Part I

Orientation

Workshop 1: Introduction to Laboratory Instruments

Objective: To develop the ability to use the common components and instruments in the laboratory..

Outcome: After successful completion of this session, the student will be able to

1. Identify resistors, capacitors and read their values
2. Measure values of resistors using the multi-meter
3. Use the oscilloscope to display and measure the characteristics of electrical signals

Equipment Required:

1. DC power supply
2. Digital multi-meter
3. Analog multi-meter
4. Protoboard
5. Oscilloscope
6. Signal generator
7. Spectrum analyzer

Components Required:

1. Resistors: 330 Ω, 10 kΩ, 100 kΩ, 1 MΩ
2. Capacitors: 4.7 μF, 220 μF, 10 pF, 100 nF
3. Diodes: 1N4001, Zener diode, Red LED
4. Transistor: 2N2222

1.1 Identifying Resistors and Capacitors

Task 1. *Read the values of the provided resistors using the colour code, including tolerances. Record the values in Table 1.1.*

Label	1st band	2nd band	3rd band	4th band	Value	Tolerance
R_1	1	0	10^1	±5%	100 Ω	100 Ω ± 5%
R_2	1	0	10^4	± 5%	100 kΩ	100 kΩ ± 5%
R_3	1	0	10^5	± 5%	1 MΩ	1 MΩ ± 5%
R_4	3	3	10^1	± 10 %	330 Ω	330 ± 10 %

Table 1.1: Resistor details

Task 2. *Read the values of the provided capacitors. Identify their types (ceramic or electrolytic) and record in Table 1.2. You may use non-standard abbreviations "Ce" for ceramic and "El" for electrolytic when recording the type.*

Label	Printed value/identifier	Capacitance	Type
C_1	220MF	220MF	El
C_2	4.7	4.7MF	El
C_3	10	10 pF	Ce
C_4			

Table 1.2: Capacitor details

Label	Analog MM reading	Analog MM scale	Digital MM reading	Digital MM scale
R_1	110 Ω	10 Ω	99.559 Ω	100 Ω
R_2	100 kΩ	1 kΩ	0.10224 MΩ	1 MΩ
R_3	900 kΩ	100k Ω	1.0183 MΩ	10 MΩ
R_4	290 Ω	10 Ω	0.32915 kΩ	1 kΩ

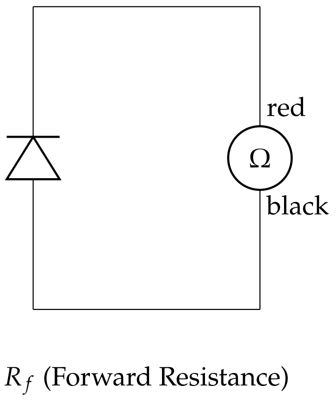
Table 1.3: Resistance values measured using the multi-meter

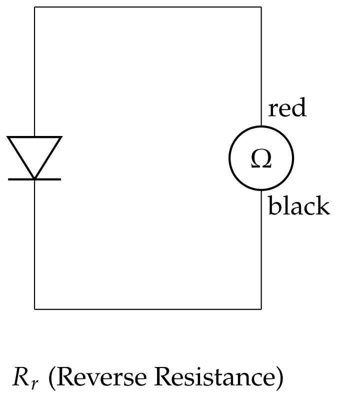
1.2 Measuring Resistance Using the Multi-meter

Task 3. *Measure the exact resistance values of the provided resistors using the multi-meter ohm scale. Record your observations in Table 1.3.*
Hint: *Fix the resistors on the protoboard to hold them still when measuring.*

The resistance of a forward biased diode (R_f) is low and the resistance of a reversed biased diode (R_r) is high. By observing R_f and R_r, it is possible to test a diode and determine whether it is working or not.

Task 4. *Identify the diode pins and fix the diode on the protoboard. Use the analog multi-meter as an ohmmeter (refer Figure 1.1), and measure and record R_f and R_r. Is the diode working properly?*





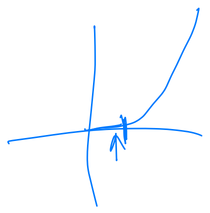


Figure 1.1: Measuring forward and reverse resistances of a diode using the analog multi-meter

~~2.50~~
250kΩ
3.5kΩ
55 Ω
7.5 Ω

A digital multi-meter in the *Diode Mode* can be used to check whether a diode is working. In contrast to the resistance, in this mode, the multi-meter will display the bias voltage drop across the diode.

Task 5. *Use the digital multi-meter in the Diode Mode to check the provided diode. Draw the setup under each configuration (forward biased and reverse biased, similar to that of Figure 1.1) and record the displayed voltage drop.*

1.3 Measuring DC Bias Voltages of a BJT

Task 6. *Identify the Emitter (E), Base (B) and Collector (C) pins of the BJT and draw the pin configuration.*

Forward Biased		Reversed Bias	
R_{BE}	R_{BC}	R_{BE}	R_{BC}

Table 1.4: Junction resistances of a BJT

Task 7. *Fix the transistor on the protoboard and measure the resistances of the B-E (R_{BE}) and the B-C (R_{BC}) junctions separately, in forward biased and reversed biased configurations. Record your observations in Table 1.4.*

Task 8. *Construct the circuit given in Figure 1.2 on the protoboard. The transistor is 2N2222. Apply +5 V to V_{cc} while the Emitter terminal is grounded.*

a) Apply 0 V to V_{in} and measure V_{CE} using the multi-meter. Is the LED ON or OFF?

b) Apply +5 V to V_{in} and measure V_{CE} using the multi-meter. Is the LED ON or OFF?

c) Measure and record I_B and I_C using the analog and the digital multi-meter respectively.

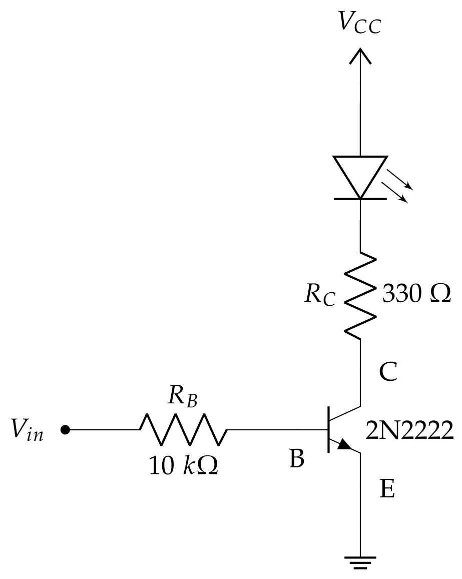


Figure 1.2: Transistor as a switch

1.4 Observing Signals Using the Oscilloscope

Task 9. *Connect the signal generator across a 10 $k\Omega$ resistor as in Figure 1.3. Apply a sinusoidal signal with a peak-to-peak voltage (V_{pp}) of 1 V and frequency 1 kHz. Connect the oscilloscope probes of channel 1 and observe the signal. Plot V_{in} vs t.*

Task 10. *Measure and record the peak-to-peak amplitude of the observed signal using the* ***Cursors*** *of the oscilloscope.*

Task 11. *Change the input (V_{in}) to a sinusoidal signal with frequency 150 kHz and plot V_{in} vs t. Measure the period of the signal using the* ***Cursors****.*

Task 12. *Change the input (V_{in}) to a sinusoidal signal with a 100 mV peak-to-peak amplitude, a frequency of 150 kHz, and a 1 V DC offset. Observe the signal using the oscilloscope with* ***AC coupling*** *and* ***DC coupling****, separately. Plot the two waveforms on the same plot. You may use different amplitude scales to fit the waveform within the observable area.*

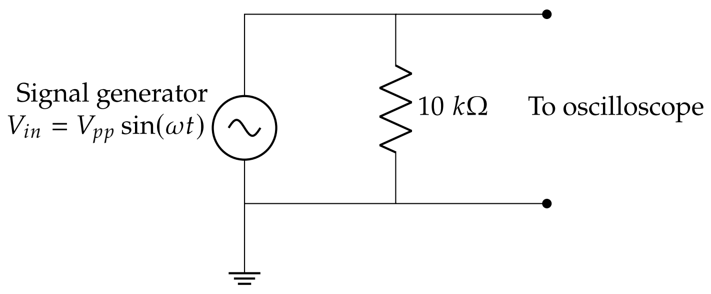


Figure 1.3: Connecting signal generator to a load resistor

Task 13. *Change the input (V_{in}) to a square signal with 100 mV peak-to-peak amplitude, a frequency of 150 kHz, a 0 V DC offset and a 30% duty cycle. Measure the Root Mean Square voltage of the signal using the* ***Measure*** *function of the oscilloscope.*

Task 14. *RMS voltage of a square waveform can be calculated as*

$$V_{RMS} = V_{pp}\sqrt{D}$$

where V_{pp} is the peak-to-peak amplitude and D is the duty cycle expressed as a fraction. Calculate the RMS voltage for the square wave using the above relationship and comment on the agreement with the direct measurement from the oscilloscope.

♣ The End ♣

Workshop 2: Introduction to MATLAB and Simulink

Objective: To start using MATLAB as a computational and simulation tool for Engineering applications

Outcome: After successfully completing this session, the student would be able to

1. Use MATLAB for Matrix operations
2. Manipulate control structures in MATLAB programming efficiently
3. Utilize in-built functions in MATLAB to solve problems
4. Construct user-defined functions appropriately
5. Represent data in a graphical form using MATLAB
6. Simulate simple systems using Simulink
7. Understand the wide variety of applications of MATLAB as a tool

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online

2.1 Introduction to MATLAB

MATLAB is a tool for numerical computation and visualization. It can be used for tasks such as math computation, algorithm development, modeling, simulation, prototyping, data analysis, exploration, visualization, scientific and engineering graphics, and application development (Graphical User Interface building). In the laboratory practice module you will be using MATLAB mainly for the telecommunication laboratories. This orientation session will guide you through the basics of MATLAB.

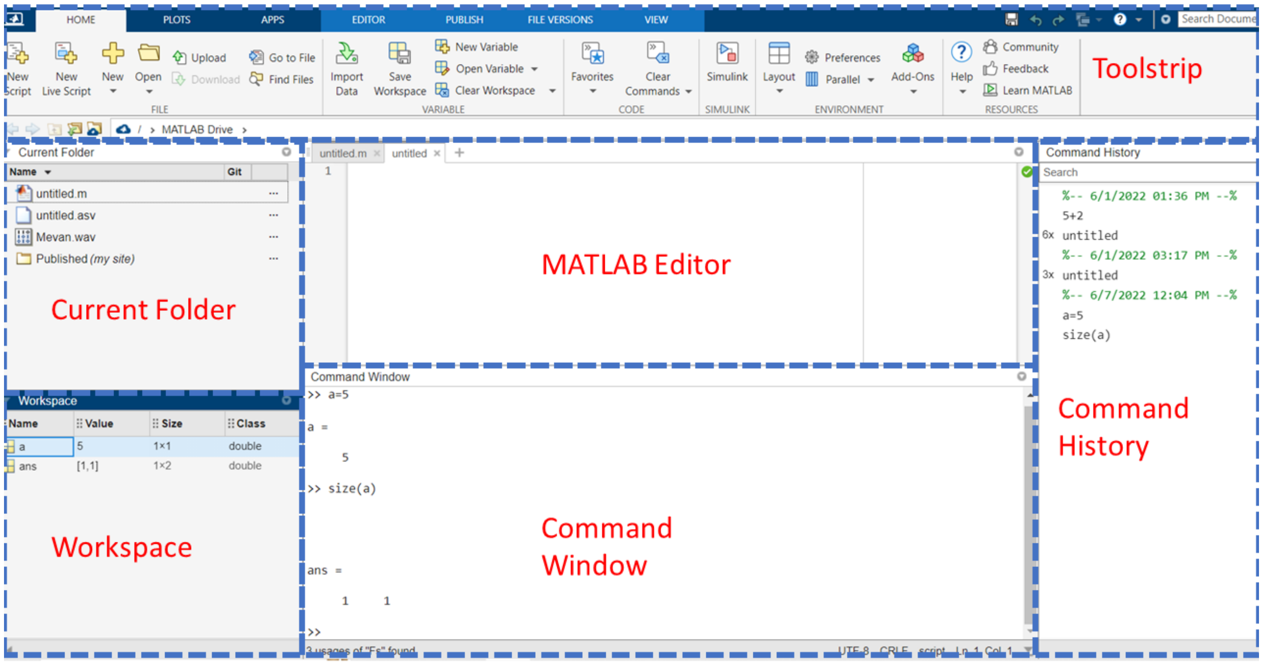
2.2 Installing MATLAB

MATLAB is an extremely useful tool when studying telecommunications. However, it should be noted that MATLAB is not a free software. If you have MATLAB already installed in your computer or if you have a MATLAB license you can use the desktop version for practicals. Otherwise, you can use the trial version of the MATLAB online (`https://uk.mathworks.com/products/matlab-online.html`) version. A Guide for setting up MATLAB online will be provided to you.

2.3 MATLAB Layout

Figure 2.1 shows the layout of MATLAB online version (The layout of MATLAB desktop version is similar). The layout mainly consists of,the following sections.

- Command Window
- MATLAB Editor
- Command History
- Current Folder
- Workspace



Current Folder:

Name	Git
untitled.m	
untitled.asv	
Mevan.wav	
Published *(my site)*	

Workspace:

Name	Value	Size	Class
a	5	1×1	double
ans	[1,1]	1×2	double

Command Window:

```
>> a=5

a =

     5

>> size(a)

ans =

     1     1

>>
```

Command History:

```
%-- 6/1/2022 01:36 PM --%
5+2
6x untitled
%-- 6/1/2022 03:17 PM --%
3x untitled
%-- 6/7/2022 12:04 PM --%
a=5
size(a)
```

Figure 2.1: The MATLAB Layout

- Toolstrip

Command Window: Area used to execute simple commands or operations directly (Adding two numbers, operations with matrices etc).
MATLAB Editor: Used to write MATLAB scripts or MATLAB functions. Script and functions can contains several operations and commands.
Workspace: Displays the details of all the variables defined in command window and script area sections in the current session.
Command History: Displays all the commands executed in command window and script area sections in the current session.
Current Folder: Shows the current folder. You can use this section to change the current directory.

2.4 Matrices and Operations in MATLAB

All MATLAB variables are multidimensional arrays, irrespective of the type of data. Typically, we will encounter only two-dimensional arrays (also known as matrices) in MATLAB. For example a real number is represented as a 1×1 matrix. The variable names are,

- case sensitive
- can contain up to 31 characters
- must start with a letter.

2.4.1 Scalars, Vectors, and Matrices

All of the above data types are represented as matrices in MATLAB.
Scalar: A scalar is simply a real/complex number. A scalar is represented as a 1×1 matrix. Type in the following command in the command window to define the scalar a and set it's value to 5.

$$>> a = 5 \tag{2.1}$$

Observe the output. You can suppress the output of a command by simply typing ";" after the command (Use $>> a = 5;$). The size(A) function returns the shape of the matrix A. Type in $>>$ size(a) to obtain the shape of the matrix.

Vectors: Vectors can be either row vectors or column vectors. An example column vector c is given below

$$c = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}. \tag{2.2}$$

An example row vector r is shown below.

$$r = \begin{bmatrix} 1 & 2 & 3 & 4. \end{bmatrix}. \tag{2.3}$$

In MATLAB the above column vector is represented as a 4×1 matrix, whereas the above row vector is represented as a 1×4 matrix. To define the column vector c, input,

$$>> c = [1;2;3;4] \tag{2.4}$$

To define the row vector r, input,

$$>> r = [1,2,3,4] \tag{2.5}$$

or

$$>> r = [1\ 2\ 3\ 4] \tag{2.6}$$

Confirm the shapes of each vector using the size command.

There are different ways of declaring vectors. For example a vector having elements between 0 and 10 with intervals of 0.1 can be defined as below.

$$>> c1 = (0 : 0.1 : 10) \tag{2.7}$$

A vector having 100 equally spaced elements from 0 to 2π can be defined as below.

$$>> c1 = \mathrm{linspace}(0, 2 * \mathrm{pi}, 100) \tag{2.8}$$

Matrices: Let us define the matrix A given by,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{2.9}$$

in MATLAB. We can use the command,

$$>> A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9] \tag{2.10}$$

2.4.2 Indexing in Matrices

The matrix indices begin from 1 (note that in many programming languages the indexing begins with 0). In addition indices must be positive integers within the range of the dimensions of the matrix.

Try out the following commands for matrix A, that was declared earlier, and understand the results.

1. $>> A(2,3)$ - Returns the scalar at location $(2,3)$ of the matrix as a 1×1 matrix (scalar).
2. $>> A(:,3)$ - Returns the third column of the matrix as a 3×1 matrix (column vector).

3. $>> A(6)$ - Returns the sixth element of the matrix as a 1×1 matrix (Elements are counted column-wise as $1, 4, 7, 2, 5, 8$).
4. $>> A(1, :)$ - Returns the first row of the matrix as a 1×3 matrix (row vector).
5. $>> A(2 : 3, 1 : 2)$ - Returns the sub-matrix of matrix A with rows ranging from 2 to 3 and columns ranging from 1 to 2.

Indexing in vectors is similar. You can use commands $>> c(3)$ and $>> r(3)$ to obtain the third elements of the vectors c and r.

2.4.3 Matrix Concatenation

Matrix concatenation arises in various applications. Two or more matrices can be concatenated. For example consider the matrices,

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, Y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, Z = \begin{bmatrix} 9 & 10 & 11 \\ 12 & 13 & 14 \end{bmatrix}. \tag{2.11}$$

These matrices can be concatenated row-wise to form the matrix T where,

$$T = \begin{bmatrix} 1 & 2 & 5 & 6 & 9 & 10 & 11 \\ 3 & 4 & 7 & 8 & 12 & 13 & 14. \end{bmatrix}. \tag{2.12}$$

We can accomplish this in MATLAB using,

$$>> T = [X\ Y\ Z]; \tag{2.13}$$

Also we can obtain the matrix M by concatenating X and Y column-wise as,

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}. \tag{2.14}$$

This can be accomplished in MATLAB using,

$$>> M = [X; Y]; \tag{2.15}$$

Note that when performing concatenation, special attention must be given to the dimension of the matrices. For instance, $>> M = [X; Y; Z]$ will generate an error since the matrix Z has 3 columns.

2.4.4 Arithmetic Operations

The basic arithmetic operations that can be performed in MATLAB are listed below.

- + Addition
- - Subtraction
- * Multiplication
- $\wedge$ Power
- $'$ Transpose for a real matrix
- \ left division ($>> x = A\backslash b$ is the solution of $Ax = b$)
- / right division ($>> x = A/b$ is the solution of $xA = b$)

Dimensions must agree when performing matrix addition, subtraction etc. For matrices * performs matrix multiplication (not the element-wise multiplication). Hence, the multiplied matrices should have proper dimensions.

We can also mix scalars and matrices in arithmetic operations. For example $a + b$, where a is a matrix and b is a scalar will add b to each element of a. Similarly, $a * b$ will multiply each element of a by b.

To perform element-wise operations between matrices, we precede the operators by '.'

- $a. * b$ multiplies each element of a by the respective element of b
- $a./b$ divides each element of a by the respective element of b
- $a.\backslash b$ divides each element of b by the respective element of a
- $a. \wedge b$ raises each element of a by the respective element of b.

Task 15. *If* $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, *use MATLAB to obtain the following.*

1. $A + 3$
2. $A - 3$
3. $A * 3$
4. $A/3$
5. $A. \wedge 2$

Task 16. *If* $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, *and* $B = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$, *use MATLAB to obtain the following.*

1. $A * B$
2. $A. * B$
3. $A./B$
4. $A.\backslash B$
5. $A. \wedge B$

2.4.5 Relational and Logic Operators in MATLAB

The relational operators are used to perform comparisons between MATLAB variables. Following are the major relational operators.

- == Equal
- ∼= Not equal to
- < Strictly smaller
- > Strictly greater
- <= Smaller than or equal to
- >= Greater than or equal to

For two equal dimensional matrices A and B, >> $A == B$ returns a logical matrix C with the same dimensions as A and B. Each entry of this matrix indicates whether the corresponding entries of A and B are equal. Other operators are similar.

The logical operators perform element-wise logical operations. >> $A\&B$ and >> $A|B$ perform element-wise AND and element-wise OR operations between matrices A and B, respectively.

Task 17. *Apply the above operations to the two matrices defined in task 16.*

2.5 Control Structures

So far we defined matrices and performed simple operations using MATLAB. In addition to this knowledge, writing a MATLAB program for an application requires the use of control structures. Under control structures we will examine conditionals (or selection) and loops.

2.5.1 Conditionals

Conditionals are used to execute one or more statements if a particular condition is met. In programming languages conditionals are implemented using "If" statements.

The syntax of an "If" statement is given below.

```
if (condition_1)
   MATLAB commands
elseif (condition_2)
   MATLAB commands
elseif (condition_3)
   MATLAB commands
else
   MATLAB commands
```

Task 18. *Type the following commands in the MATLAB command window and observe the results.*

```
value1 = 5;
value2 = 10;
if value1 > value2
 value3 = 1
elseif value1 == value2
 value3 = 0
else
 value3 = −1
end
```

```
value1 = 5;
value2 = 10;
if value1 > value2
 disp('Greater than')
elseif value1 == value2
 disp('Equal to')
else
 disp ('Less than')
end
```

disp() is a built-in function of MATLAB, that can be used to display texts. There are more advanced built-in functions to do this as well. For the time being, we shall use disp(). You will be introduced to such built in functions later in this workshop.

2.5.2 Loops

Loops are used to repeatedly execute the same set of statements on all the objects within a sequence (for example executing the same set of statements on all the objects of a vector). We will study the "For" and the "While" loops.

Given below is the syntax of a "For" loop.

```
for controlVariable = start : step : stop
    MATLAB commands
end
```

Task 19. *In order to understand the "For" loop, execute the following MATLAB commands.*

```
for i = 1:2:10
 i
end
```

```
total = 0;
for i = 1:2:10
 total = total + i;
end
total
```

Task 20. *We can implement the above code using a while loop as follows. Execute the following MATLAB code.*

```
total = 0;
i = 1
while i<10
 total = total + i;
 i = i + 2;
end
total
```

A *break* statement can be used within a loop to stop the loop's execution after a particular condition is met.

```
total = 0;
for i = 1:2:10
 total = total + i;
 i
 if(total>20)
    break;
 end
end
total
```

Similarly, a *continue* statement implemented within a loop will check for a particular condition. But instead of completely stopping the execution of the loop, it will stop the execution of the commands after the continue statement of the iterations in which the condition is met. To observe the difference between "break" and "continue", run the following commands.

```
total = 0;
for i = 1:2:10
 if(i == 7)
    continue;
 end
 total = total + i;
end
total
```

```
total = 0;
for i = 1:2:10
 if(i == 7)
    break;
 end
 total = total + i;
end
total
```

2.6 MATLAB Scripts and Functions

When problems become complicated and require re–evaluation, entering commands on the MATLAB command window is not practical. The solution is to write functions/scripts as m-files in MATLAB, and reuse them wherever necessary.

2.6.1 MATLAB Scripts

Scripts are collections of commands executed in sequence, when called. They are saved with an extension ".m".

On the home tab of the MATLAB main window, there is an icon named "New Script". Click on it and open the MATLAB editor. Type the following commands on the editor window.

```
% This is a comment in MATLAB
array = [1 2 3 4 5 6 7 6 5 4 3 2 1];
numOfElements = length(array);
total = 0;
for i = 1:numOfElements
 total = total + array(1,i);
end
total
```

Save the script as myFirstScript.m (You can give any name. Only the extension should be .m). Type "myFirstScript" in command window and enter. You will observe the answer for total. Every time you want to execute the same set of commands, you can save the common set of commands as a MATLAB script by a certain name, and just call out that name on the command window.

2.6.2 MATLAB Functions

A Function is a 'black box' that communicates with workspace through input(s) and output(s). Figure 2.2 illustrates the basic components of a MATLAB functions.

There are two types of MATLAB functions. There are built in MATLAB functions and the user defined functions.

Built-in Functions

Built-in functions are the functions that are defined in MATLAB and can be readily used without implementation. There are thousands of Built-in functions in MATLAB. They can be used as tools to write MATLAB programs very conveniently and efficiently. Examples for built-in functions are min, max, mean, size, and inv functions. For example min function can be used to find the minimum element of a vector.

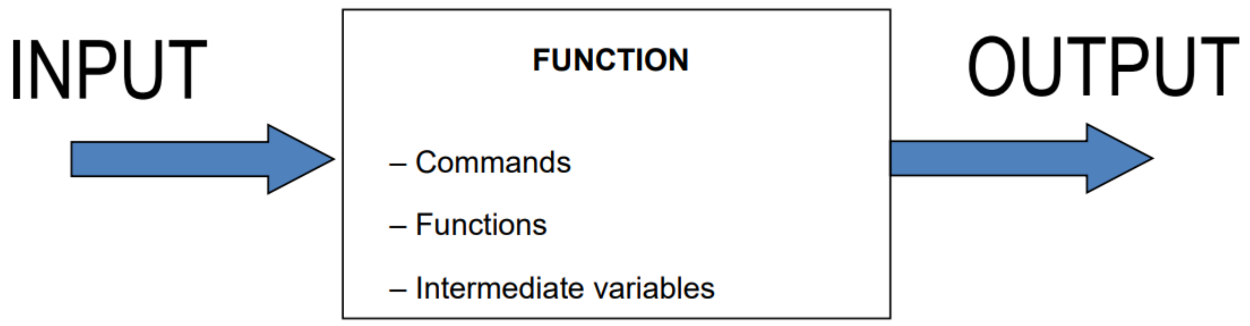


Figure 2.2: The Components of a MATLAB Function

Task 21. *Find the operations performed by the other functions mentioned. Explore different types of MATLAB functions and their functionalities.*

Using the MATLAB **help** feature is one of the most effective ways of learning MATLAB. On the *home* window of MATLAB, there is an icon named *help*. Click on it and get the Product help window. You can search for any Built-in function in this help window.

You can also type "»help *functionName*" on the MATLAB command window to get the details of known built-in functions. For example you can type "»help sin" to get the details about sin function.

User defined functions

User defined functions are the custom functions defined by the users for their own applications. Similar to MATLAB scripts, such functions are written in the MATLAB editor. General syntax of a user defined function is given below.

```
function output(s) = function_name(inputs)
  MATLAB commands
```

The user defined functions are written in separate ".m" files. The "function_name" must match the file name. In other words, the ".m" file containing the function should be saved by the name function_name.m.

Implement the following function in a new ".m" file.

```
function x = impedance (r,c,l,w)
%Impedance function calculates Xc, Xl, Z(magnitude) and Z(angle) of the RLC
%connected in series.
if nargin ~= 4
 error('Please input 4 arguments')
end
x(1) = 1/(w*c);
x(2) = w*l;
Zt = r + (x(2) - x(1))*i;
x(3) = abs(Zt);
x(4) = angle(Zt);
```

Task 22. *Now that you know how to write MATLAB scripts as well, in order to call the function you have just defined, write the following script in a separate ".m" file and run it on the MATLAB command window. You can use any name for your script.*

```
R = input("Enter R: ");
C = input("Enter C: ");
L = input("Enter L: ");
W = input("Enter w: ");
```

```
Y = impedance(R, C, L, W)
```

You can call user defined functions within a script or within another user defined function. Directly calling them from the MATLAB command window is also possible.
Note: Make sure that all your scripts and functions are saved in the same folder, before you execute them. You should set that folder as your current work folder as well. Instructors will help you in this regard.

Task 23. *Complete the following recursive function to generate the n-th term of the Fibonacci sequence. In the Fibonacci sequence, the zeroth term is zero and the first term is 1. Every term which follows is equal to the sum of the previous two terms.*

```
function fn = fibonacci (n)
%Generates the nth Fibonacci number
%Input = n
%Output = F(n)
if n < 0
 error('n less than zero')
elseif n == 0
 fn = 0;
elseif n == 1
 fn = 1;
else
 %Complete the function by entering the %relevant MATLAB commands here
end
```

2.7 Plotting

Visualizing data is an important aspect in Engineering. Graphs are used to get a vivid idea about the data and information that you can obtain using certain systems and applications. MATLAB supports plotting of graphs in 2D as well as in 3D. We shall first learn on how to plot in 2D. The following example will give you a good understanding about plotting in 2D.

Imagine we need to plot the graphs of sin(x) and cos(x).

```
close all;
clear all;
clc;
x = 0:0.01:2*pi;
y1 = sin(x);
y2 = cos(x);
plot (x,y1,'r-');
hold on;
plot (x,y2,'g--');
xlabel('Angle in radians');
ylabel('y1 and y2');
title('My first graph in Matlab');
legend('sin(x)','cos(x)');
```

Functionality of each of the built-in functions should be easy to understand. The instructors in the labs will help you if you have any issues. The main command to know is plot(). plot(x,y) will simply plot y against x. plot(x,y,LineSpec) can be used to specify the style of the curve (color, dotted/ dashed etc.). Visit the plot function's help page and find more about it. To learn about 3D plotting in MATLAB, use the help window as well as online materials (`https://uk.mathworks.com/help/matlab/visualize/creating-3-d-plots.html`). The main functions that you should be familiar with are meshgrid, mesh and surf.

2.8 Self-Learning MATLAB Exercises

Instructors/Lecturers will help you to go through these tasks. You are expected to finish these at home. The idea of these tasks is to enhance your knowledge on MATLAB as well as to familiarize you with different applications of MATLAB.

2.8.1 Use of in-built functions and vectorization

MATLAB is optimized for operations involving matrices and vectors. Consider a MATLAB inbuilt-function f which has a scalar input and a scalar output. If we need to calculate f for each entry of a matrix X we can use $f(X)$. This is known as vectorization. For example if need to calculate the sine value of each entry of the matrix

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \tag{2.16}$$

we can use sin(X).

Task 24. *Generate a 2D array (with the name randomArray and dimensions (5,10)) comprising of random integers which are uniformly distributed in the region 50 to 100 using the built in functions in MATLAB (You can use rand function).*

1. *Calculate the sine value of each of the elements in "randomArray" by*

   (a) *vectorization.*

   (b) *using a for loop.*

2. *Use the tic-toc timer in MATLAB to measure the time elapsed in each case.*

3. *Rename the script file as "task24_<index>.m" and submit to the link on Moodle.*

2.8.2 Sound Processing in MATLAB

Save the sound file anthem.wav in your current work folder. To read the sound file into MATLAB as a matrix, use the following command.

```
[anthem,fs]=audioread('anthem.wav');
```

To play it back,

```
sound(anthem,fs)
```

The array named anthem comprises the numerical values which represent the amplitudes of the sound file. We can play around those numbers and do lots of processing, just by doing various matrix operations and arithmetic operations. If you are interested, you can experiment and learn.

2.8.3 User defined functions

Task 25. *Write a simple MATLAB script (in an m-file) to perform the following task. Input 2 arrays (2D) with the same dimensions and compare the corresponding elements of those arrays. If equal, output 1, and if not, output 0. The output array should be a binary array with the same dimensions as that of the input arrays. If the input arrays do not have the same dimensions, output an error message.*
Rename the script file as "task25_<index>.m" and submit to the link on Moodle.

2.8.4 Image Processing

MATLAB can be used for image processing and Machine vision applications. To get an idea about such applications, execute the following commands and observe the results. Those who are interested can learn more by experimenting and self-study.

```
image = imread("someImage.jpg"); %someImage.jpg should be saved in your work folder.
imshow(image)
grayImage = rgb2gray(image);
imshow(grayImage)
imTemp = image;
imTemp(:,:,2) = 0;
imTemp(:,:,3) = 0;
imshow(imTemp)
```

2.9 Introduction to Simulink

Simulink is a program for simulating signals and dynamic systems. As an extension of MATLAB, Simulink adds many features specific to the simulation of dynamic systems while retaining all of MATLAB's general purpose functionality.

You can access Simulink by clicking on the Simulink icon on the home tab of the MATLAB main window.

2.9.1 Simulink Layout

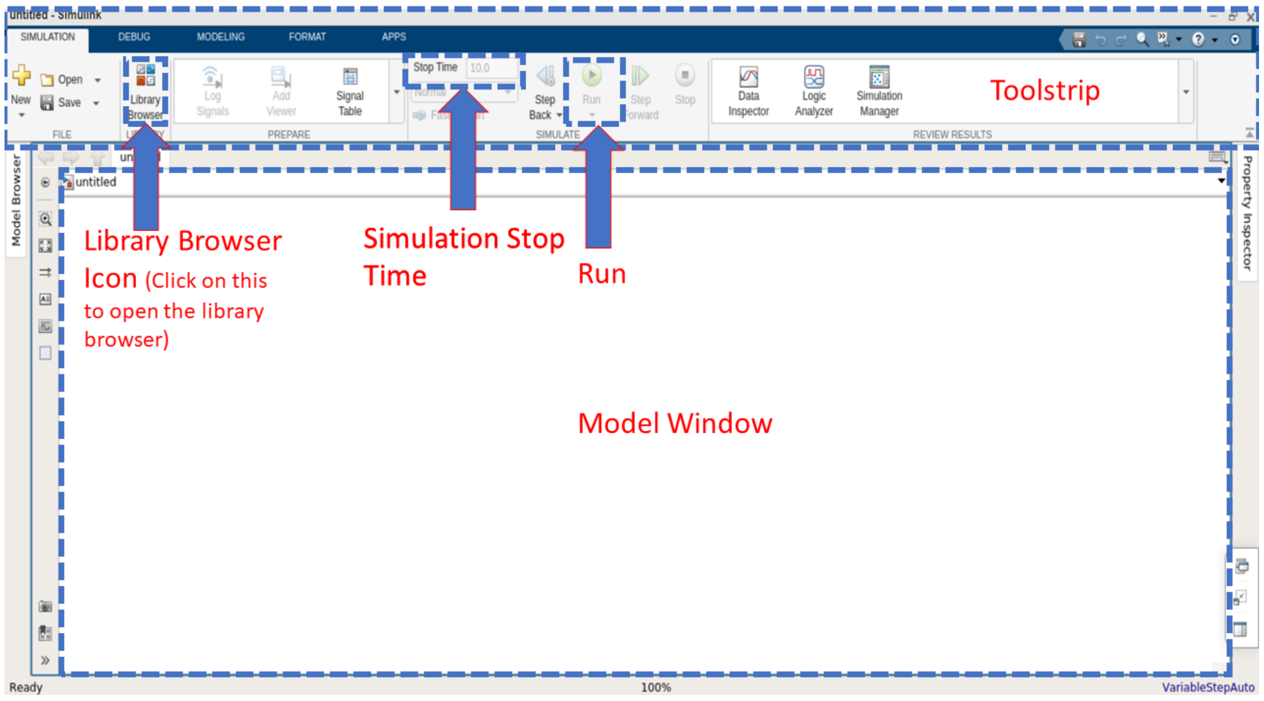


Figure 2.3: The Simulink Layout

Figure 2.3 shows the Simulink Layout. To construct a Simulink model, you first copy blocks from the Simulink Library Browser to the Model window. As shown in Figure 2.3 Simulink Library Browser can be opened by the Simulink Library Icon located in the toolstrip.

2.9.2 Constructing and simulating a simple model in Simulink

In this section you will develop a simple Simulink model to integrate a sine wave. Both the original wave and the integrated wave will be displayed for comparison purposes.

Figure 2.4 shows the block diagram of the completed model. We will analye the step by step process of buiding the above model.

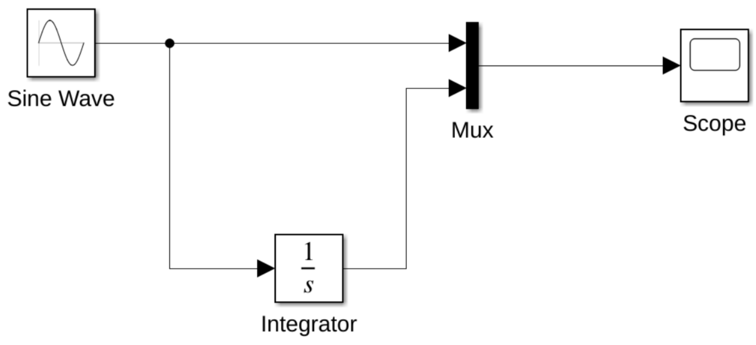


Figure 2.4: The Simulink Model

1. Select *File* > *New* > *Model* to construct a new model.
2. To create the simple model shown in Figure 2.4, you need four blocks:
   - Sine Wave — To generate an input signal for the model
   - Integrator — To process the input signal
   - Scope — To visualize the signals in the model
   - Mux — To multiplex the input signal and processed signal into a single scope

   You can add each block using the following steps.

   - Select the **Simulink/Sources** library in the Simulink Library Browser. The Simulink Library Browser will then display the Sources library. Select the **Sine Wave** block in the Simulink Library Browser, then drag it to the model window. A copy of the Sine Wave block appears in the model window. Click on the Sine Wave block to change it's properties.Change the Frequency to 3 rad/s and the Sample time to 0.001.
   - Select the **Simulink/Sinks** library in the Simulink Library Browser. Select the **Scope** block from the Sinks library, then drag it to the model window. A Scope block appears in the model window.
   - Select the **Simulink/Continuous** library in the Simulink Library Browser. Select the **Integrator** block from the Continuous library, then drag it to the model window. An Integrator block appears in the model window.
   - Select the **Simulink/Signal Routing** library in the Simulink Library Browser. Select the **Mux** block from the Sinks library, then drag it to the model window. A Mux block appears in the model window.
3. Make the connections as Figure 2.4. The connection between Sine Wave block and Mux block is somewhat different from the other three. Because the output port of the Sine Wave block already has a connection, you must connect this existing line to the input port of the Integrator block. The new line, called a branch line, carries the same signal that passes from the Sine Wave block to the Mux block. To weld a connection to an existing line:
   - Position the mouse pointer on the line between the Sine Wave and the Mux block.
   - Press and hold the Ctrl key, then drag a line to the Integrator block's input port.

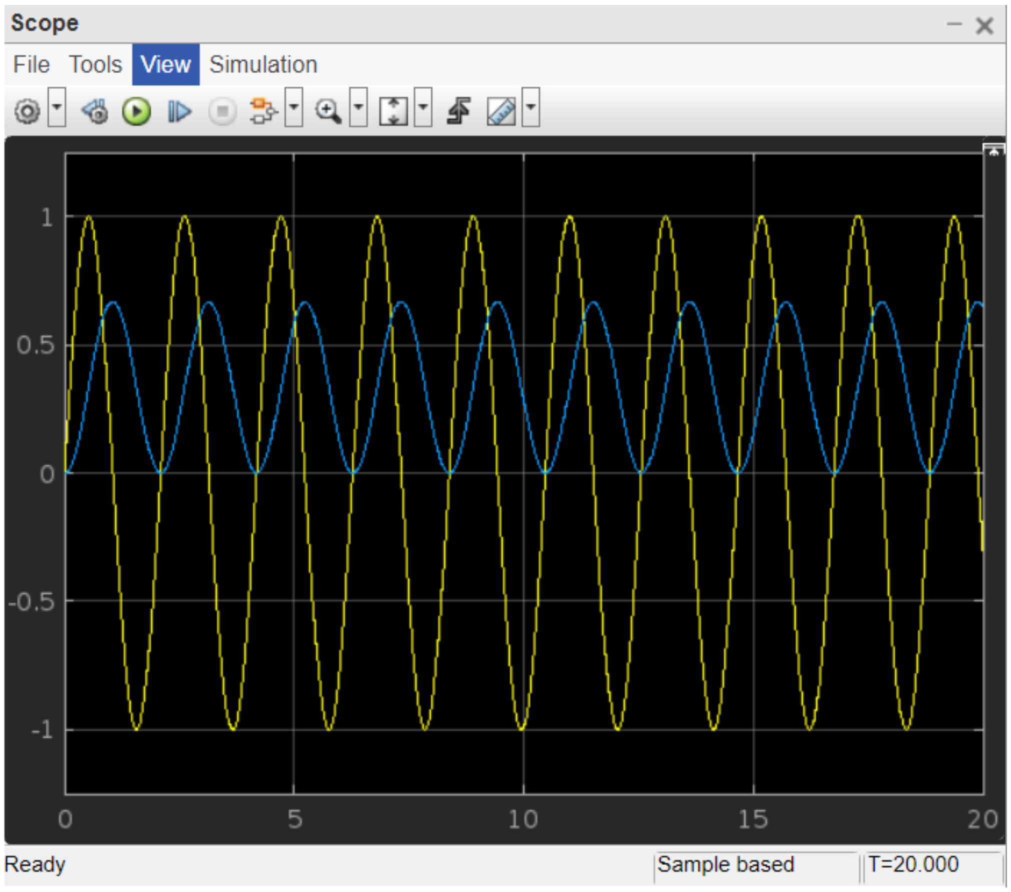


Figure 2.5: The Sine Wave and the Integration of the Sine Wave

4. Set the simulation stop time to 20s (shown in Figure 2.3).

5. Now simulate your sample model and observe the results. To run the simulation click *Run* button shown in Figure 2.3.

6. Double-click the **Scope** block in the model window. The Scope window should display the simulation results as shown in Figure 2.5.

2.10 Self-Learning Simulink Exercises

Task 26. *Study following Simulink blocks using Scope:*

- *Signal generator*
- *Digital clock*
- *Step*
- *Uniform random number*
- *Pulse generator*
- *Sine wave function*

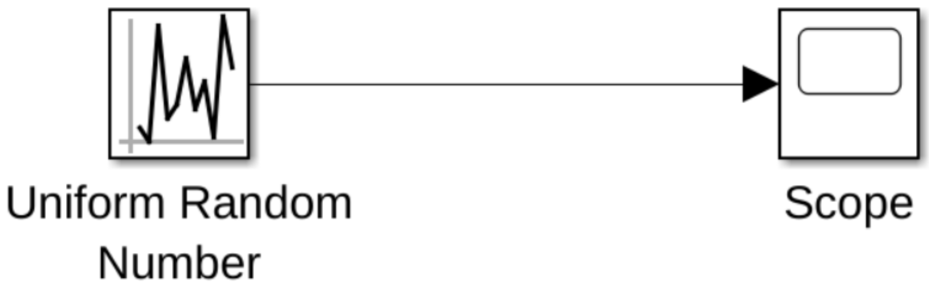


Figure 2.6: The Uniform Random Number Block

For example the use of uniform random number is shown in Figure 2.6.

Select the **Simulink/Sources** library in the Simulink Library Browser. Select the **Uniform Random Number** block in the Simulink Library Browser, then drag it to the model window. Click on the Uniform Random Number block and change the Minimum to -2 and the Sample time to 0.3.

Set the simulation time to 20s and Run the simulation. You will observe a graph similar to Figure 2.7.

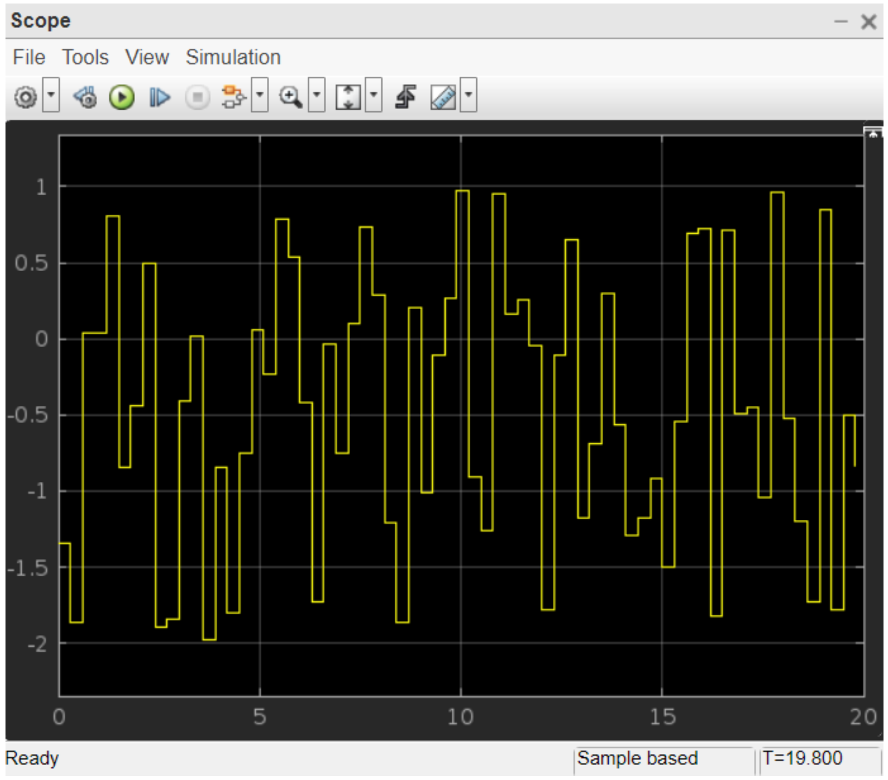


Figure 2.7: The Plot of the Uniform Random Numbers

Task 27. *Follow the steps below to design a multiplier with a square-wave input.*

1. *Start a Simulink session.*
2. *Copy the pulse generator block by dragging it to your model (The block can be found in Simulink/Sources). Specify the parameters as follows.*
   - *Period: 1*

- *Pulse width: 50%*
- *Amplitude: 0.7*

3. *Next, multiply the output of the signal generator by a gain. Gain block can be found in the Simulink/Math Operations library. Set the gain to 0.9 from the properties of the block.*
4. *Use the Scope block to analyze the output.*
5. *Connect the blocks.*
6. *Set the simulation stop time to 10s and run the simulation.*
7. *Save the model as "task27_<index>.slx" and submit to the link on Moodle.*

♣ The End ♣

Workshop 3: OMNET++ Simulation Software

Objective: To become familiar with the OMNET++ environment for the simulation of communication networks.

Outcome: After successfully completing this session, the student would be able to

1. Get a basic understanding of the OMNET++ simulation platform.
2. Simulate a simple communication network using OMNET++

Equipment Required:

1. A Personal Computer
2. OMNET++ (most recent version)

3.1 Introduction

Communication networks allow many users to communicate with each other irrespective of geographical location and run many other applications and obtain services. The resources of the communication network are shared by all its users. Protocols allow users in diverse geographical locations, using different types of hardware and software to communicate with each other over a network using a standard set of rules.

Simulation tools are commonly used to study the performance of communication networks and protocols in different situations as well as to design new protocols. OMNET++ (**O**bjective **M**odular **Ne**twork **T**estbed in C++), which we will use in this lab, is a well-known open-source simulation tool for communications networks used extensively by the academic community. In this session, you will first get familiar with the OMNET++ simulation environment, and then use it to implement a very simple communication network.

Use the learning resources at `https://docs.omnetpp.org` to learn more about OMNET++. In particular, to learn OMNET++ in detail, see `https://doc.omnetpp.org/omnetpp/manual/`. This session is an adaptation of the initial parts of the *Tictoc* tutorial. After this introductory session, please complete the *Tictoc* tutorial available at `https://docs.omnetpp.org/tutorials/tictoc/`.

3.2 Setting up the OMNET++ simulation environment

Step A: Follow following video (upto 6:55) to download and install OMNET++ on your computer.
`https://www.youtube.com/watch?v=PfAWhrmoYgM&list=PLaBPUIXZ8s4AwAk5EelikvvyG4EzX2hpx&index=2`

The process of getting OMNET++ ready to run on your computer involves downloading and installing the software, configuring and building it, and then verifying your installation.

Once you have completed the installation step, when you look into the new omnetpp-6.0 directory, you should see directories named `doc, images, include, tools`, etc., and files named `mingwenv.cmd, configure, Makefile`, and others. In the doc directory, you will find the document `InstallGuide` which will help you further on the installation process.

At the end of this step, you will have launched OMNET++ and will see the Welcome Screen. Close this screen for now.

Step B: Next, follow the instructions at `https://inet.omnetpp.org/Installation.html` to install the INET framework.

The INET Framework is the standard protocol model library of OMNeT++. INET contains models for the Internet protocol stack and many other protocols and components. Several other simulation frameworks take INET as a base, and extend it into specific directions, such as vehicular networks (Veins, CoRE), overlay/peer-to-peer networks (OverSim), or LTE (SimuLTE).

3.3 Creating and running your first OMNET++ project

Let us begin with a "network" that consists of two nodes. The nodes will do something simple: one of the nodes will create a packet, and the two nodes will keep passing the same packet back and forth. We'll call the nodes `tic` and `toc`. There is a 100ms propagation delay between `tic` and `toc`.
Step A: Create the project Tictocfirst using the guidelines provided below.

- Start the OMNeT++ IDE (Integrated Development Environment) by typing `omnetpp` in your terminal (Command window).
- Once in the IDE, choose *New* -> *OMNeT*++ Project from the menu.
- A wizard dialog will appear. Enter `tictocfirst` as project name, choose Empty project when asked about the initial content of the project, then click *Finish*. An empty project will be created, as you can see in the *Project Explorer*.
- A wizard dialog will appear. Enter tictocfirst as project name, choose Empty project when asked about the initial content of the project, then click Finish. An empty project will be created, as you can see in the Project Explorer.

This project will hold all files that belong to your simulation. You need to create three files the `NED` file, the `.cc` file and the `.ini` file as described below:

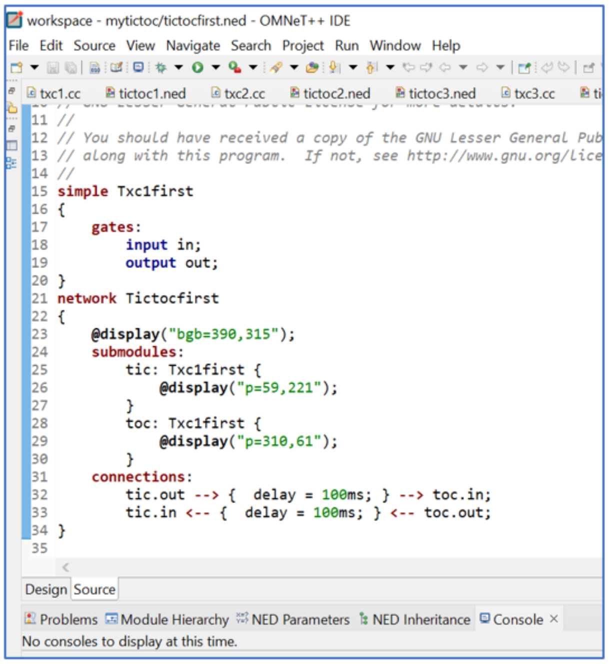
- **The NED file:** OMNeT++ uses the NED (NEtwork Description) language to define components and to assemble them into larger units like networks. NED lets the user declare simple modules, and connect and assemble them into compound modules. The user can label some compound modules as *networks*; that is, self-contained simulation models. Channels are another component type, whose instances can also be used in compound modules. We start implementing our model by adding a NED file. To add the file to the project, right-click the project directory in the *Project Explorer* panel on the left, and choose *New* -> *Network Description File* (NED) from the menu. Enter `tictocfirst.ned` when prompted for the file name.

  Switch into *Source* mode to enter text into your NED file. Your NED file `tictocfirst.ned` contents should be as below.

```
simple Txc1first
{
gates:
input in;
output out;
}
network Tictocfirst
{
@display("bgb=390,315");
submodules:
tic: Txc1first {
@display("p=59,221");
}
toc: Txc1first {
@display("p=310,61");
}
connections:
tic.out −−> { delay = 100ms; } −−> toc.in;
tic.in <−− { delay = 100ms; } <−− toc.out;
}
```

The first block in the file declares `Txc1first` as a simple module type. Simple modules are atomic on NED level. They are also active components, and their behavior is implemented in C++. The `Txc1first` simple module type is represented by the C++ class `Txc1first`. Two instances of `Txc1first`, named `tic` and `toc` will pass the same messages between each other with a delay of 100ms. The name of this small network is `Tictocfirst`.

After you have created your NED file, switch back to *Design* mode to see the network you have created. The view of the two modes is shown in Figure 3.1.


workspace - mytictoc/tictocfirst.ned - OMNeT++ IDE
File Edit Source View Navigate Search Project Run Window Help
txc1.cc tictoc1.ned txc2.cc tictoc2.ned tictoc3.ned txc3.cc
Design Source
Problems Module Hierarchy NED Parameters NED Inheritance Console
No consoles to display at this time.

```
//
// You should have received a copy of the GNU Lesser General Pub
// along with this program.  If not, see http://www.gnu.org/lice
//
simple Txc1first
{
    gates:
        input in;
        output out;
}
network Tictocfirst
{
    @display("bgb=390,315");
    submodules:
        tic: Txc1first {
            @display("p=59,221");
        }
        toc: Txc1first {
            @display("p=310,61");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

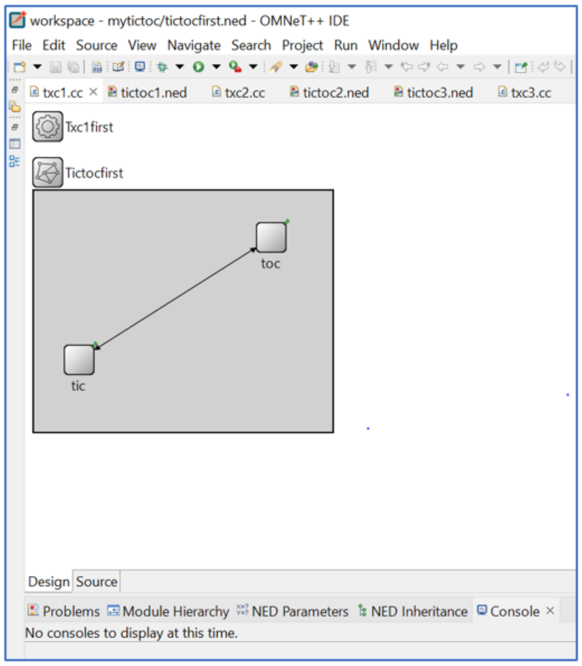

workspace - mytictoc/tictocfirst.ned - OMNeT++ IDE
File Edit Source View Navigate Search Project Run Window Help
txc1.cc tictoc1.ned txc2.cc tictoc2.ned tictoc3.ned txc3.cc
Txc1first
Tictocfirst
toc
tic
Design Source
Problems Module Hierarchy NED Parameters NED Inheritance Console
No consoles to display at this time.

Figure 3.1: Network design with the NED file

- **The C++ file:** `Txc1first` class needs to subclass from OMNeT++'s `cSimpleModule` class, and needs to be registered in OMNeT++. This is done in the `txc1first.cc` file. We now need to implement the functionality of the `Txc1first` simple module in C++. Create a file named `txc1.cc` by choosing *New -> Source File* from the project's context menu (or *File -> New -> File* from the IDE's main menu), and enter the following content:Your `txc1first.cc` file contents should be as below. Note the comments in the code for further clarity.

```
#include <string.h>
#include <omnetpp.h>

  using namespace omnetpp;

  /**
   * Derive the Txc1first class from cSimpleModule. In the Tictocfirst network,
   * both the 'tic' and 'toc' modules are Txc1first objects, created by OMNeT++
   * at the beginning of the simulation.
   */
  class Txc1first : public cSimpleModule
  {
   protected:
     // The following redefined virtual function holds the algorithm.
     virtual void initialize() override;
     virtual void handleMessage(cMessage *msg) override;
  };
```

```
// The module class needs to be registered with OMNeT++
Define_Module(Txc1first);

void Txc1first::initialize()
{
  if (strcmp("tic", getName()) == 0) {
//The 'ev' object works like 'cout' in C++. We add log statements to Txc1 so that it //prints what it is doing.
      EV << "Sending initial message\n";
      cMessage *msg = new cMessage("tictocMsg");
      send(msg, "out");
  }
}

void Txc1first::handleMessage(cMessage *msg)
{
  // msg->getName() is name of the msg object, here it will be "tictocMsg".
  EV << "Received message '" << msg->getName() << "', sending it out again\n";
  send(msg, "out");
}
```

We redefine two methods from `cSimpleModule`: `initialize()` and `handleMessage()`. They are invoked from the simulation kernel: the first one only once, and the second one whenever a message arrives at the module.

In `initialize()` we create a message object (`cMessage`), and send it out on gate out. Since this gate is connected to the other module's input gate, the simulation kernel will deliver this message to the other module in the argument to `handleMessage()` – after a 100ms propagation delay assigned to the link in the NED file. The other module just sends it back (another 100ms delay), so it will result in a continuous ping-pong.

Messages (packets, frames, jobs, etc) and events (timers, timeouts) are all represented by `cMessage` objects (or its subclasses) in OMNeT++. After you send or schedule them, they will be held by the simulation kernel in the "scheduled events" or "future events" list until their time comes and they are delivered to the modules via `handleMessage()`.

We add log statements to `Txc1first` so that it prints what it is doing. OMNeT++ provides a sophisticated logging facility with log levels, log channels, filtering, etc. that are useful for large and complex models, but in this model we'll use its simplest form `EV`, which is an object that works like `cout` in C++.

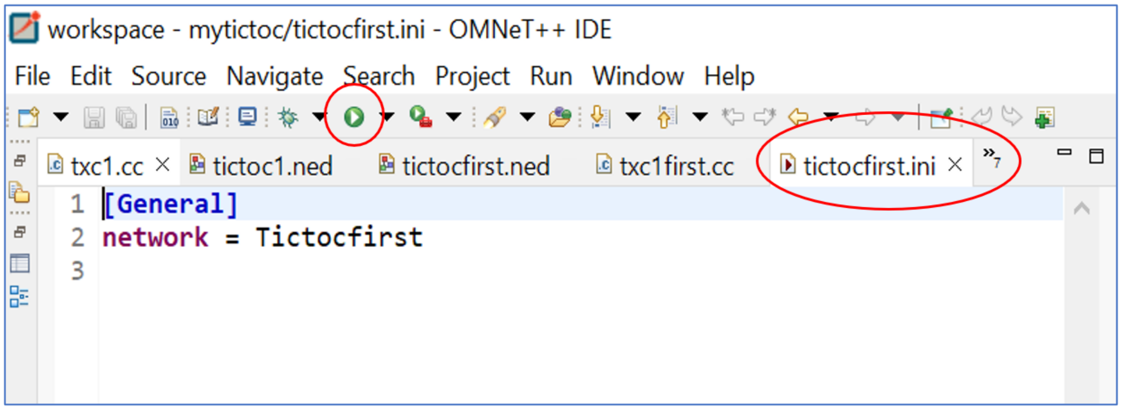
- **The .ini file:** To be able to run the simulation, we need to create an `omnetpp.ini` file. This file tells the simulation program which network you want to simulate (as NED files may contain several networks). Your `omnetpp.ini` file contents should be as below.

```
[General]
network = Tictocfirst
```

The name you have given for the network in your NED file is `Tictocfirst`, which appears here.

Step B: After you have created the three files, follow the steps below to compile and run your first project.

- Launch the simulation by selecting `tictocfirst.ini` (in either the editor area or the *Project Explorer*) as shown in Figure 3.2, and pressing the *Run* button.


workspace - mytictoc/tictocfirst.ini - OMNeT++ IDE
File Edit Source Navigate Search Project Run Window Help
txc1.cc
tictoc1.ned
tictocfirst.ned
txc1first.cc
tictocfirst.ini

```
[General]
network = Tictocfirst
```

Figure 3.2: The OMNET++ IDE

- After successfully launching your simulation, you should see a new GUI window appear, similar to the one in Figure 3.3. The window belongs to `Qtenv`, the main OMNeT++ simulation runtime GUI. You should also see the network containing *tic* and *toc* displayed graphically in the main area.

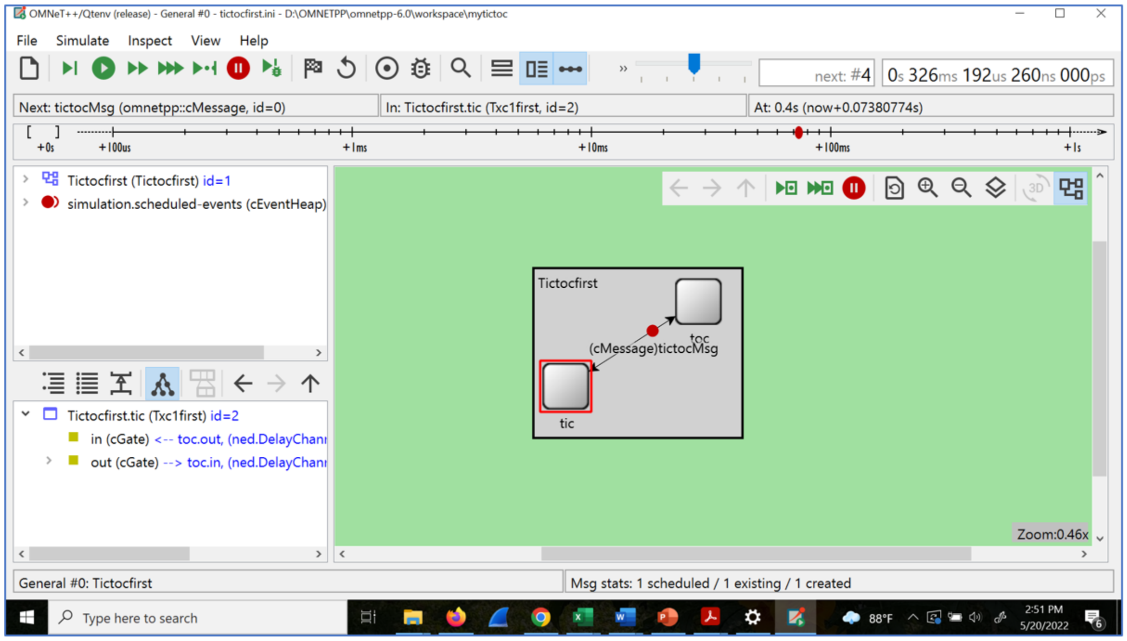

OMNeT++/Qtenv (release) - General #0 - tictocfirst.ini - D:\OMNETPP\omnetpp-6.0\workspace\mytictoc
File Simulate Inspect View Help
next: #4
0s 326ms 192us 260ns 000ps
Next: tictocMsg (omnetpp::cMessage, id=0)
In: Tictocfirst.tic (Txc1first, id=2)
At: 0.4s (now+0.07380774s)
+0s +100us +1ms +10ms +100ms +1s
Tictocfirst (Tictocfirst) id=1
simulation.scheduled-events (cEventHeap)
Tictocfirst.tic (Txc1first) id=2
in (cGate) <-- toc.out, (ned.DelayChann
out (cGate) --> toc.in, (ned.DelayChann
Tictocfirst
toc
(cMessage)tictocMsg
tic
Zoom:0.46x
General #0: Tictocfirst
Msg stats: 1 scheduled / 1 existing / 1 created
Type here to search
88°F
2:51 PM
5/20/2022

Figure 3.3: The Qtenv runtime GUI

- Press the *Run* button on the toolbar to start the simulation. What you should see is that `tic` and `toc` are exchanging messages with each other.

The main window toolbar displays the current simulation time. This is virtual time, it has nothing to do with the actual (or wall-clock) time that the program takes to execute. Actually, how many seconds you can simulate in one real-world second depends highly on the speed of your hardware and even more on the nature and complexity of the simulation model itself.

You can play with slowing down the animation or making it faster with the slider at the top of the graphics window, single-step through it or run in express mode using with the tools on the top tool bar. You can stop the simulation by hitting the STOP button on the toolbar. You can exit the simulation program by clicking its *Close* icon or choosing *File -> Exit*.

3.4 Observing simulation results

In this section we will learn two simple methods of observing simulation results.

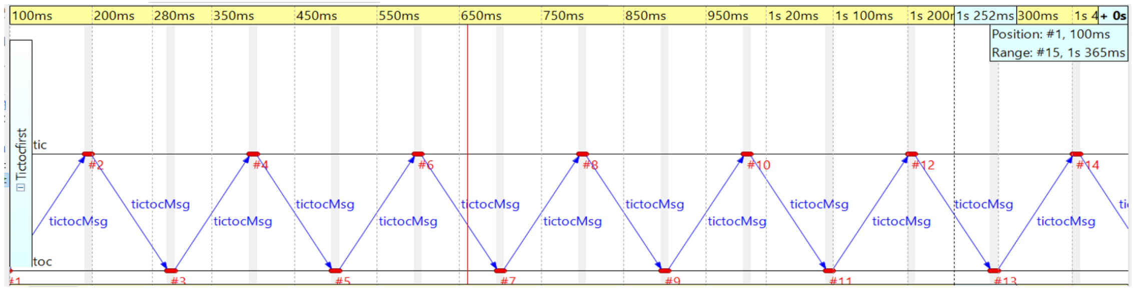
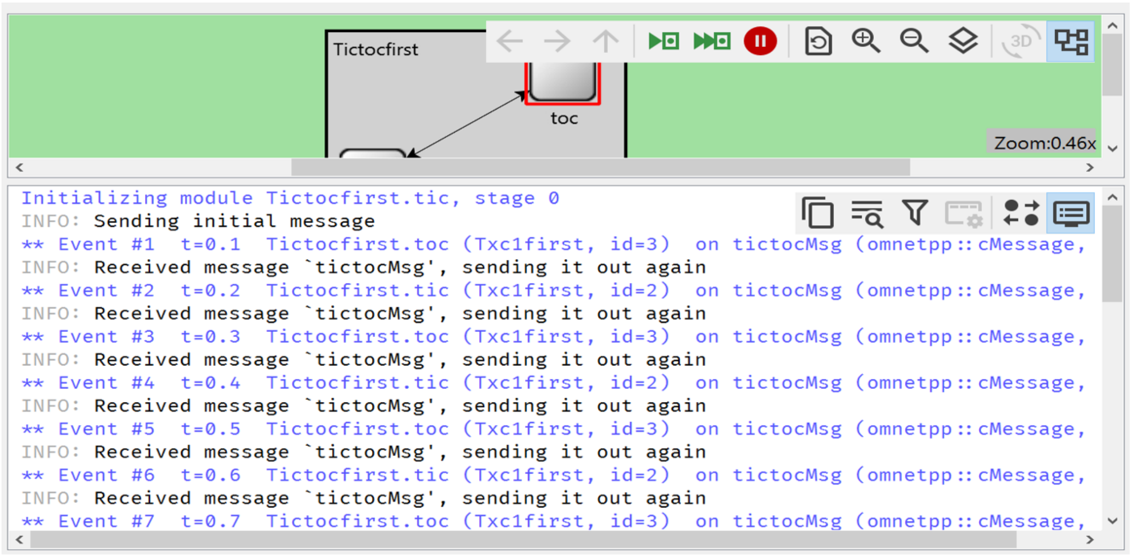


Figure 3.4: Message exchanges



```
Initializing module Tictocfirst.tic, stage 0
INFO: Sending initial message
** Event #1  t=0.1  Tictocfirst.toc (Txc1first, id=3)  on tictocMsg (omnetpp::cMessage,
INFO: Received message `tictocMsg', sending it out again
** Event #2  t=0.2  Tictocfirst.tic (Txc1first, id=2)  on tictocMsg (omnetpp::cMessage,
INFO: Received message `tictocMsg', sending it out again
** Event #3  t=0.3  Tictocfirst.toc (Txc1first, id=3)  on tictocMsg (omnetpp::cMessage,
INFO: Received message `tictocMsg', sending it out again
** Event #4  t=0.4  Tictocfirst.tic (Txc1first, id=2)  on tictocMsg (omnetpp::cMessage,
INFO: Received message `tictocMsg', sending it out again
** Event #5  t=0.5  Tictocfirst.toc (Txc1first, id=3)  on tictocMsg (omnetpp::cMessage,
INFO: Received message `tictocMsg', sending it out again
** Event #6  t=0.6  Tictocfirst.tic (Txc1first, id=2)  on tictocMsg (omnetpp::cMessage,
INFO: Received message `tictocMsg', sending it out again
** Event #7  t=0.7  Tictocfirst.toc (Txc1first, id=3)  on tictocMsg (omnetpp::cMessage,
```

Figure 3.5: Simulation logs

- Observing message exchanges: Run the simulation again. Use the ***Record*** button in the `Qtenv` graphical runtime environment after launching, to record the message exchanges during the simulation into an *event log file*. After you exit the simulation, the log file can be analyzed later with the *Sequence Chart* tool in the IDE. The `results` directory in the project folder contains the `.elog` file. Double-clicking on it in the OMNeT++ IDE opens the Sequence Chart tool, and the event log tab at the bottom of the window. You will see a chart similar to Figure 3.4. You can also open separate output windows for ***tic*** and ***toc*** by right-clicking on their icons and choosing *Component log* from the menu.

- Observing simulation logs: When you run the simulation in the OMNeT++ runtime environment, the output similar to Figure 3.5 will appear in the log window. This is the simplest form of logging available in OMNET++.

3.5 Warm-up exercises with OMNET++

The first OMNET++ simulation that you just completed is adapted from the *Tictoc tutorial* available at `https://docs.omnetpp.org/tutorials/tictoc/`

Follow the complete *Tictoc tutorial*, which contains a series of simulation models numbered 1 through 16. The models are of increasing complexity. They start from the basics and introduce new OMNeT++ features or simulation techniques in each step. The additions are well commented, so the model sources can help you build up a good working knowledge of OMNeT++ in a short time.

♣ The End ♣