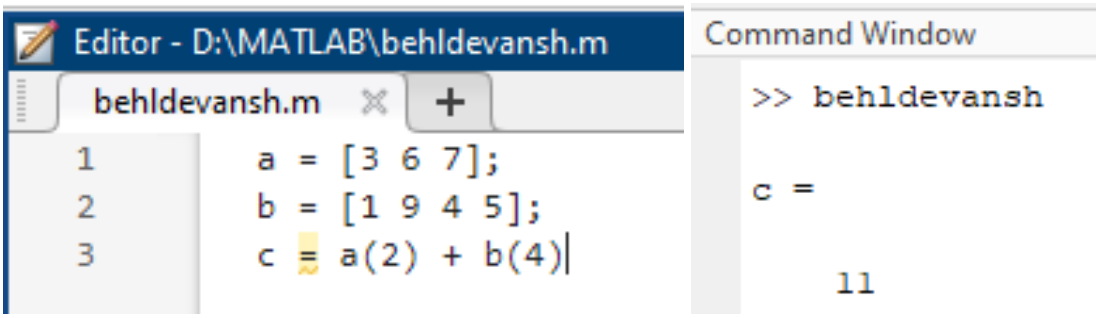
Q-7 Arrays

Ans-



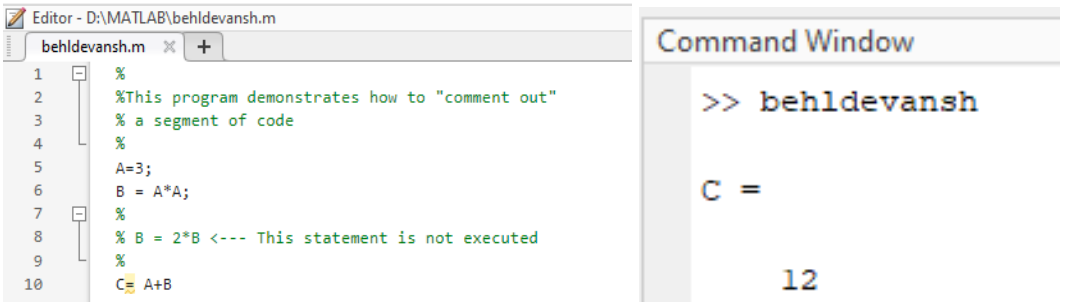
Q-8 Extracting an individual element of an array

Ans-



Q-9 Comment

Ans-



Q-10 Continuation to next line

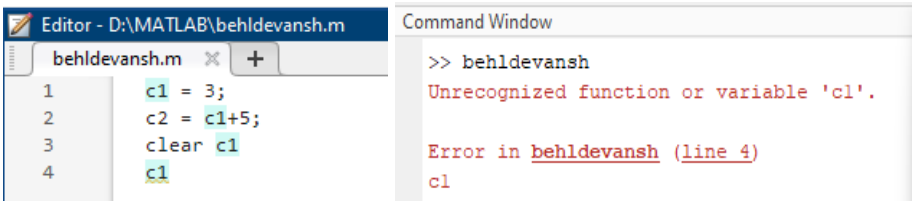
Ans-

summation1 = 1 + 3 + 5 + 7 ...+ 9 + 11

Note: The three periods (...) allow continuation to the next line of commands. The two lines in the above example are essentially one line of "summation1 = 1+3+5+7+9+11".

Q-11 "Clear" a variable

Ans-



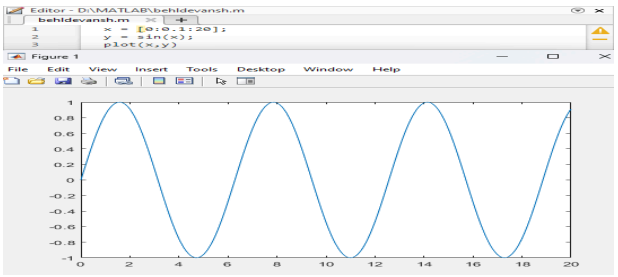
Q-12 Naming a variable

Ans-

(i) Matlab variables are case sensitive. For example, "ASU" and "asu" are two different variables. (ii) An underscore (_) or a number (0-9) can also be part of the name of a variable. For example, "MAE_384" is a legitimate variable name. (iii) Some names are reserved for special constants. For example (see Ex. 11), "pi" is an intrinsic constant with a fixed value of 3.14159...

Q-13 Making a quick plot

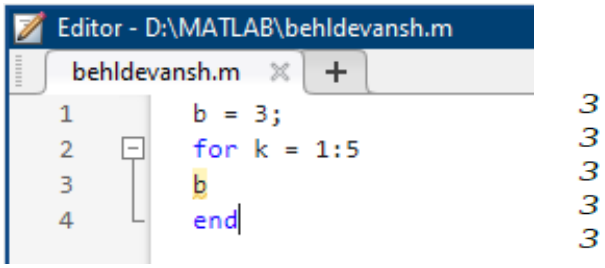
Ans-



Part 2. Basic looping

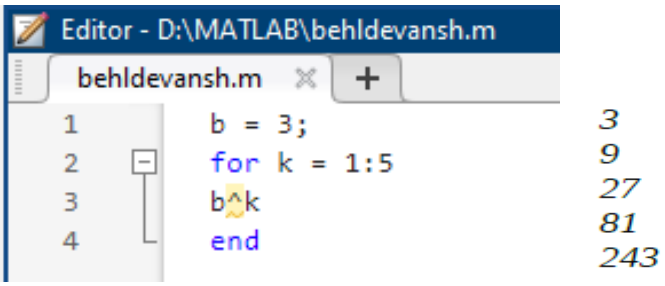
Q-14 Loop: Using for command

Ans-



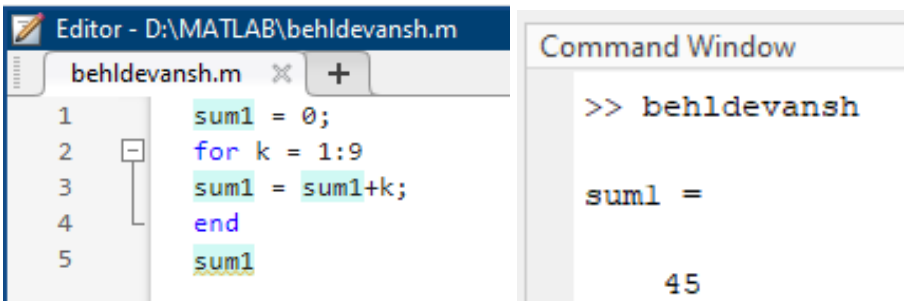
Q-15 For loop: Utility of the dummy index

Ans-



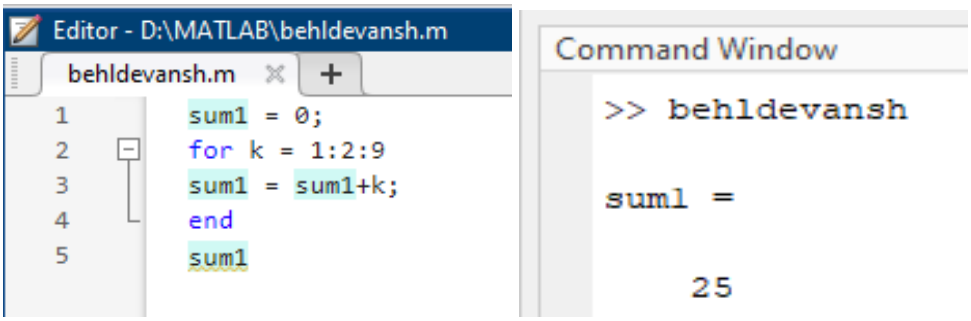
Q-16 For loop: More on the dummy index

Ans-



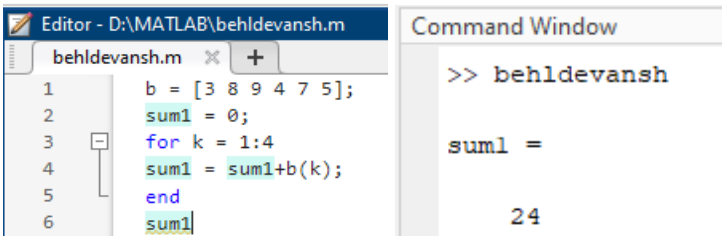
Q-17 For loop: More on the dummy index

Ans-



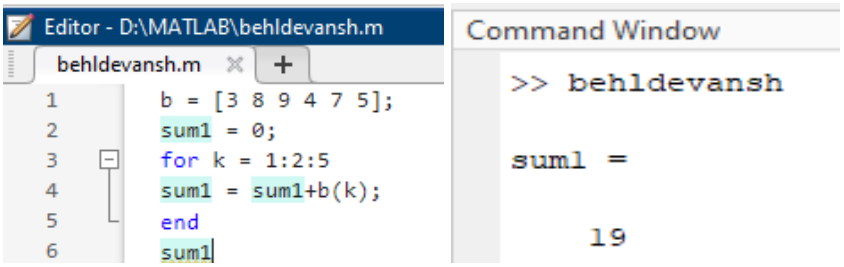
Q-18 Treatment of array within a loop

Ans-



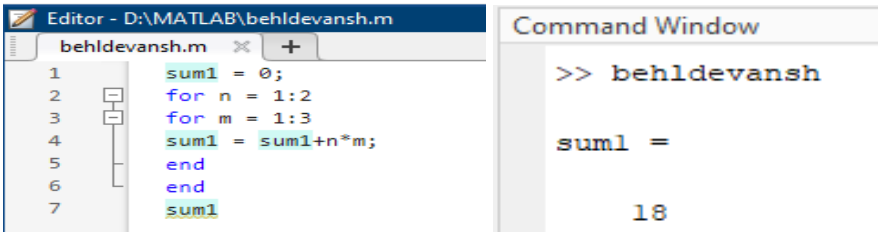
Q-19 Treatment of array within a loop

Ans-



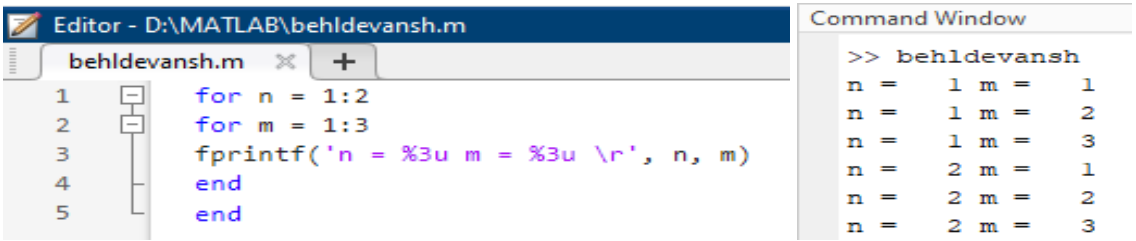
Q-20 Double loop

Ans-



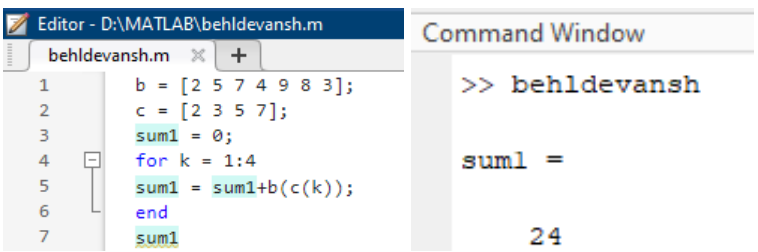
Q-21 Double loop

Ans-



Q-22 More complicated use of loop and index

Ans-



Part 3. Basic branching

Q-23 The if command

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
1  num1 = 7;
2  if (num1 > 5)
3  fprintf('%4u is greater than 5 \r', num1)
4  else
5  fprintf('%4u is less than or equal to 5 \r', num1)
6  end
```

Command Window

```
>> behldevansh
   7 is greater than 5
```

Q-24 if - elseif - else

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
1  num1 = 4;
2  if (num1 >= 5)
3  fprintf('%4i is greater than or equal to 5 \r', num1)
4  elseif (num1 > 1)
5  fprintf('%4i is less than 5 but greater than 1 \r', num1)
6  elseif (num1 == 1)
7  fprintf('%4i equals 1 \r', num1)
8  elseif (num1 > -3)
9  fprintf('%4i is less than 1 but greater than -3 \r', num1)
10 else
11 fprintf('%4i is less than or equal to -3 \r', num1)
12 end
```

Command Window

```
>> behldevansh
   4 is less than 5 but greater than 1
```

Q-25 An application - determine whether a given year is a leap year

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

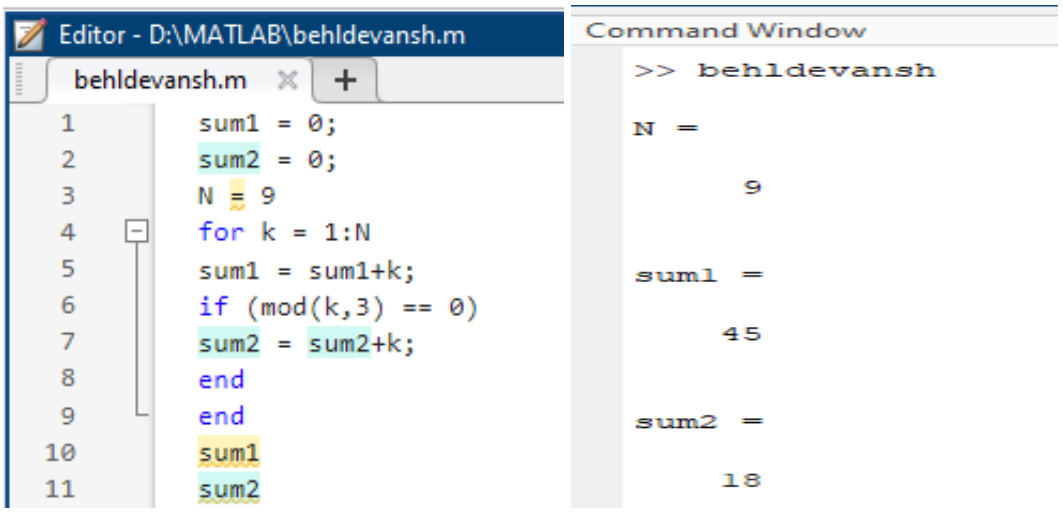
```
1  nyear = 1975;
2  if (mod(nyear, 400) == 0)
3  fprintf('%6u is a leap year', nyear)
4  elseif (mod(nyear,4) == 0) & (mod(nyear,100) ~= 0)
5  fprintf('%6u is a leap year', nyear)
6  else
7  fprintf('%6u is not a leap year', nyear)
8  end
```

Command Window

```
>> behldevansh
  1975 is not a leap year>>
```

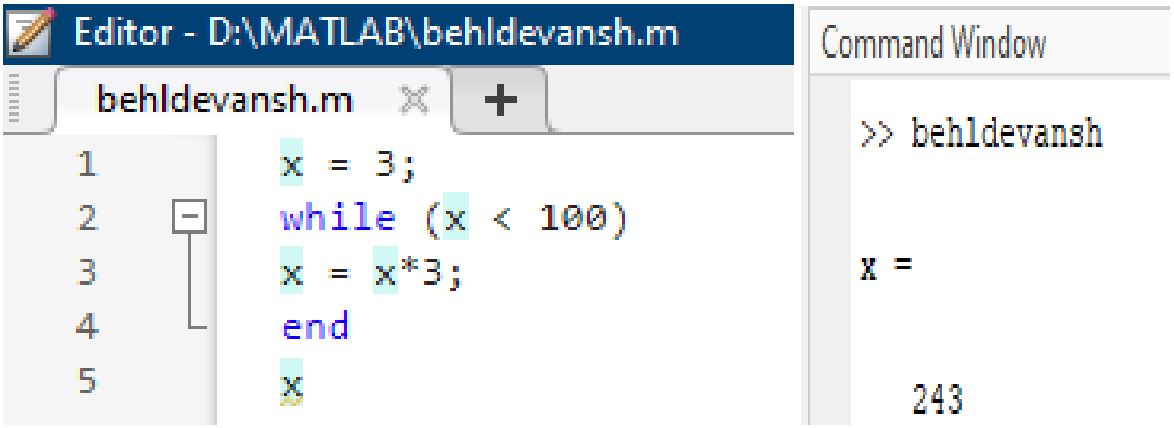
Q-26 Combine looping and branching

Ans-



Q-27 The while loop

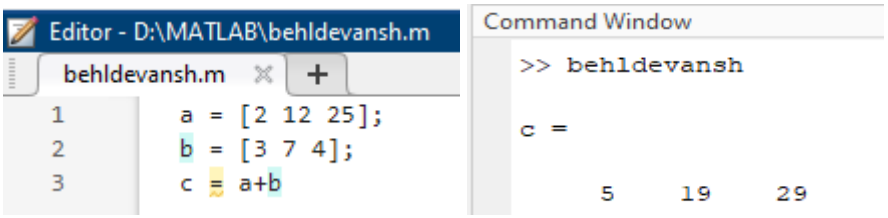
Ans-



Part 4. Array and Matrix

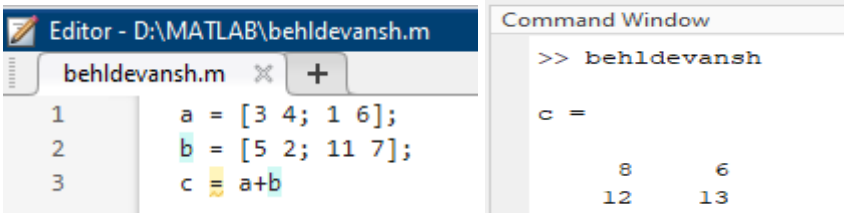
Q-28 Assign the content of a (one-dimensional) array; Addition of two arrays

Ans-



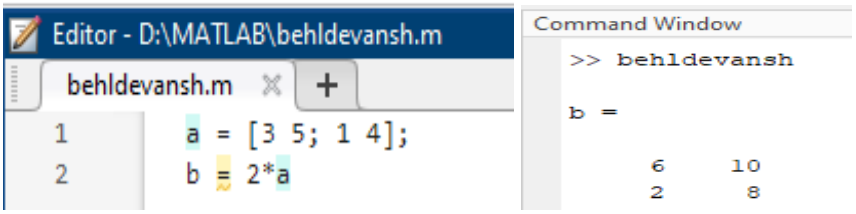
Q-29 Assign the content of a matrix; Addition of two matrices

Ans-



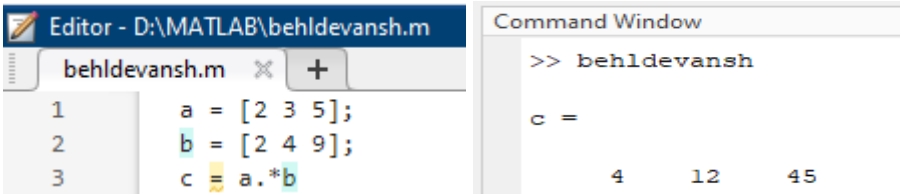
Q-30 Multiplication involving a scalar and an array (or a matrix)

Ans-



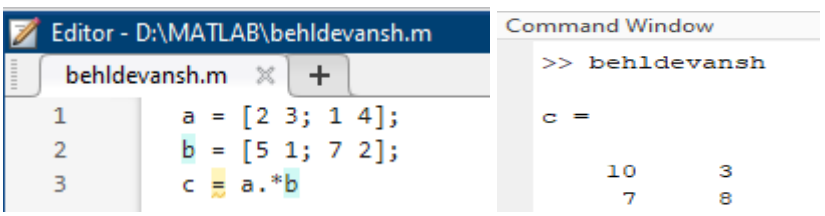
Q-31 Element-by-element multiplication involving two 1-D arrays or two matrices of the same dimension

Ans-



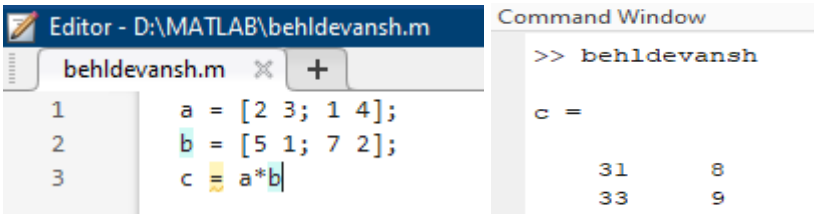
Q-32 Element-by-element multiplication of two matrices

Ans-



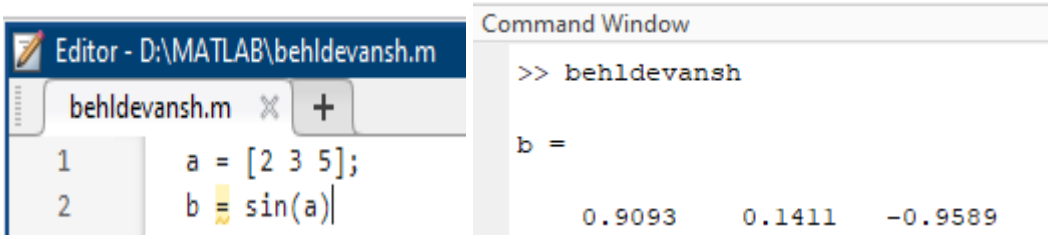
Q-33 Direct (not element-by-element) multiplication of two matrices

Ans-



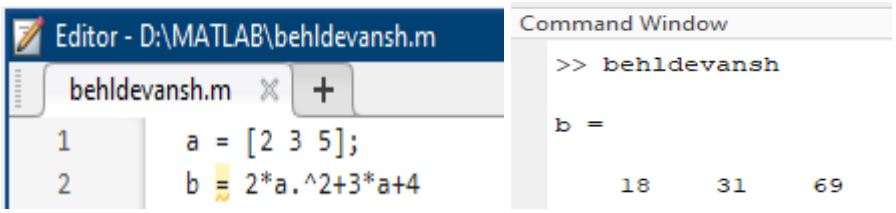
Q-34 Elementary functions with a vectorial variable

Ans-



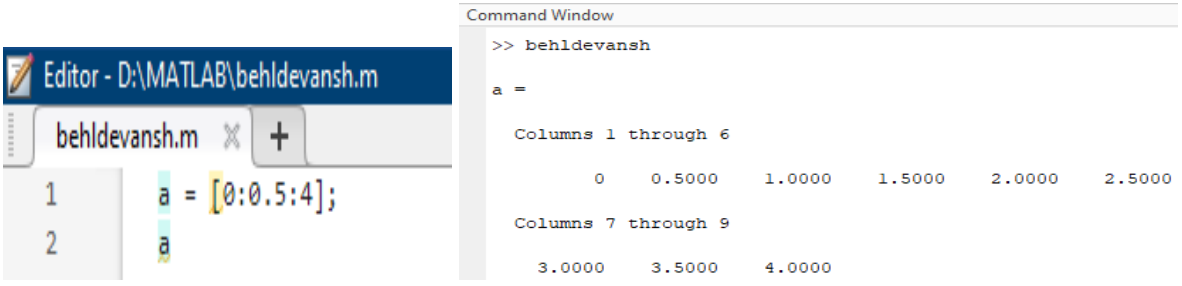
Q-35 Another example of elementary functions with a vectorial variable

Ans-



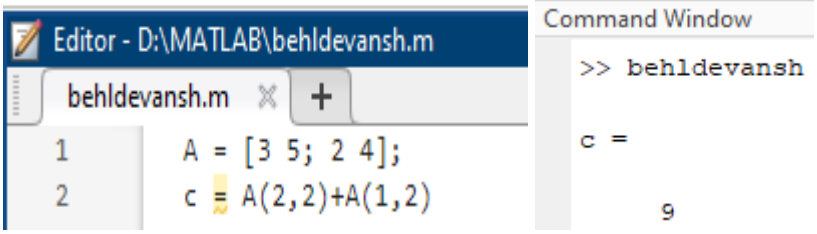
Q-36 An efficient way to assign the content of an array

Ans-



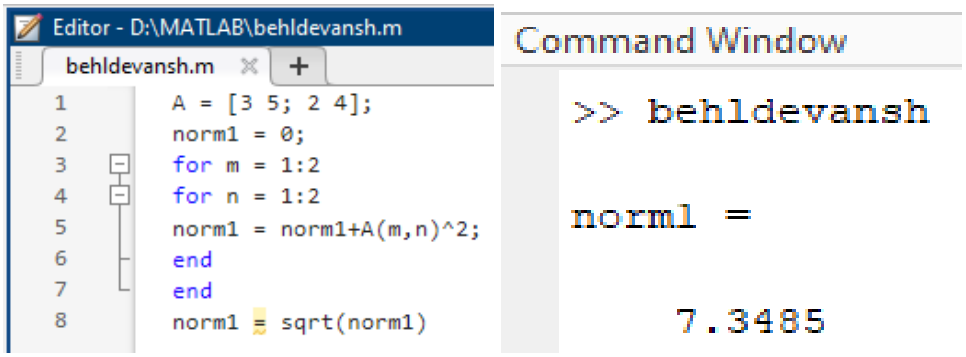
Q-37. Extracting the individual element(s) of a matrix

Ans-



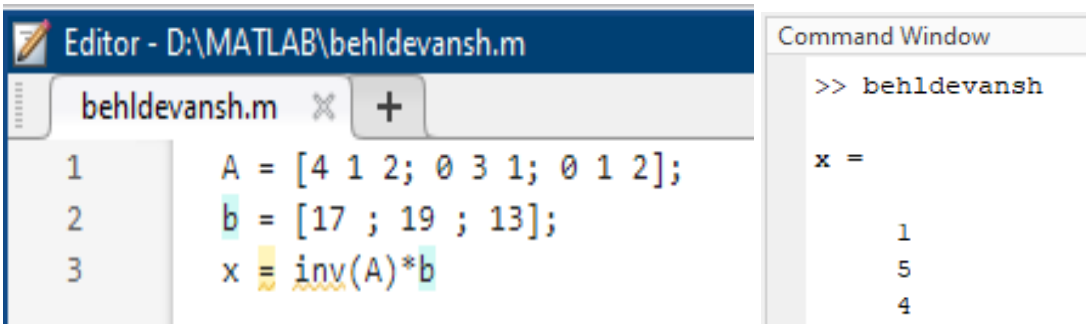
Q-38 Another example for the usage of index for a matrix

Ans-



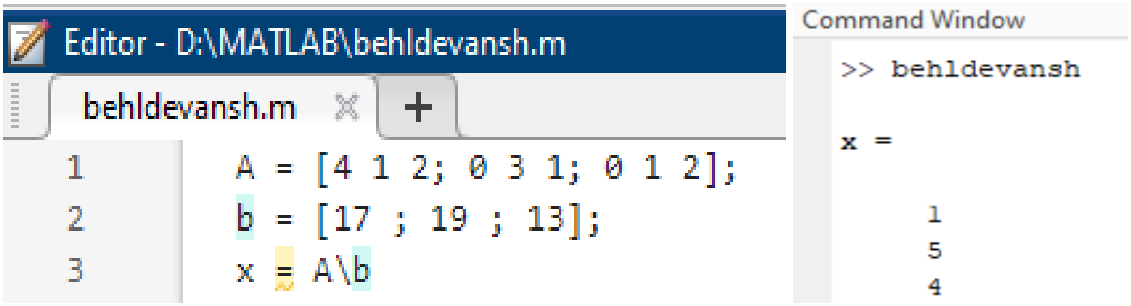
Q-39 Solving a system of linear equation

Ans-



Q-40 An alternative to Ex. 39

Ans-

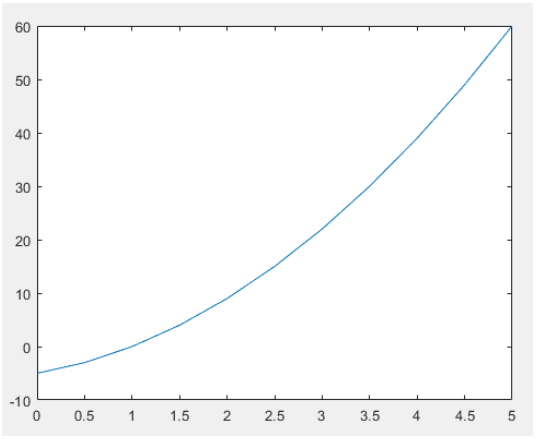


Part 5. Basic graphic applications

Q-41 A quick example of plot command: Draw a curve

Ans-

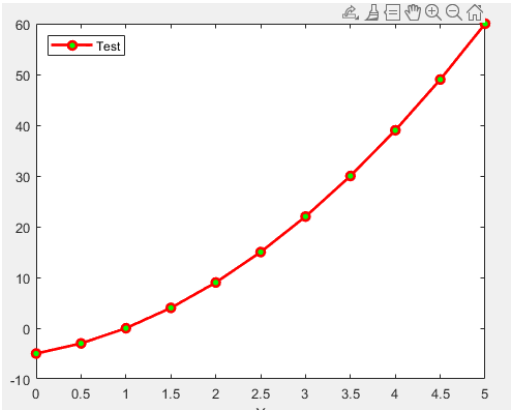
```
Editor - D:\MATLAB\behldevansh.m
behldevansh.m
1  a = [0:0.5:5];
2  b = 2*a.^2 + 3*a -5;
3  plot(a,b)
4
```



Q-42 Refine the plot: Line pattern, color, and thickness

Ans-

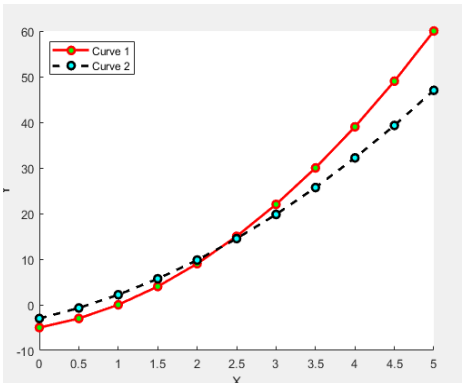
```
Editor - D:\MATLAB\behldevansh.m
behldevansh.m
1  a = [0:0.5:5];
2  b = 2*a.^2 + 3*a -5;
3  plot(a,b,'-or','MarkerFaceColor','g','LineWidth',2)
4  xlabel('X'); ylabel('Y'); legend('Test','Location','NorthWest
```



Q-43 Draw multiple curves

Ans-

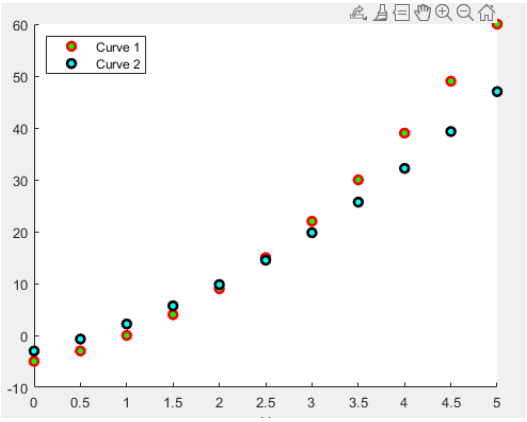
```
Editor - D:\MATLAB\behldevansh.m
behldevansh.m
1  a = [0:0.5:5];
2  b = 2*a.^2 + 3*a -5;
3  c = 1.2*a.^2+4*a-3;
4  hold on
5  plot(a,b,'-or','MarkerFaceColor','g','LineWidth',2)
6  plot(a,c,'--ok','MarkerFaceColor','c','LineWidth',2)
7  xlabel('X'); ylabel('Y'); legend('Curve 1','Curve 2','Locatio
8
```



Q-44 Draw symbols

Ans-

```
a = [0:0.5:5];
b = 2*a.^2 + 3*a -5;
c = 1.2*a.^2+4*a-3;
hold on
plot(a,b,'or','MarkerFaceColor','g','LineWidth',2)
plot(a,c,'ok','MarkerFaceColor','c','LineWidth',2)
xlabel('X'); ylabel('Y'); legend('Curve 1','Curve 2','Loca
```



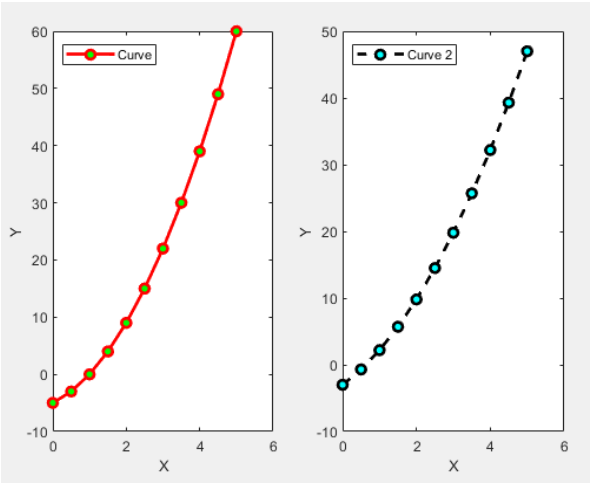
Q-45 Plot with multiple panels

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
1  a = [0:0.5:5];
2  b = 2*a.^2 + 3*a -5;
3  c = 1.2*a.^2+4*a-3;
4  subplot(1,2,1)
5  plot(a,b,'-or','MarkerFaceColor','g','LineWidth',2)
6  xlabel('X'); ylabel('Y'); legend('Curve ','Location','NorthW
7  subplot(1,2,2)
8  plot(a,c,'--ok','MarkerFaceColor','c','LineWidth',2)
9  xlabel('X'); ylabel('Y'); legend('Curve 2','Location','North
```



Part 6 External (user-defined) function

Q-46 How to construct and use an external function

Ans-

myfunc1.m

```
function outcome = myfunc1(x)
outcome = 2*x^2 + 3*x + 7;
```

mainprog1.m

```
for n = 1:5
    x = n*0.1;
    z = myfunc1(x);
    fprintf('x = %4.2f  f(x) = %8.4f \r',x,z)
end
```

```
x = 0.10  f(x) =  7.3200
x = 0.20  f(x) =  7.6800
x = 0.30  f(x) =  8.0800
x = 0.40  f(x) =  8.5200
x = 0.50  f(x) =  9.0000
```

Q-47 A function with multiple input parameters

Ans-

myfunc2.m

```
function outcome = myfunc2(x,a,b,c)
outcome = a*x^2 + b*x + c;
```

mainprog2.m

```
for n = 1:5
    x = n*0.1;
    z = myfunc2(x,2,3,7);
    fprintf('x = %4.2f  f(x) = %8.4f \r',x,z)
end
```

```
x = 0.10  f(x) =  7.3200
x = 0.20  f(x) =  7.6800
x = 0.30  f(x) =  8.0800
x = 0.40  f(x) =  8.5200
x = 0.50  f(x) =  9.0000
```

Part 7 Use external files and prompt for input and output

Q-48 Open a file and write the output to the file

Ans-

Editor - D:\MATLAB\behldevansh.m *

behldevansh.m *

```
fID1 = fopen('myoutput1.txt','w');
for n = 1:4
b1 = n ; b2 = n^2 ; b3 = n^3;
fprintf(fID1,'%7u %7u %7u \r',b1,b2,b3);
end
```

```
1    1    1
2    4    8
3    9   27
4   16   64
```

Q-49 Read data from an existing file

Ans-

Editor - D:\MATLAB\behldevansh.m *

behldevansh.m *

```
fID1 = fopen('myoutput1.txt','r');
for n = 1:4
b = fscanf(fID1,'%7u %7u %7u \r',3);
btotal = b(1)+b(2)+b(3);
fprintf('%7u + %7u + %7u = %7u \r', b(1), b(2), b(3), btotal)
end
```

```
1 +    1 +    1 =    3
2 +    4 +    8 =   14
3 +    9 +   27 =   39
4 +   16 +   64 =   84
```

Q-50 Create a prompt to request user input

Ans-

Editor - D:\MATLAB\behldevansh.m *

behldevansh.m *

```
num1 = input('Enter your age');
if (num1 > 17)
fprintf('You are eligible to vote')
else
fprintf('You are not eligible to vote')
end
```

MATLAB Assignment-2

Q-1 WAP in MATLAB to find the square, cube and square root of any number. The output should look like this.

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
1  a = input('enter the number n\n')
2  b = a*a;
3  c = a*a*a;
4  d = sqrt(a);
5  fprintf('%4u square equals %4u \r', a, b)
6  fprintf('%4u cube equals %4u \r', a, c)
7  fprintf('The square root of %2u is %6.4f \r', a, d)
```

Command Window

```
>> behldevansh
enter the number n
3

a =

     3

   3 square equals    9
   3 cube equals   27
The square root of  3 is 1.7321
```

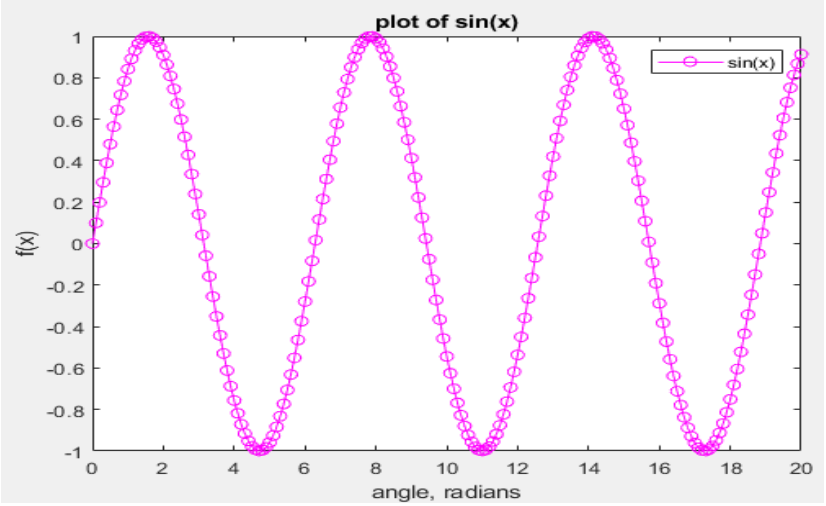
Q-2 WAP to plot the function sinx for values of x from 0 to 20 in increments of 0.1. The title of the plot should be 'plot of sin(x)'. The axes should be labeled, and the legend should show sin(x) with magenta color and circles as marker

Ans-

Editor - D:\MATLAB\behldevansh.m

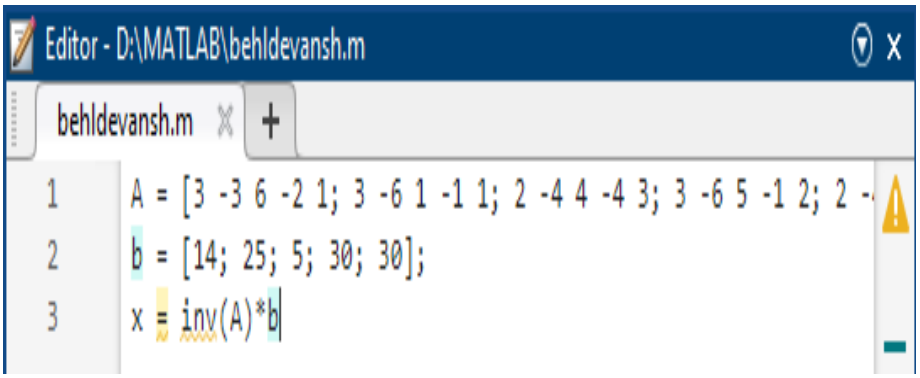
behldevansh.m

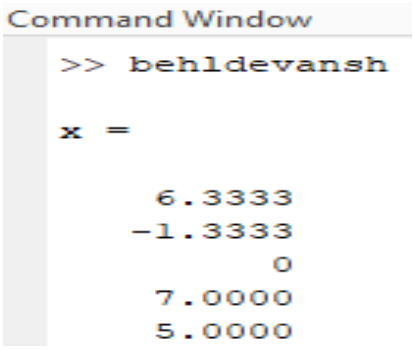
```
1  x = [0:0.1:20];
2  y = sin(x);
3  plot(x,y)
4  plot(x,y,'mo-')
5  title('plot of sin(x)');
6  xlabel('angle, radians')
7  ylabel('f(x)')
8  legend('sin(x)');
```



Q-3 Solving Simultaneous Equations:

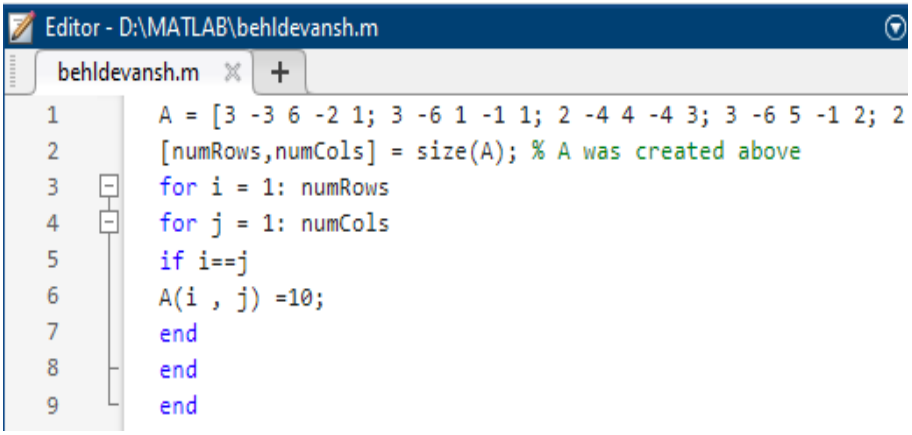
Ans-





Q-4 WAP to make all the diagonal elements of a matrix equal to 10.

Ans-



A =

10	-3	6	-2	1
3	10	1	-1	1
2	-4	10	-4	3
3	-6	5	10	2
2	-4	9	1	10

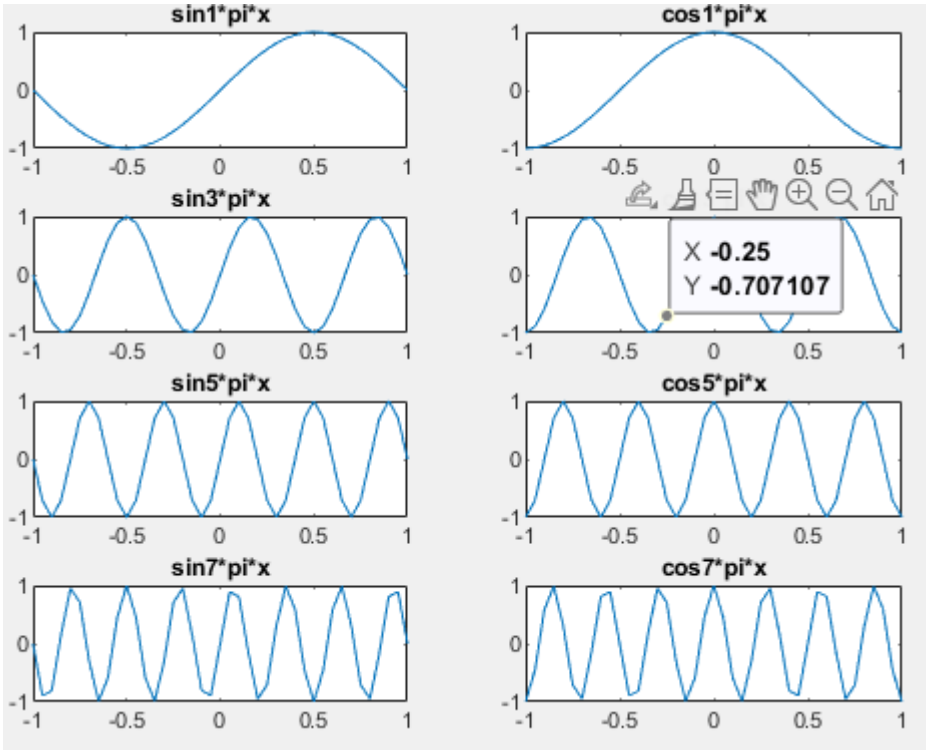
Q-5 WAP to create a figure with 8 subplots of 4 rows and 2 columns. First column shows the plots of sinx, sin3x, sin5x and sin7x, and the second column shows plots of cosx, cos3x, cos5x and cos7x.

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
1   x = -1:.05:1;
2   for n = 1:2:8
3   subplot(4,2,n), plot(x,sin(n*pi*x)),title(['sin',num2str(n
4   subplot(4,2,n+1), plot(x,cos(n*pi*x)),title(['cos',num2str
5   end
```



MATLAB Assignment-3

Q-1 Bob fires a cannon horizontally off a hill. The following data is the height of the cannon ball above the ground at different times. The sensor used to measure the height of the cannon above the ground is buggy. Write a function that removes any data point that is greater than the previous data point. Fit a second order polynomial to this data. Plot the cleaned data as data points and the fitted line on the same graph and add a legend, axis labels and a title. Calculate the derivative of the data (the speed of the ball in the y direction) using the data from the line of best fit. On a second plot, plot the derivative as points. Publish the script as a pdf.

Data:

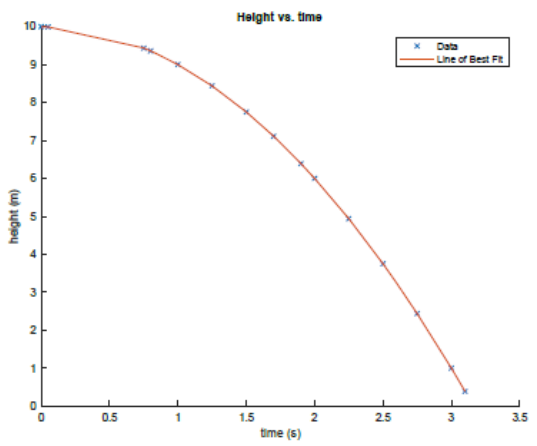
Time (ms)	0	50	750	800	825	1000	1250	1500	1700	1900	2000	2250	2500	2600	2750	3000	3100
Height (m)	10	9.99	9.43	9.36	9.5	9.0	8.44	7.75	7.11	6.39	6.0	4.94	3.75	5.0	2.44	1.0	0.39

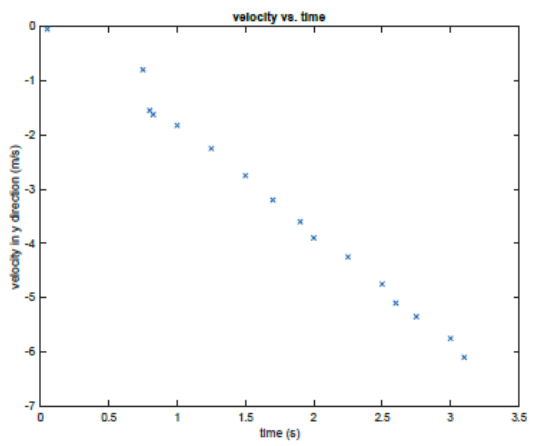
Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
%Practice Test Script
%time data converted from ms to s
time=[0,50,750,800,825,1000,1250,1500,1700,1900,2000,2250,2500,2600,2750,3000,3100]*1e-3;
%height in m
height=[10,9.99,9.43,9.36,9.5,9.0,8.44,7.75,7.11,6.39,6.0,4.94,3.75,5.0,2.44,1.0,0.39];
[heightClean,timeClean]=cleanUp(height,time); %call the function to
cleanup the height Data
%fit a second order polynomial to the data
P=polyfit(timeClean,heightClean,2);
heightFit=polyval(P,time);
%plot the cleaned data as data points and line of best fit
figure
hold on
plot(timeClean,heightClean, 'x');
plot(time,heightFit);
xlabel('time (s)');
ylabel('height (m)');
title('Height vs. time');
legend('Data','Line of Best Fit');
%Calculate Derivative
dhdt=diff(heightFit)./diff(time);
figure
plot(time(2:end),dhdt, 'x');
xlabel('time (s)');
ylabel('velocity in y direction (m/s)' )
title('velocity vs. time');
```





Q-2 Let us plot projectile trajectories using equations for ideal projectile motion:

$$\begin{aligned} y(t) &= y_0 - \frac{1}{2}gt^2 + (v_0 \sin(\theta_0))\, t, \\ x(t) &= x_0 + (v_0 \cos(\theta_0))\, t, \end{aligned}$$

where y(t) is the vertical distance and x(t) is the horizontal distance traveled by the projectile in metres, g is the acceleration due to Earth's gravity (9.8 m/s2) and t is time in seconds. Let us assume that the initial velocity of the projectile is 50.75 m/s and the projectile's launching angle is 5π/12 radians. The initial vertical and horizontal positions of the projectile are 0 m. Let us now plot y vs. t and x vs. t in two separate graphs with the vector:

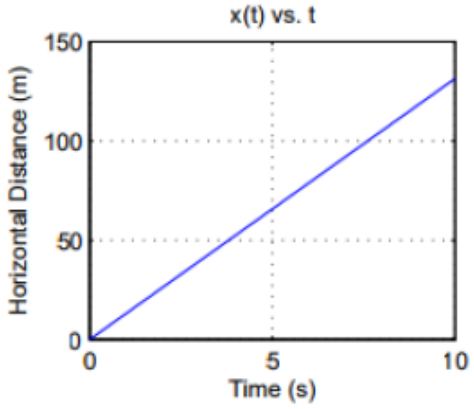
t = 0:0.1:10 representing time in seconds. Give appropriate titles to the graphs and label the axes. Make sure the grid lines are visible.

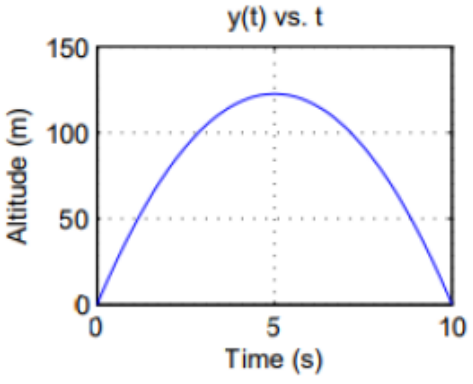
Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
t = 0 : 0.1 : 10;
g = 9.8;
v0 = 50.75;
theta0 = 5*pi/12;
y0 = 0;
x0 = 0;
y = y0 - 0.5 * g * t.^2 + v0*sin(theta0).*t;
x = x0 + v0*cos(theta0).*t;
figure 1;
plot(t,x);
title('x(t) vs. t');
xlabel('Time (s)');
ylabel('Horizontal Distance (m)');
grid on;
figure 2;
plot(t,y);
title('y(t) vs. t');
xlabel('Time (s)');
ylabel('Altitude (m)');
grid on;
```





Q-3-You are given a text file called ballSamples that contains the following information:

a. The first column contains the ball number starting at 1

b. The second column contains the masses of the ball

c. The third column contains the colour of the ball, with 0 = red, 1 = blue, and 2 = yellow

Write a script that reads the information from the text file and:

I. Groups each ball by colour into a 2D array that contains the ball number in the first row and the mass in the second row

II. Displays the average mass of each group

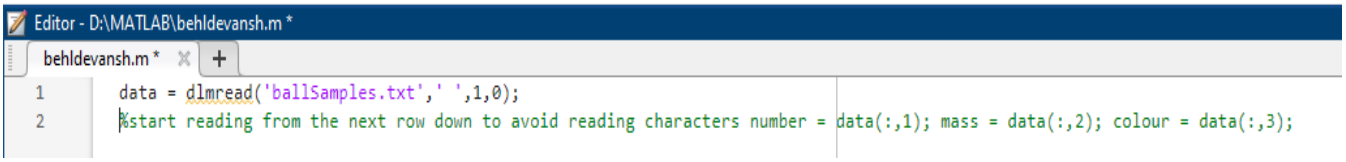
III. Finds the ball number of the heaviest red ball and the lightest blue ball

IV. Prints out which group has the largest number of balls, and the number of balls in that specific group

Assume that there is some sort of header for each column in the first line of the file, and that each column is separated by spaces.

Ans-

//Read the File



//sort the balls


```
red = zeros(2,length(data));
rcount = 1;
blue = zeros(2,length(data));
bcount = 1;
yellow = zeros(2,length(data));
ycount = 1;
for i = 1:length(number)
    if colour(i) == 0
        red(1,rcount) = number(i,1);
        red(2,rcount) = mass(i,1);
        rcount = rcount+1;
    elseif colour(i) == 1
        blue(1,bcount) = number(i,1);
        blue(2,bcount) = mass(i,1);
        bcount = bcount+1;
    else
        yellow(1,ycount) = number(i,1);
        yellow(2,ycount) = mass(i,1);
        ycount = ycount+1;
    end
end
red = red(:,rcount-1);
```

```
blue = blue(:,bcount-1);
yellow = yellow(:,ycount-1);
ravg = sum(red(2,:))/length(red);
bavg = sum(blue(2,:))/length(blue);
yavg = sum(yellow(2,:))/length(yellow);
fprintf('average of red masses: %f0.2 | average of blue masses: %f0.2 | average of yellow masses:
%f0.2',ravg,bavg,yavg);
```

//Find the heaviest red ball and lightest blue ball


```
max = 0;
max_i = 1;
min = 1000;
min_i = 1;
for i = 1:rcount-1
    if red(2,i)>max
        max = red(2,i);
        max_i = i;
    end
end

for i = 1:bcount-1
    if blue(2,i)<min
        min = blue(2,i);
        min_i = i;
    end
end
fprintf('The heaviest red ball is number %d, and it has a mass of %f. The lightest blue ball is
%d, and it has a mass of %f.',max_i,max,min_i,min);
```

//Find the biggest group

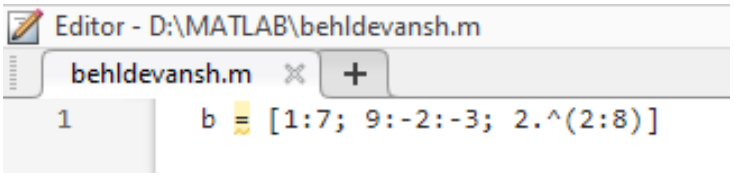

```
[val,idx] = max([rcount-1,bcount-1,ycount-1]);
if idx == 1
    fprintf('The largest group is the red group. It has %d balls', val);
elseif idx ==2
    fprintf('The largest group is the blue group. It has %d balls', val);
else
    fprintf('The largest group is the yellow group. It has %d balls', val);
```

MATLAB Assignment-4

Q-1 Find a short MATLAB expression to build the matrix

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 9 & 7 & 5 & 3 & 1 & -1 & -3 \\ 4 & 8 & 16 & 32 & 64 & 128 & 256 \end{pmatrix}$$

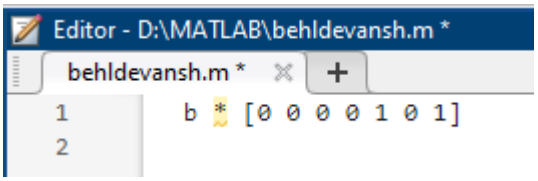
Ans-



Q-2 Give a MATLAB expression that uses only a single matrix multiplication with B to obtain

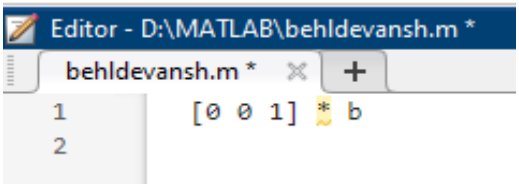
(a) the sum of columns 5 and 7 of B

Ans-



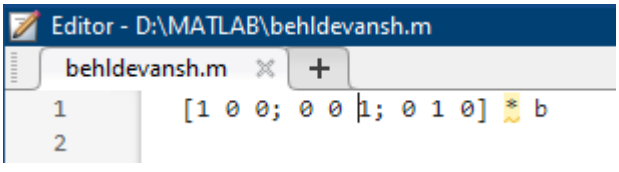
(b) the last row of B

Ans-



(c) a version of B with rows 2 and 3 swapped

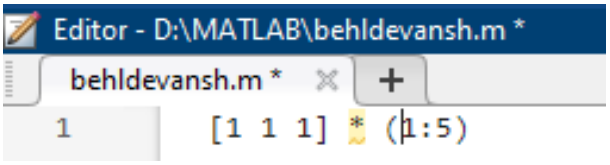
Ans-



Q-3 Give a MATLAB expression that multiplies two vectors to obtain

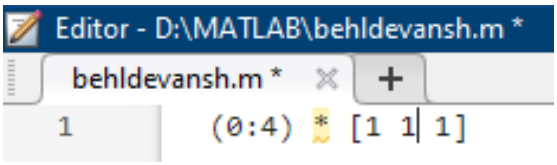
$$(a)\ \text{the matrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

Ans-



$$(b)\ \text{the matrix} \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$$

Ans-



Q-5

(a) Write down the function g(t) that has the shape of a sine wave that increases linearly in frequency from 0 Hz at t = 0 s to 5 Hz at t = 10 s.

Ans-

Answer: The instantaneous frequency of function $g(t)$ at time t is

$$f(t) = t \cdot \frac{5\ \text{Hz}}{10\ \text{s}} = \frac{t}{2\ \text{s}^2}$$

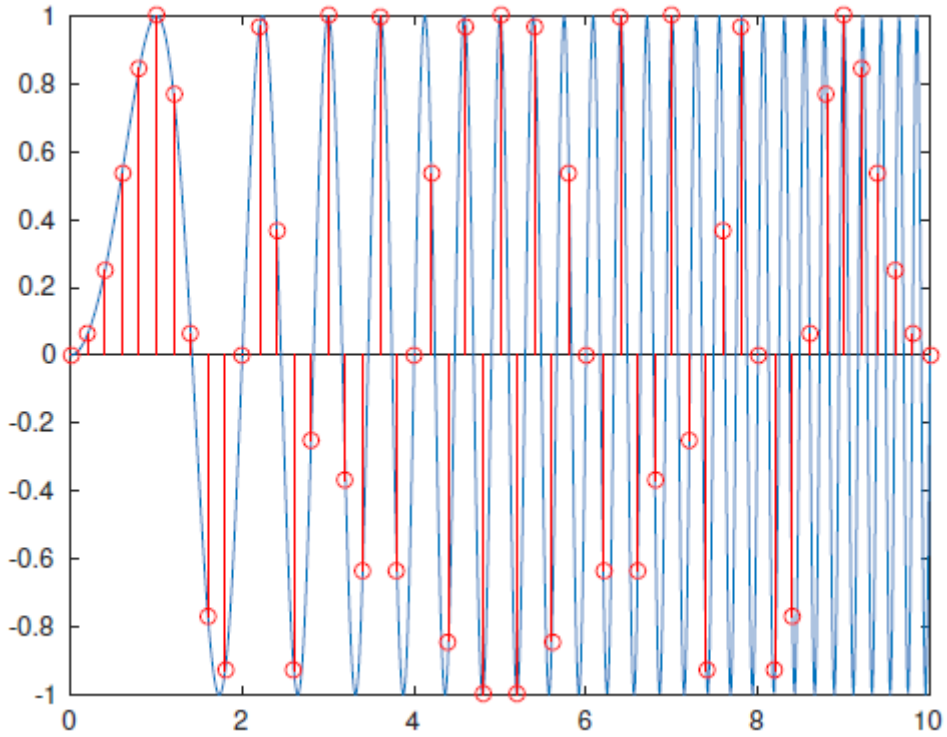
and since the phase of a sine wave is 2π times the integrated frequency so far, we get

$$g(t) = \sin\left(2\pi \int_0^t f(t')\,\mathrm{d}t'\right) = \sin\left(2\pi \frac{t^2}{4\ \text{s}^2}\right) = \sin\left(\frac{\pi t^2}{2\ \text{s}^2}\right)$$

(b) Plot the graph of this function using MATLAB's plot command.

(c) Add to the same gure (this can be achieved using the hold command) in a different colour a graph of the same function sampled at 5 Hz, using the stem command.

Ans-

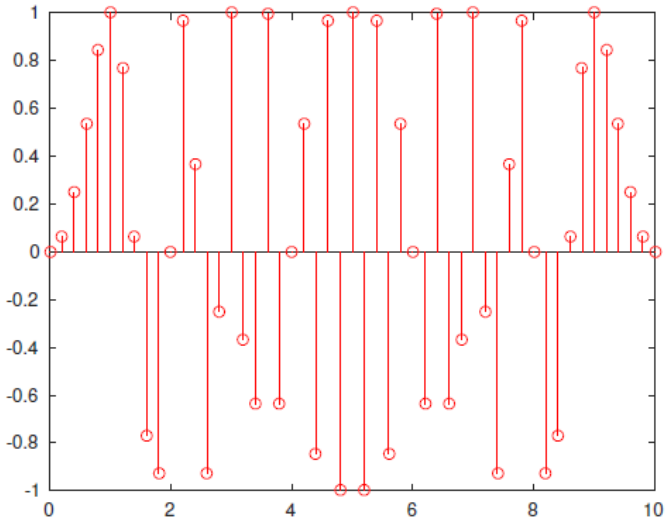


(d) Plot the graph from (c) separately. Can you explain its symmetry?

Ans-

A sine wave with a frequency f larger than half the sampling frequency fs cannot be

distinguished based on the sample values from a sine wave of frequency fs □ f. In other words, the sample values would have looked the same had we replaced the instantaneous frequency f(t) with fs=2 □ jfs=2 □ f(t)j, and the latter is symmetric around fs=2, which is in this graph 2.5 Hz and occurs at t = 5 s.



Q-5

Use MATLAB to write an audio waveform (8 kHz sampling frequency) that contains a sequence of nine tones with frequencies 659, 622, 659, 622, 659, 494, 587, 523, and 440 Hz. Then add to this waveform a copy of itself in which every other sample has been multiplied by □1. Play the waveform, write it to a WAV le, and use the specgram command to plot its spectrogram with correctly labelled time and frequency axis.

Ans-

Editor - D:\MATLAB\behldevansh.m

behldevansh.m

```
f = [659 622 659 622 659 494 587 523 440];
fs = 8000;  % sampling frequency
d = 0.5;    % duration per tone
t = 0:1/fs:d-1/fs;
w = sin(2 * pi * f' * t)/2;
w = w'; w = w(:)';
w = [w, w .* (mod((1:length(w)), 2) * 2 - 1)];
audiowrite('matlab_answer-2.wav', w, fs);
specgram(w, [], fs);
```