

EXPERIMENT: NO: 1

NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

AIM: Introduction to

- a. MATLAB environment
- b. Plotting basic graphs
- c. Evaluating basic mathematical expressions

a. MATLAB environment

MATLAB is an interactive mathematical program that allows mathematical, statistical, etc. calculations as well visualization of data. To solve many of the engineering problems, you will find MATLAB to be a powerful tool to use. MATLAB has hundreds of built-in functions and can be used to solve problems ranging from the very simple to the sophisticated and complex. Whether you want to do some simple numerical or statistical calculations, some complex statistics, solve simultaneous equations, make a graph, or run an entire simulation program, MATLAB can be an effective tool.

Command window – this is where MatLab commands are entered.

Workspace – lists all current variables

Directory – lists the files in the current working folder

Command history – lists all previous commands

Some Basic MatLab Commands:

- | | |
|--------------|--|
| • clear all | Clears <i>workspace</i> of all variables |
| • close all | Closes all figure and plot windows |
| • plot (x,y) | Plots vector “y” verses “x” |
| • % | Separates comments from script |
| • help | Used to find the syntax of a command |

“help” is used in the following manner: help <command>, where <command> is a MatLab command.

Basic Arithmetic Operators

+ : Arithmetic addition
- : Arithmetic subtraction
* : Arithmetic multiplication
/ : Arithmetic division
^ : Exponent or power ($3^4 = 3^4$)
.* : Element by element (for arrays)

Built-in Waveform Functions

- cos(): cosine
- sin(): sine
- square(): generates a square wave
- sawtooth(): generates a sawtooth wave
- sawtooth(t,0.5): generates a triangle wave

b. Plotting basic graphs

Quite often, we want to plot a set of ordered pairs. This is a very easy task with the MATLAB **plot(x, y)** command that plots y versus x , where x is the horizontal axis (abscissa) and y is the vertical axis (ordinate).

To use the ‘plot’ function in Matlab, first make sure that the matrices/vectors are of equal dimensions. For example, to plot vector $X = [3 \ 9 \ 27]$ vs time, the vector for time would also need to be a 1x3 vector (i.e. $t = [1 \ 2 \ 3]$).

Syntax

```
clear all           % clear all previous variables
X = [3 9 27];       % my dependent vector of interest
t = [1 2 3];         % my independent vector
figure              % create new figure
plot(t, X)
```

Labelling Axes

```
plot(t,X); grid; xlabel('TIME (Sec)'); ylabel('DISTANCE (M)'); title('Plot for Distance vs Time')
```

Legends

For multiple plots, the same can be distinguished from each other via a legend, the syntax is very similar to the axis labelling above. It is also possible to set colors for the different vectors and to change the location of the legend on the figure.

```
X = [3 9 27]; % dependent vectors of interest
Y = [10 8 6];
Z = [4 4 4];
t = [1 2 3]; % independent vector
```

```
figure
hold on % allow all vectors to be plotted in same figure
plot(t, X, 'blue', t, Y, 'red', t, Z, 'green')
```

Subplots

```
% subplot (nrows,ncols,plot_number)
x=0:.1:2*pi; % x vector from 0 to 2*pi, dx = 0.1
subplot(2,2,1); % plot sine function
Plot(x,sin(x));
subplot(2,2,2); % plot cosine function
plot(x,cos(x));
subplot(2,2,3) % plot negative exponential function
plot(x,exp(-x));
subplot(2,2,4); % plot x^3
plot(x, x.^3);
```

Create Line Plot

Create a two-dimensional line plot using the **plot** function. For example, plot the value of the sine function from 0 to 2π .

```
x = linspace(0,2*pi,100);
y = sin(x);
plot(x,y)
```

Label the axes and add a title

```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```

Plot Multiple Lines

By default, MATLAB clears the figure before each plotting command. Use the **figure** command to open a new figure window. You can plot multiple lines using the **hold on** command. Until you use **hold off** or close the window, all plots appear in the current figure window.

```
figure
x = linspace(0,2*pi,100);
```

```
y = sin(x);  
plot(x,y)  
hold on  
y2 = cos(x);  
plot(x,y2)  
hold off
```

Change Line Appearance

We can change the line color, line style, or add markers by including an optional line specification when calling the **plot** function. For example:

- **'.'** plots a dotted line.
- **'g.'** plots a green, dotted line.
- **'g.*'** plots a green, dotted line with star markers.
- **'*'** plots star markers with no line.

The symbols can appear in any order. You do not need to specify all three characteristics (line color, style, and marker). For more information about the different style options, see the **plot** function page.

For example, plot a dotted line. Add a second plot that uses a dashed, red line with circle markers.

```
x = linspace(0,2*pi,50);  
y = sin(x);  
plot(x,y,':')  
hold on  
y2 = cos(x);  
plot(x,y2,'--ro')  
hold off
```

C. Evaluating basic mathematical expressions

```
a = 3;  
b = 5;  
c = a+b
```

Output:

8

The semicolon at the end of a statement acts to suppress output (to keep the program running in a "quiet mode"). (2) The third statement, $c = a+b$, is not followed by a semicolon so the content of the variable c is "dumped" as the output.

Basic math operations

```
a = 3;  
b = 9;  
c = 2*a+b^2-a*b+b/a-10
```

Output:

53

The right hand side of the third statement includes all 4 of the basic arithmetic operators, + (addition), - (subtraction), * (multiplication), and / (division), in their usual meaning. It also includes the symbol, ^, which means "to the power of", so "b^2" means (the content of b) to the power of 2, i.e., $9^2 = 81$. The right hand side of that statement is first evaluated: $\text{r.h.s.} = 2 \times 3 + 9^2 - 3 \times 9 + 9/3 - 10 = 53$. The content of r.h.s., now 53, is then assigned to the variable c in the left hand side. Since this statement is not followed by a semicolon, the content of c is dumped as the final output.

Formatted output

```
a = 3;  
b = a*a;  
c = a*a*a;  
d = sqrt(a);  
fprintf('%4u square equals %4u \r', a, b)
```

```
fprintf('%4u cube equals %4u \r', a, c)
fprintf('The square root of %2u is %6.4f \r', a, d)
```

Output:

3 square equals 9

3 cube equals 27

The square root of 3 is 1.7321

The command "fprintf" is for formatted output, using the format specified in the first string '...' in the parentheses. The "%4u" (4-digit integer) and "%6.4f" (real number that preserves 4 digits to the right of the floating point) are the format for the variable(s) for output. The "sqrt" in the 4th statement is the intrinsic function for square root.

Intrinsic math functions and constants

```
x = pi;
y = sin(pi/2)
z = exp(-sin(pi/2))
```

Output:

y = 1

z = 0.3679

Remarks: "pi" (= 3.14159...) is a reserved intrinsic constant. A function within a function is allowed. The innermost function will be evaluated first, so the 3rd statement leads to $z = \exp(-1) = 0.3679$.

```
a = [0:0.5:5];
b = 2*a.^2 + 3*a -5;
plot(a,b)
```

Exercise:

a. Find the roots of the equations $6x^5 - 41x^4 + 97x^3 - 97x^2 + 41x - 6$

b. Find the values of x,y,z of the equations $x+y+z=3$, $x+2y+3z=4$, $x+4y+9z=6$

c. For $f(x)=8x^8 - 7x^7 + 12x^6 - 5x^5 + 8x^4 + 13x^3 - 12x^2 + 9$ compute $f(2)$, roots of $f(x)$ and plot for $0 \leq x \leq 20$

EXERCISE BASED ON EXPERIMENT 2

NAME: - VARUN VERMA

BRANCH: - EEE

ROLL NO.: - 02211504919

**AIM: - INTRODUCTION TO MATRICES AND
VECTORS**

GIVEN NUMERICAL:

$$A = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 3 & 4 \\ -1 & 1 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 6 & 4 \\ -1 & 0 & 2 \end{bmatrix}$$

Where A and B are two 3 X 3 matrix.

Perform the following operations:

Sum: $A + B$

Subtraction: $A - B$

Multiplication: $A * B$

Division: A/B or $A \setminus B$

Inverse of A: $\text{inv}(A)$

dot product of the matrix

Ans: -

1. SUM: - $A+B$

2.SUBSTRACTION: - $A - B$

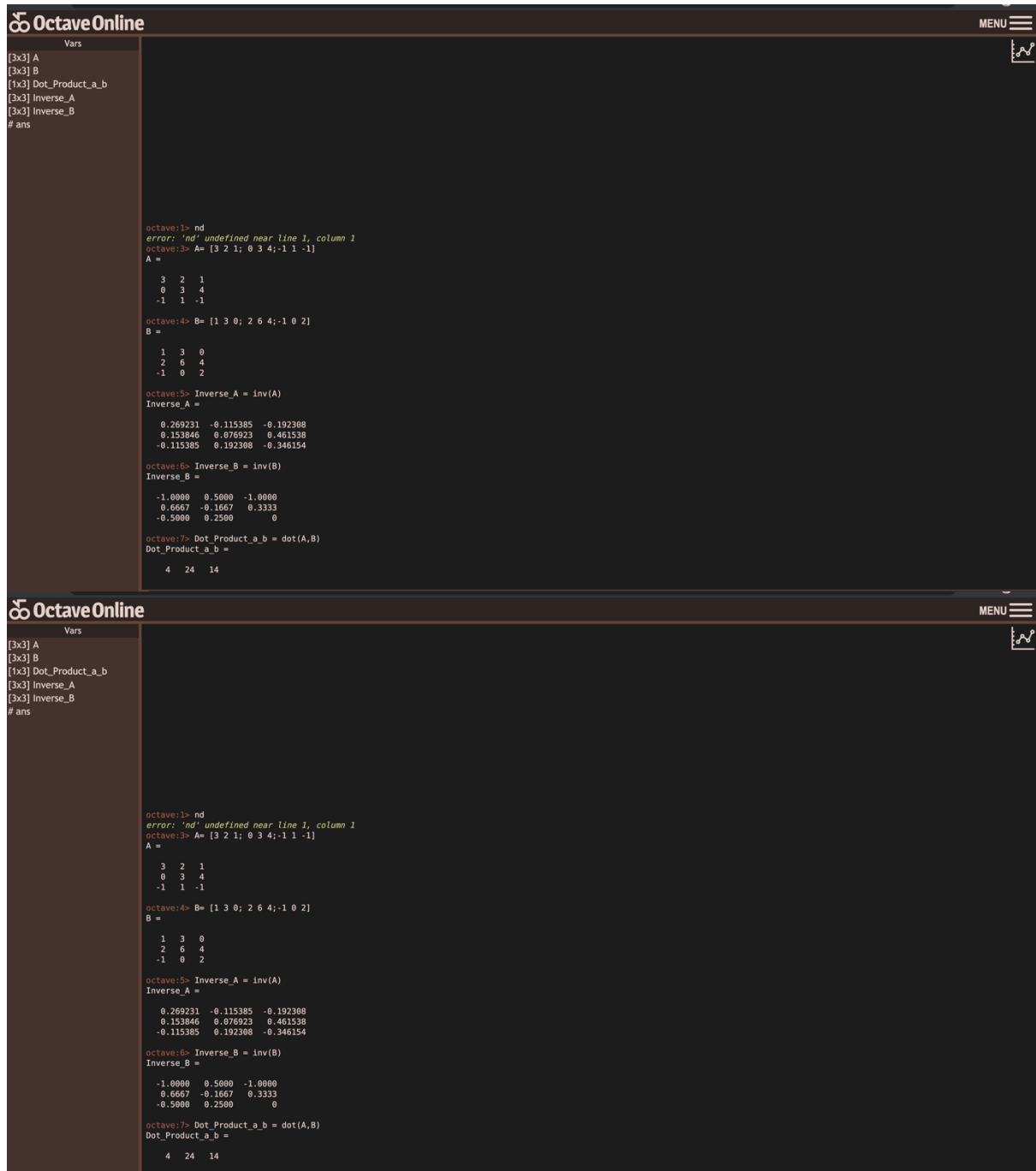
3.MULTIPLICATION: - $A*B$

4.DIVISION: - A/B

5. INVERSE OF A: - $\text{inv}(A)$

6. DOT PRODUCT OF THE MATRIX :- $\text{dot}(A,B)$

SCREEN SHOTS: -



The image displays two screenshots of the OctaveOnline web interface, showing the execution of MATLAB code for matrix operations. The interface includes a 'Vars' panel on the left and a main workspace on the right.

Top Screenshot:

```
OctaveOnline
Vars
[3x3] A
[3x3] B
[1x3] Dot_Product_a_b
[3x3] Inverse_A
[3x3] Inverse_B
# ans

octave:1> nd
error: 'nd' undefined near line 1, column 1
octave:3> A= [3 2 1; 0 3 4;-1 1 -1]
A =
    3    2    1
    0    3    4
   -1    1   -1

octave:4> B= [1 3 0; 2 6 4;-1 0 2]
B =
    1    3    0
    2    6    4
   -1    0    2

octave:5> Inverse_A = inv(A)
Inverse_A =
    0.269231   -0.115385   -0.192308
    0.153846    0.076923    0.461538
   -0.115385    0.192308   -0.346154

octave:6> Inverse_B = inv(B)
Inverse_B =
   -1.0000    0.5000   -1.0000
    0.6667   -0.1667    0.3333
   -0.5000    0.2500     0

octave:7> Dot_Product_a_b = dot(A,B)
Dot_Product_a_b =
    4    24    14
```

Bottom Screenshot:

```
OctaveOnline
Vars
[3x3] A
[3x3] B
[1x3] Dot_Product_a_b
[3x3] Inverse_A
[3x3] Inverse_B
# ans

octave:1> nd
error: 'nd' undefined near line 1, column 1
octave:3> A= [3 2 1; 0 3 4;-1 1 -1]
A =
    3    2    1
    0    3    4
   -1    1   -1

octave:4> B= [1 3 0; 2 6 4;-1 0 2]
B =
    1    3    0
    2    6    4
   -1    0    2

octave:5> Inverse_A = inv(A)
Inverse_A =
    0.269231   -0.115385   -0.192308
    0.153846    0.076923    0.461538
   -0.115385    0.192308   -0.346154

octave:6> Inverse_B = inv(B)
Inverse_B =
   -1.0000    0.5000   -1.0000
    0.6667   -0.1667    0.3333
   -0.5000    0.2500     0

octave:7> Dot_Product_a_b = dot(A,B)
Dot_Product_a_b =
    4    24    14
```

EXERCISE BASED ON EXPERIMENT 3:

NAME: - VARUN VERMA

BRANCH: - EEE

Roll No.: - 02211504919

AIM: - EXPLORING INTERACTIVE INPUTS AND OUTPUTS, MANIPULATIONS OF CHARACTER STRINGS, ADVANCED PLOTTING FEATURES AND ITERATIONS

EXERCISE: -

For loop:

Q1: - A program to print the number from 1 to 10

Ans: -

For loop

For i= 1:10

disp(i)

end

screenshot:

```
octave:2> x= [1:10]
x =
     1     2     3     4     5     6     7     8     9    10

octave:3> x =1
x = 1
octave:4> for i=1:10
> > disp(i)
> > end
1
2
3
4
5
6
7
8
9
10
```

Q 2: - A program to calculate the sum of squares of number from 1 to n.

Ans: -

n= 10

s=0;

for i= 1:n

s=s+i^2;

disp(s)

end

screen shot:

```
octave:5> n = 10
n = 10
octave:6> s=0
s = 0
octave:7> for i=1:10
> > s= s+ i^2
> > end
s = 1
s = 5
s = 14
s = 30
s = 55
s = 91
s = 140
s = 204
s = 285
s = 385
```

Q3: - Calculate the factorial of any number n.

Ans: -

n = 10

f = 1;

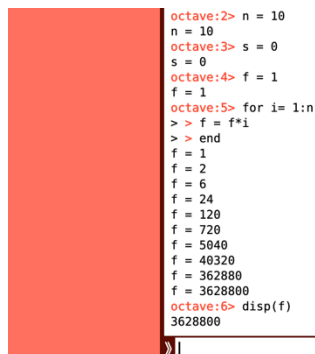
for i= 1:n

f =f*i

end

disp(f)

Screen Shot: -



```
octave:2> n = 10
n = 10
octave:3> s = 0
s = 0
octave:4> f = 1
f = 1
octave:5> for i= 1:n
> > f = f*i
> > end
f = 1
f = 2
f = 6
f = 24
f = 120
f = 720
f = 5040
f = 40320
f = 362880
f = 3628800
octave:6> disp(f)
3628800
```

Q4: -

Ans: -

X = [1 2 3]

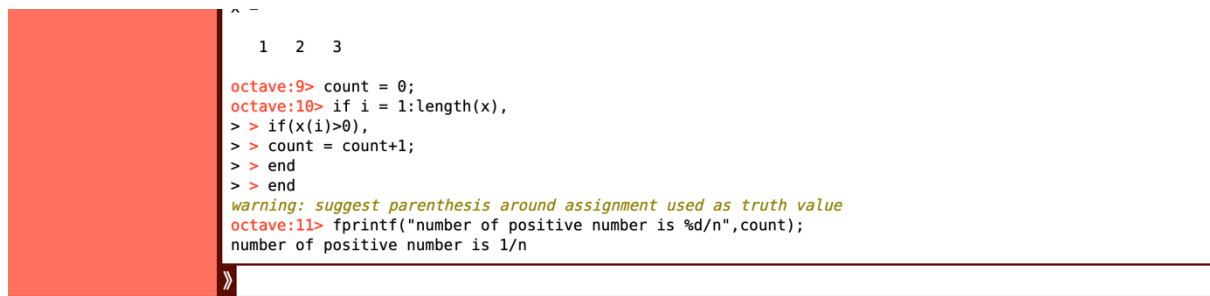
Count =0;

If i= 1:length(x)

If(x(i)>0),

```
Count = count+1;  
End  
End  
fprintf("number of positive number  
is%d/n",count);
```

Screen Shot: -



```
1 2 3  
octave:9> count = 0;  
octave:10> if i = 1:length(x),  
> > if(x(i)>0),  
> > count = count+1;  
> > end  
> > end  
warning: suggest parenthesis around assignment used as truth value  
octave:11> fprintf("number of positive number is %d/n",count);  
number of positive number is 1/n
```

WHILE LOOP:

Q2: - A program to display powers of Break,
return

1. Program to display prime no. from 1 to 50.

The program exit from the inner loop using
break, if the number is not a prime.

Ans: -

X = 1

While x<=5

X = 2 *x

```

End
Break, return
For n= 2:50
For m= 2:50
If(~mod(n,m))
Break
End
End
If(m>(n/m))
fprintf("%d is prime/n",n);
end
end

```

screen shot

```

octave:18> Break, return
error: 'Break' undefined near line 1, column 1
octave:19> for n = 2:50
> > for m = 2:50
> > if(~mod(n,m))
> > break
> > nd
> > end
> > end
> > if(m>(n/m))
> > fprintf("%d is prime/n",n)
> > end
> > end
2 is prime/n3 is prime/n5 is prime/n7 is prime/n11 is prime/n13 is prime/n17 is prime/n19 is prime/n23 is prime/n29 is prime/n31 is prime/n37 is
prime/n41 is prime/n43 is prime/n47 is prime/n

```

Q1: - A program to print the number from 1 to 10

Ans:

```

I = 1;
While i<10

```

Disp(i)

I=i+10

End

Screen shot

```
octave:12> i=1;
octave:13> while i<=10
> > disp(i)
> > i=i+1
> > end
1
i = 2
2
i = 3
3
i = 4
4
i = 5
5
i = 6
6
i = 7
7
i = 8
8
i = 9
9
i = 10
10
i = 11
. . . . .
```

EXPERIMENT: NO: 4
NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

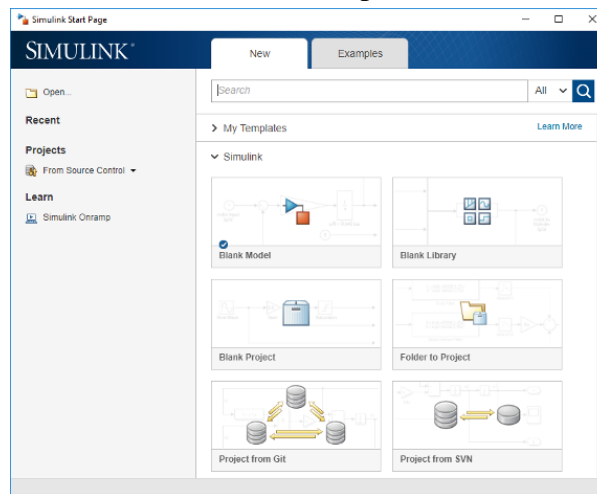
AIM: Introduction to MATLAB Simulink, designing different circuits in Simulink and observing outputs.

MATLAB Simulink Basics

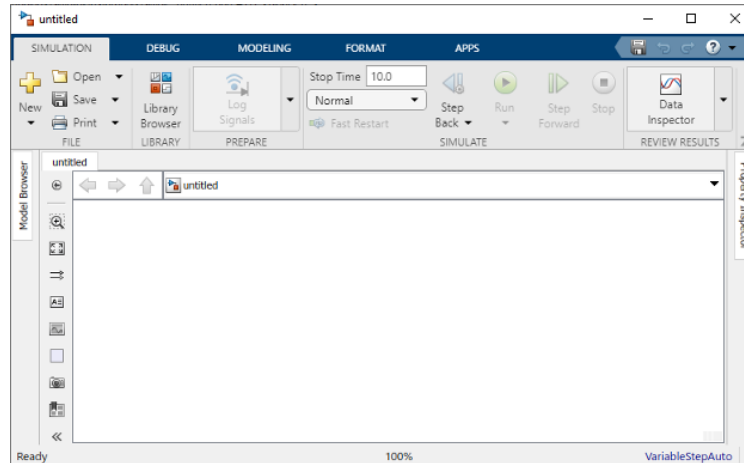
Simulink is a MATLAB-based graphical programming environment for modeling, simulating and analyzing multi-domain dynamical systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in automatic control and digital signal processing for multi-domain simulation and model-based design.

Open New Model: The Simulink Editor can be used to build models.

1. Start MATLAB®. From the MATLAB tool strip, click the **Simulink** button .



2. Click the **Blank Model** template to open the Simulink Editor.




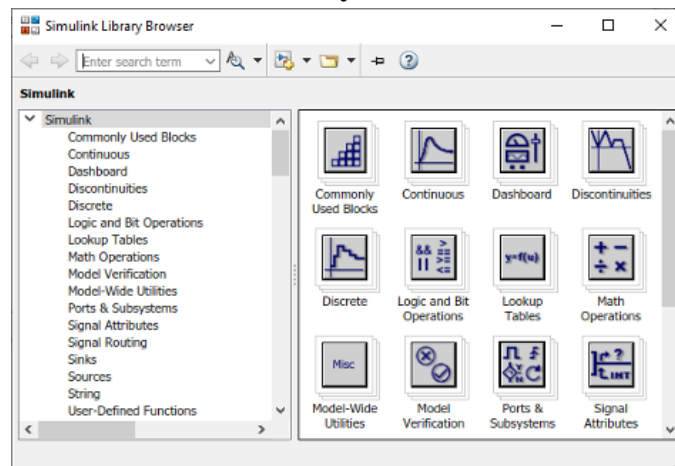
- From the **Simulation** tab, select **Save > Save as**. In the **File name** text box, enter a name for your model and click **Save**. The model is saved with the file extension **.slx**.


Open Simulink Library Browser

Simulink provides a set of block libraries, organized by functionality in the Library Browser. The following libraries are common to most workflows:

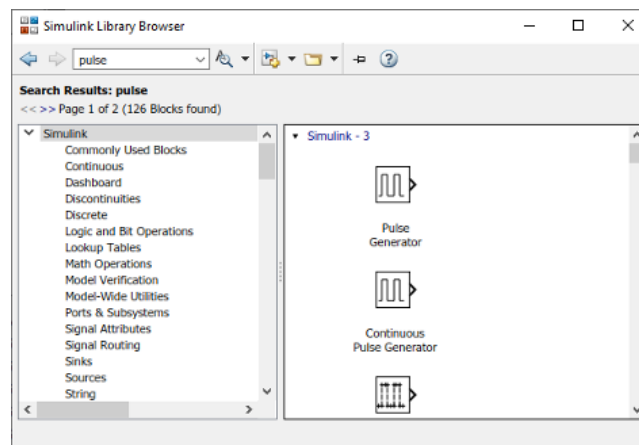
- Continuous — Blocks for systems with continuous state
- Discrete — Blocks for systems with discrete states
- Math Operations — Blocks that implement algebraic and logical equations
- Sinks — Blocks that store and show the signals that connect to them
- Sources — Blocks that generate the signal values that drive the model

- From the Simulation tab, click the **Library Browser** button .



- Set the Library Browser to stay on top of the other desktop windows. On the Simulink Library Browser toolbar, select the **Stay on top** button . To browse through the block libraries, select a category and then a functional area in the left pane. To search all of the available block libraries, enter a search term.

For example, find the Pulse Generator block. In the search box on the browser toolbar, enter pulse, and then press Enter. Simulink searches the libraries for blocks with pulse in their name or description and then displays the blocks.

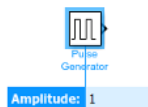


Get detailed information about a block. Right-click the Pulse Generator block, and then select **Help for the Pulse Generator block**. The Help browser opens with the reference page for the block. Blocks typically have several parameters. You can access all block parameters by double-clicking the block.

Add Blocks to a Model

To start building the model, browse the library and add the blocks.

From the Sources library, drag the Pulse Generator block to the Simulink Editor. A copy of the Pulse Generator block appears in your model with a text box for the value of the **Amplitude** parameter. Enter 1. Other blocks to your model can be added using the same approach. Parameter values are held throughout the simulation.



Connect Blocks

Connect the blocks by creating lines between output ports and input ports.

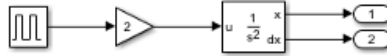
1. Click the output port on the right side of the Pulse Generator block. The output port and all input ports suitable for a connection are highlighted.



2. Click the input port of the Gain block. Simulink connects the blocks with a line and an arrow indicating the direction of signal flow.

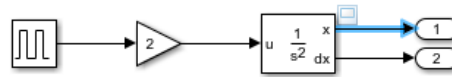


3. Connect the output port of the Gain block to the input port on the Integrator, Second-Order block.
4. Connect the two outputs of the Integrator, Second-Order block to the two Output blocks.
5. Save your model. In the **Simulation** tab, click the **Save** button.



Add Signal Viewer

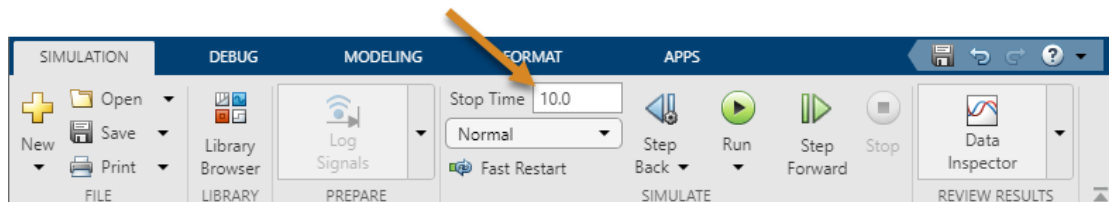
To view simulation results, connect the first output to a Signal Viewer. Click the signal. In the **Simulation** tab under Prepare, click **Add Viewer**. Select **Scope**. A viewer icon appears on the signal and a scope window opens. You can open the scope at any time by double-clicking on it.



Run Simulation

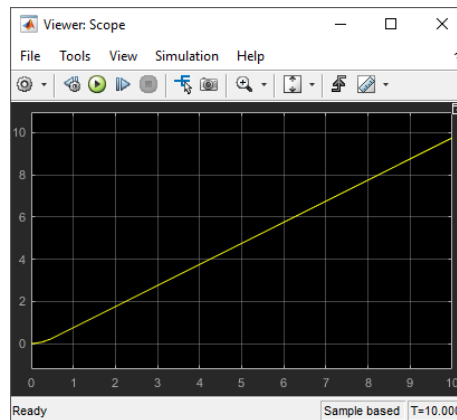
After you define the configuration parameters, you are ready to simulate your model.

1. In the **Simulation** tab, set the simulation stop time by changing the value in the toolbar.



The default stop time of 10.0 is appropriate for this model. This time value has no unit. The time unit in Simulink depends on how the equations are constructed. Other models could have time units in milliseconds or years.

2. To run the simulation, click the **Run** button . The simulation runs and produces the output in the viewer.



Annotate Signals

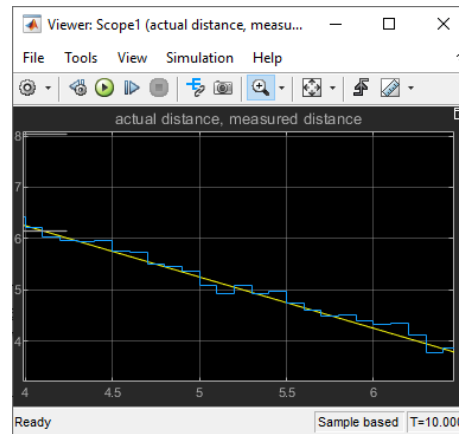
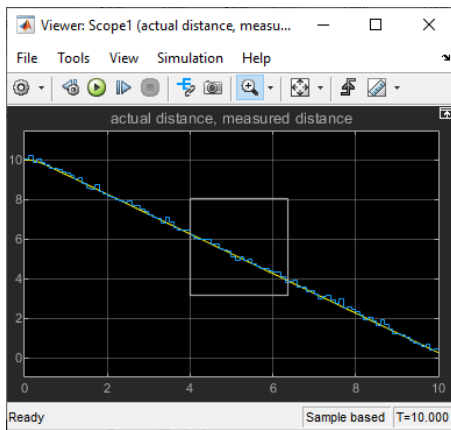
Add signal names to the model.

1. Double-click the signal and type the signal name.
2. To finish, click away from the text box.
3. Repeat these steps to add the names as shown.



Zoom

To zoom into the graph, click the **Zoom** button . Left-click and drag a window around the region you want to see more closely. You can repeatedly zoom in to observe the details.



EXPERIMENT: NO: 5

NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

AIM: To design different circuits in Simulink and observing outputs.

Module: Building simple circuits and verifying circuit analysis techniques through scope.

A. Nodal Analysis

B. Mesh Analysis

C. Bridge rectifier using diodes

a. NODAL ANALYSIS

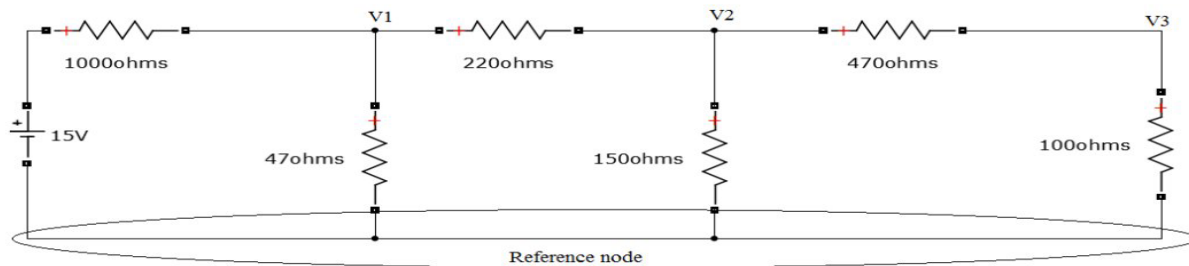
A direct current circuit is mainly composed of resistors, current sources and voltage sources. Simulink helps simulate the circuit to compute nodal voltages. We can model the circuit in figure to determine the nodal voltages at node 1 and node 2. Following are the modelling steps:

1. Model the resistors through the Simulink block (SimPowerSystems--→Elements-- →Series RLC Branch. (Set the resistance to the given value, set L=0 and C=0)
2. Model the current source through the Simulink block (SimPowerSystems--→Electrical Sources--→Controlled Current Source)
3. Model the constant for the current source through (Simulink--→Sources--→Constant)
4. Model the voltage measurement through (SimPowerSystems--→Measurements-- →Voltage Measurement)
5. Block name display can be invoked through (Simulink--→Sinks--→Display)
6. Nodes can be assigned by invoking the neutral through (SimPowerSystems--→Elements- --→Neutral. The node number can be changed by double click on the neutral.
7. Continuous power GUI must be embedded in the circuit. (SimPowerSystems--→powerGUI

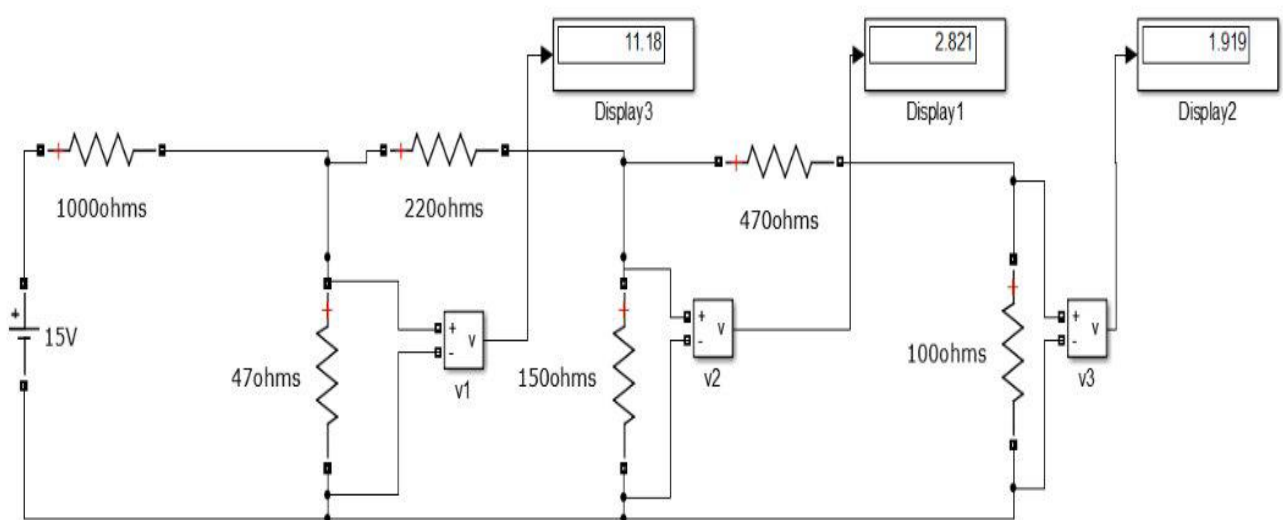
1. NODAL ANALYSIS

In electric circuit analysis, nodal analysis, node-voltage analysis, or the branch current method is a method of determining the voltage (potential difference) between "nodes" (points where elements or branches connect) in an electrical circuit in terms of the branch currents.

CIRCUIT DIAGRAM:



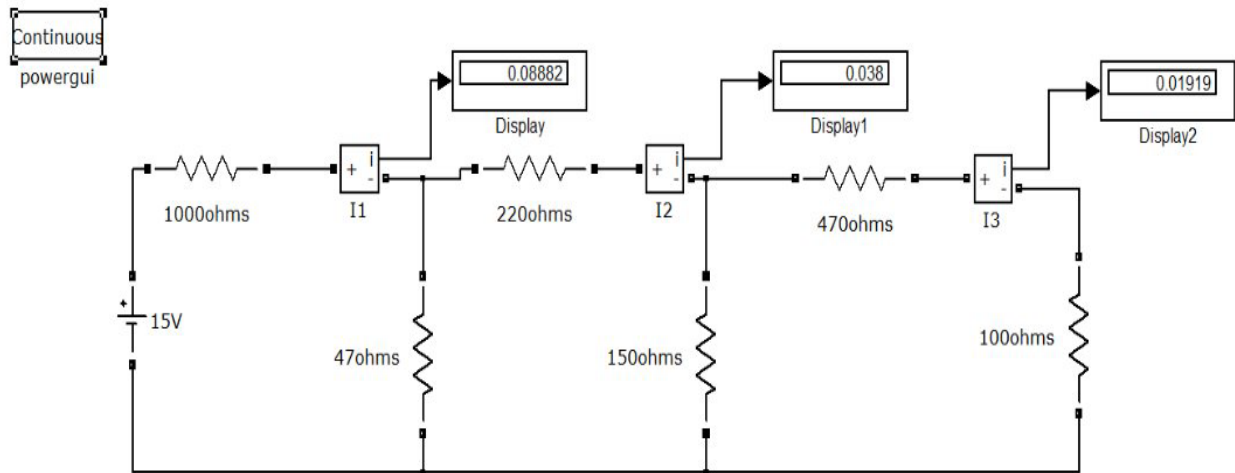
SIMULATION DIAGRAM:



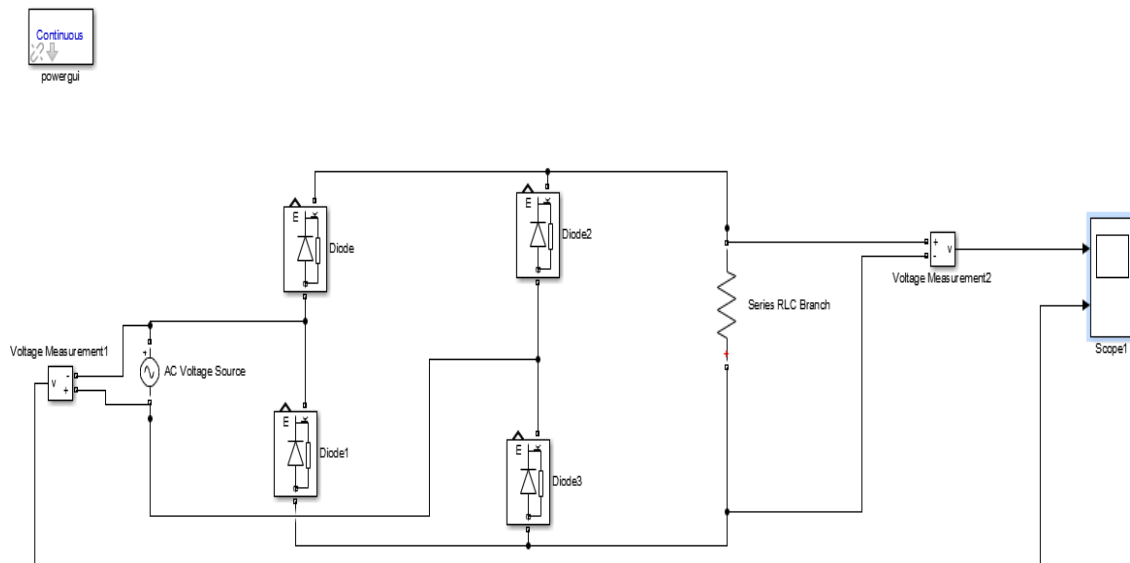
2. MESH ANALYSIS

Multi-source DC circuits may be analyzed using a mesh current technique. The process involves identifying a minimum number of small loops such that every component exists in at least one loop. KVL is then applied to each loop. The loop currents are referred to as mesh currents as each current interlocks or meshes with the surrounding loop currents. As a result there will be a set of simultaneous equations created, an unknown mesh current for each loop. Once the mesh currents are determined, various branch currents and component voltages may be derived.

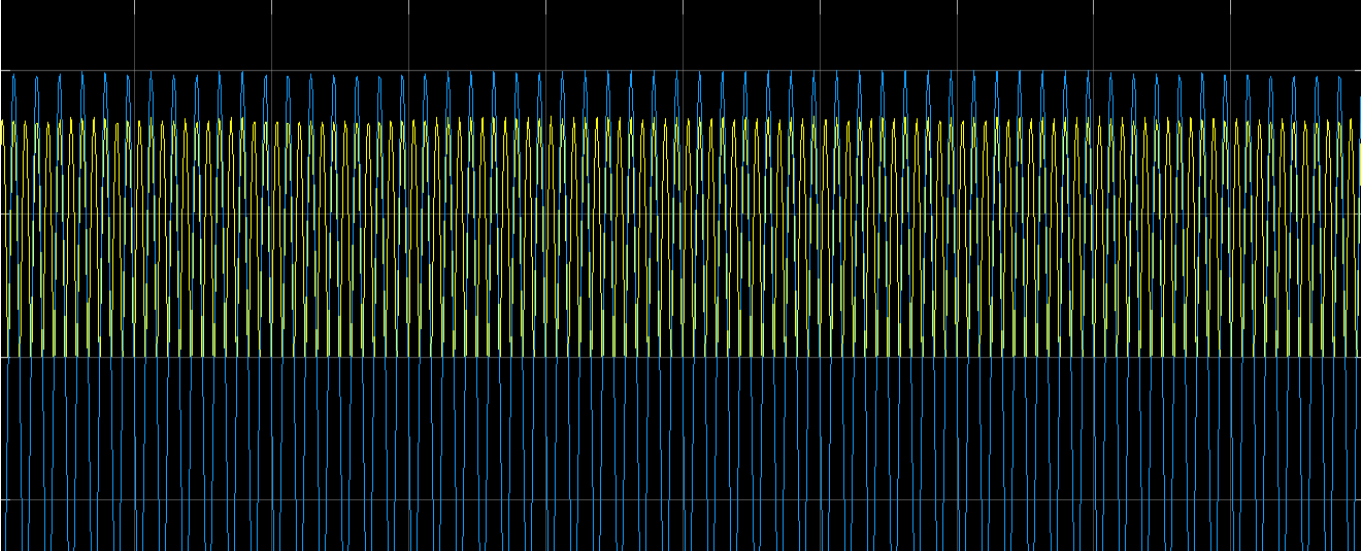
SIMULATION DIAGRAM:



3. BRIDGE RECTIFIER USING DIODES SIMULATION DIAGRAM:



OUTPUT RESPONSE



EXPERIMENT: NO: 6

NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

AIM: Creating and working with Functions and Directories in MATLAB

In MATLAB, there are several commands to make file-navigation easier. There are several commands which help to know which directory you are currently in, to change the current directory in MATLAB path, to modify the MATLAB path, to create new directories, to view contents of the current directory and, to copy and move files from one directory to another.

A function file is also an M-file, like a script file, except the variables in a function are all local. Function files are like programs or subroutines. A function file begins with a function definition line, which has a well-defined list of inputs and outputs. Without this line, the file becomes a script file. The first word in the function definition line, **function**, must be typed in lowercase. The function name must be same as the file name (without the *.m* extension). For example, if the name of the function is **projectile**, it must be written and saved in a file with the name **projectile.m**. A function definition line may look slightly different, depending on whether there is no output, a single output or multiple outputs. Multiple output variables must be enclosed within square brackets []. A single output variable is not required to be enclosed within [].

MATLAB also allows several functions to be written in a single file. The first function can access all its sub-functions, and the sub-functions written in the same file can also access each other. However, functions outside this file cannot access these sub-functions. Nested functions are sub-functions written inside a main function, just like sub-functions but with some important distinctions. Each nested function must be terminated by an **end** statement. Moreover, all nested functions share the workspace of functions in which they are nested.

MATLAB includes automatic report generator called **publisher**, which is accessible both from menu as well as from the command line. The **publisher** publishes a script in several formats, including HTML, MS Word, PowerPoint, XML and LATEX.

The M Files

MATLAB allows writing two kinds of program files:

- **Scripts** – script files are program files with **.m extension**. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.
- **Functions** – functions files are also program files with **.m extension**. Functions can accept inputs and return outputs. Internal variables are local to the function.

We can use the MATLAB editor or any other text editor to create your **.m**files. In this section, we will discuss the script files. A script file contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

Creating and Running Script File

To create scripts files, you need to use a text editor. You can open the MATLAB editor in two ways:

- Using the command prompt
- Using the IDE

If you are using the command prompt, type **edit** in the command prompt. This will open the editor. You can directly type **edit** and then the filename (with .m extension)

```
edit
Or
edit <filename>
```

The above command will create the file in default MATLAB directory. If we want to store all Program files in a specific folder, and then we will have to provide the entire path.

We can create a folder named progs. Type the following commands at the command prompt

(>>) –

```
mkdir progs      % create directory progs under default directory
chdir progs      % changing the current directory to progs
edit prog1.m     % creating an m file named prog1.m
```

Alternatively we can choose NEW -> Script. This also opens the editor and creates a file named Untitled. We can name and save the file after typing the code.

Type the following code in the editor –

```
NoOfStudents = 6000;
TeachingStaff = 150;
NonTeachingStaff = 20;
```

```
Total = NoOfStudents + TeachingStaff ...  
      + NonTeachingStaff;  
disp(Total);
```

After creating and saving the file, we can run it in two ways –

- Clicking the **Run** button on the editor window or
- Just typing the filename (without extension) in the command prompt: >> prog1

The command window prompt displays the result –

```
6170
```

Example

Create a script file, and type the following code –

```
a = 5; b = 7;  
c = a + b  
d = c + sin(b)  
e = 5 * d  
f = exp(-d)
```

When the above code is compiled and executed, it produces the following result –

```
c =    12  
d =   12.657  
e =   63.285  
f =    3.1852e-06
```

EXPERIMENT: NO: 7

NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

AIM: To create an interactive function that calculates the RMS value, arithmetic mean, geometric mean and harmonic mean from the user-entered data.

MATLAB CODE:

1. % using for loop

```
function[AMean,GMean,HMean,RMSVAL]=means()
n = input ('enter the limit:');
for i=1:1:n
x(i) = input('enter the values of the array:');
end
Amean = sum(x)/n;
sprintf('The Arithmetic mean is: %d', mean(x))
sprintf ('The Arithmetic mean using inbuilt function is :%d', mean(x))
Gmean = (prod(x))^(1/n);
sprintf ('The geometric mean is : %d', Gmean)
sprintf ('The Geometric mean using inbuilt function is :%d', geomean(x))
Hmean = n/sum(1./x);
sprintf('The Harmonic mean is: % d', Hmean)
sprintf ('The Harmonic mean using inbuilt function is :%d', harmmean(x))
RMSVAL =sqrt (sum(x.^2/n));
sprintf ('The RMS Value is :%d', RMSVAL)
sprintf('the RMS value using inbuilt function is :%d', rms(x))
end
```

OUTPUT:

```
enter the limit:5
enter the values of the array:1
enter the values of the array:2
enter the values of the array:3
enter the values of the array:4
enter the values of the array:5
```

ans = The Arithmetic mean is: 3

ans = The Arithmetic mean using inbuilt function is :3

ans = The geometric mean is : 2.605171e+00

ans = The Geometric mean using inbuilt function is :2.605171e+00

ans = The Harmonic mean is: 2.189781e+00

ans = The Harmonic mean using inbuilt function is :2.189781e+00

ans = The RMS Value is :3.316625e+00

ans = the RMS value using inbuilt function is :3.316625e+00

2. % using switch case

```
function[AMean,GMean,HMean,RMSVAL]=means()
n = input ('enter the limit:');
for i=1:1:n
x(i) = input('enter the values of the array:');
end
sprintf('AMean=1,GMean=2,HMean=3,RMSVAL=4')
c=input('Enter choice');
switch(c)
    case 1
        AMean=sum(x)/n;
        sprintf('The Arithmetic mean is: %d', AMean)
    case 2
        GMean = (prod(x))^(1/n);
        sprintf('The geometric mean is: %d', GMean)
    case 3
        HMean = n/sum(1./x);
        sprintf('The harmonic mean is: %d', HMean)
    case 4
        RMSVAL =sqrt (sum(x.^2/n));
        sprintf('The RMS value is: %d', RMSVAL)
    otherwise
        error('wrong choice')
end
```

3. % using while loop

```
function[ Amean, Gmean, Hmean, RMSVAL] = means()
x = input('Enter the first value :');
choice = input('do you wish to continue?(y/n):','s');
```

```

while choice == 'y'
y = input('enter value :');
x = [x , y];
choice = input('do you want to enter more values?(y/n):','s');
end
AMean = sum(x)/length(x);
sprintf('The arithmetic mean is :%d', AMean)
sprintf('The arithmetic mean using inbuilt function is:%d', mean(x))
GMean = (prod(x))^(1/length(x));
sprintf('The Geometric mean is :%',GMean)
sprintf('The Geometric Mean using inbuilt function is :%d', geomean (x))
HMean = length(x)/sum(1./x);
sprintf('The harmonic mean is :%d', HMean)
sprintf('The Harmonic Mean using inbuilt function is :%d', harmmean(x))
RMSVAL = sqrt(sum(x.^2)/length(x));
sprintf('The RMS value is :%d', RMSVAL)
sprintf('The RMS value using inbuilt function is :%d', rms(x))
end

```

EXPERIMENT: NO: 8

NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

Aim: To create an interactive function that allows the user to choose between the following

- Display whether the user-entered number is even or odd
- Solve and display the result of user-entered quadratic equation.

MATLAB CODE:

1. %ODD-EVEN FUNCTION

```
function []=oddeven1()  
a=input('Enter a Number');  
if(rem(a,2)==0)  
    sprintf('The Number is EVEN')  
else  
    disp('The Number is ODD')  
end
```

OUTPUT:

Enter a Number 4

ans = The Number is EVEN

2. %SOLUTION OF A QUADRATIC EQUATION

```
function []=quadratic()  
disp('Quadratic equation is:ax^2+bx+c')  
p=input('Enter the value of a:');  
q=input('Enter the value of b:');  
r=input('Enter the value of c:');  
d=q^2-4*p*r;  
r1=(-q+sqrt(d))/2*p;  
r2=(-q-sqrt(d))/2*p;  
if(d>=0)  
    sprintf('Root 1 is %f',r1)  
    sprintf('Root 2 is %f',r2)
```

```

else
    sprintf('root 1 is %f+j %f',real(r1),imag(r1))
    sprintf('root 2 is %f+j %f',real(r2),imag(r2))
end

```

OUTPUT:

Quadratic equation is: ax^2+bx+c

Enter the value of a: 4

Enter the value of b: 2

Enter the value of c: 5

ans = root 1 is -4.000000+j 17.435596

ans = root 2 is -4.000000+j -17.435596

3. %TO CREATE MENU DRIVEN PROGRAM

```

function []=menu_choice()
l==menu('Mark the choice','To find odd/even number','To solve
quadratic equation')
switch l
    case 1
        a=input('Enter a number');
        if(rem(a,2)==0)
            sprintf('Even no')
        else
            disp('The number is odd')
        end
    case 2
        disp('Quadratic equation is:ax^2+bx+c')
        p=input('Enter the value of a:');
        q=input('Enter the value of b:');
        r=input('Enter the value of c:');
        d=q^2-4*p*r;
        r1=(-q+sqrt(d))/2*p;
        r2=(-q-sqrt(d))/2*p;
        if(d>=0)
            sprintf('Root 1 is %f',r1)

```



```
        sprintf('Root 2 is %f',r2)
else
    sprintf('root 1 is %f+j %f',real(r1),imag(r1))
    sprintf('root 2 is %f+j %f',real(r2),imag(r2))
end
otherwise
disp('wrong choice')
end
```

EXPERIMENT: NO: 9

NAME: - VARUNVERMA

BRANCH: - EEE

ROLL No.: -02211504919

Aim: Fundamental image processing using MATLAB

Module 1: Basic operations

```
%Start with a truecolor image  
img=imread('onion.png');  
imshow('onion.png')
```



```
%Convert to binary/Black and White  
img_bin=im2bw(img);  
imshow(img_bin)
```



```
%Convert to binary/Black and White  
%with custom threshold (default is 0.5)  
img_bin2=im2bw(img, 0.3);  
imshow(img_bin2)
```



```
%Convert to grayscale
img_gray=rgb2gray(img);
imshow(img_gray)
```



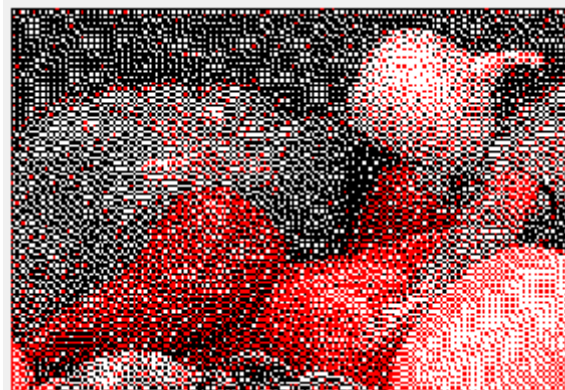
```
%reduce to indexed color
[img_indexed, map]=rgb2ind(img,8);
imshow(img_indexed, map)
```



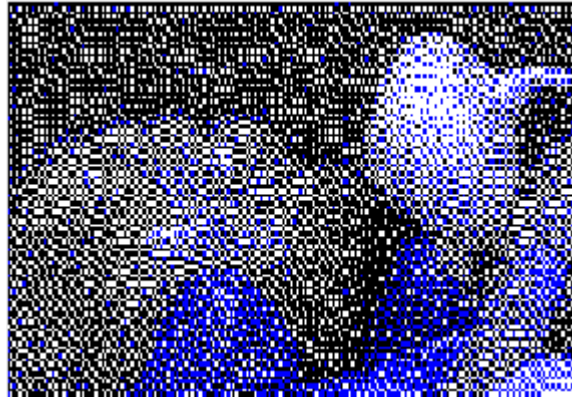
```
%reduce to indexed color
%without dithering
[img_nodither, map2]=rgb2ind(img,8, 'nodither');
imshow(img_nodither, map2)
```



```
%reduce to indexed color
%using a custom colormap
map_custom=[
    1 1 1; %white
    1 0 0; %red
    0 0 0]; %black
img_custommap=dither(img, map_custom);
imshow(img_custommap, map_custom)
```

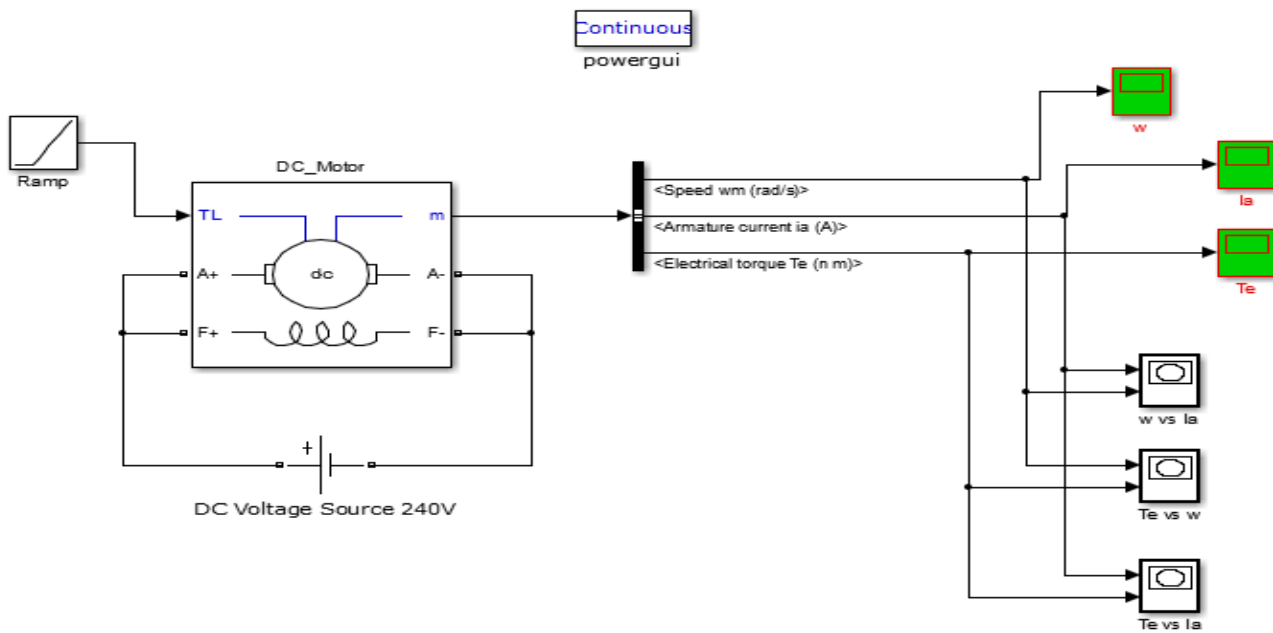


```
%Of course you can swap colormaps...
map_custom2=[
    1 1 1; %white
    0 0 1; %blue
    0 0 0]; %black
imshow(img_custommap, map_custom2)
```

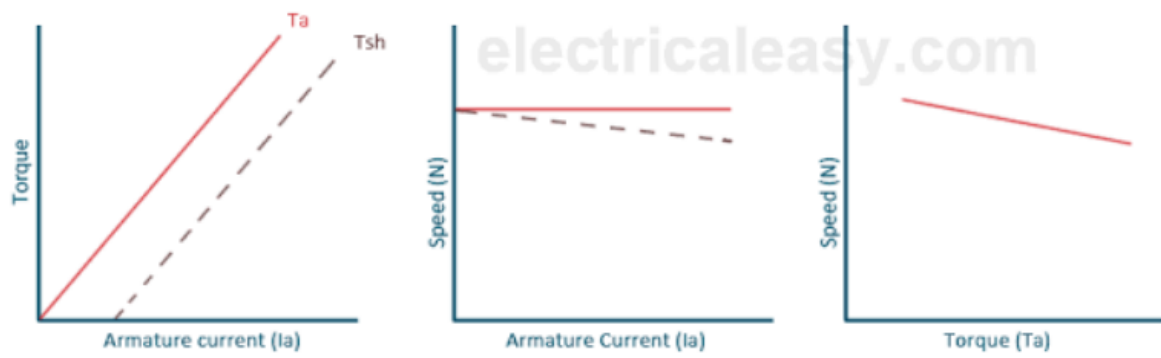


EXPERIMENT: NO: 10

Aim: Plotting and verifying the Speed-Torque characteristics of DC shunt motor.



Simulink Model for DC shunt motor



Characteristics of DC shunt motor