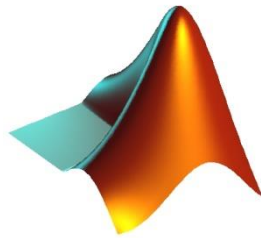


# ECEN 3714----Network Analysis

## Laboratory Manual

### **LAB 2: Introduction to MATLAB**



Oklahoma State University  
School of Electrical and Computer Engineering.

## 1. Objective

The purpose of this lab is to understand how to use MATLAB software to numerically evaluate a function or a matrix. MATLAB will be used in future classes and pre-lab works mainly for analyzing circuit frequency response.




## 2. Introduction to MATLAB Software

MATLAB is one of the interactive software systems for doing numerical computations. Several commercial software packages, including MATLAB and MATHEMATICA, are available for scientific computations needed in engineering and physical sciences. MATLAB uses highly respected algorithm that makes confident results.

Using MATLAB, the evaluation of circuit frequency response is made easy by performing just one or two commands. You can also build your own set of functions and excellent graphic facilities which makes your result easily understandable. It takes a great deal of practices to become a MATLAB expert, but you'll be on your way to getting comfortable with it and knowing that you can always grab a thick MATLAB manual for more complicated tasks.

### 2.1. Overview of the User Interface

MATLAB is started just like any other program, go to the search option of your computer, search for the MATLAB and click on it. The default MATLAB interface will then open on your screen (see Figure 2-1). Depending on the version, the user interface may be slightly different. As shown in the figure, the screen is divided into three main elements (actual display depends upon the version of MATLAB). These are:

-  File listing in the current directory
-  Command History Window
-  Command Window

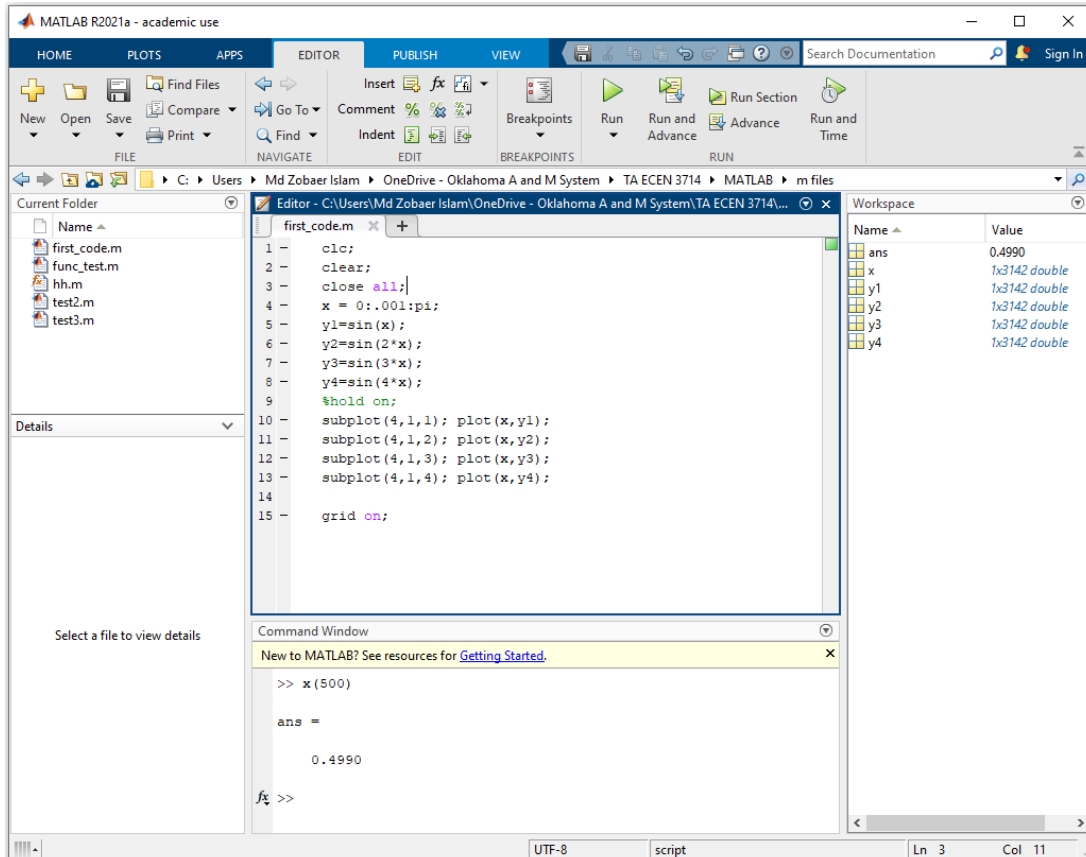


Figure 2-1

The standard mix of menus appears on the top of the MATLAB desktop that allows you to do things like file management and debugging of files you create. You will also notice a drop-down list on the upper right side of the desktop that allows you to select a directory to work in.

MATLAB works with three major types of windows:

- 1) **Command window**: It starts with `>>` sign, and you will write MATLAB commands here to execute.
- 2) **Figure window**: It will pop up when you will plot something using plotting related commands.
- 3) **Editor window**: This window is for writing and editing MATLAB programs known M-files. This window can be opened from Home → New Script or with shortcut command Ctrl + N.

## 2.2. Starting an M-file

In order to write programs like you can write in other programming languages, start a new M-file from the “File” menu.

- The command “clear all” clears all the variables. Write “**clear all**” at the beginning of your M-file.
- Write “**clc**” to clear the command window, “**close all**” to close all the open figure windows.
- Write the rest of the code / commands in the M-file. If you use semicolon after a command, then its result will not be prompted in the command window. If you do not use the semicolon, then you will see the result of that command e.g. value of the associated in the command window after you run the program.
- After you are done writing the program, save the file and “Run” the program from **Editor** → **Run** or use shortcut command **F5** to see the results in command window or figure window (if you plot something).
- You can add comments to your code using a % sign which will not be compiled. Anything after % sign will be treated as comments by MATLAB.

## 2.3. Basic Arithmetic

MATLAB uses a straightforward notation for basic arithmetic on scalars. The symbol + is used to add scalars, so `x=1+5` will give x the value 6. Similarly, MATLAB uses - for subtraction, \* for multiplication, / for division, and ^ for exponentiation. All of these work for two scalars. In addition, you can add, subtract, multiply or divide all the elements of a vector or matrix by a scalar. For example, if x is a matrix or vector, then `x+1` will add one to each element of x, and `x/2` will divide each element of x by 2. `X^2` will not square each element of x. We'll show you how to do that later. All of the basic operations (+, -, \*, /, and ^) are defined to work with complex scalars.

Another useful operator is the **colon**. You can use the colon to specify a range of numbers. Typing

```
>>x= 1:4
```

will return

```
x =
```

```
1      2      3      4
```

You can optionally give the colon a step size. For instance,

```
>>x=8:-1:5
```

will give

```
x =
```

```
8      7      6      5
```

and

```
>> x = 0:0.25: 1.25
```

will return

```
x =
```

```
0      0.25    0.5    0.75    1.0    1.25
```

The colon is a subtle and powerful operator, and we'll see more uses of it later.

## 2.4. Help

MATLAB has a fairly good help facility. The help function knows about all the commands listed in this manual. Typing help function will tell you the syntax for the function, i.e. what arguments it expects. It will also give you a short description of what the command does.

```
>> help
```

You can write “help function\_name” in the command window to see the description of that function. Example:

```
>> help sin
>> help plot
```

You can also use help in the menubar (“Search documentation”) or even from Mathworks website (<https://www.mathworks.com/help/matlab/>).

## 2.5. Basic Matrix Constructors and Operators

MATLAB has a variety of built-in functions to make it easier for you to construct vectors or matrices without having to enumerate all the elements.

The **ones** function will create a matrix whose elements are all ones. Typing ones(m,n) will create an m row by n column matrix of ones.

```
>> ones(3,4)
```

Gives

```
ans  1 1 1 1
      1 1 1 1
      1 1 1 1
```

To create a discrete-time signal named y assigned to a vector of 16 ones, you would type y = ones(1,16);. Also, giving ones a matrix as its only argument will cause it to return a matrix of ones the same size as the argument. This will not affect the original matrix you give as an argument, though. If x = [1 2 3 4; 0 9 3 8], typing ones(x) will create a matrix of ones that is two rows by four columns.

The **zeros** function is similar to the ones function. Typing zeros (m,n) will create an m-by-n matrix of zeros, and zeros(x) will create a two-by-four matrix of zeros, if x is defined the same way as above.

The **max** and **min** functions are used to find the largest and smallest values in a vector. If z = [1 2 -9 3 -3 -5],

```
>> max (z)
```

```
return
ans
      3
```

If you call **max** with a matrix as an argument, it will return a row vector where each element is the maximum value of each column of the input matrix. Max is also capable of returning a second value: the index of the maximum value in the vector. To get this, you assign the result of the call to max to be a two element vector instead of just a single variable. If z is defined as above, [a b] = max (z) will assign a to be 3, the maximum value of the vector, and b to be 4, the index of that value. The MATLAB function min is exactly parallel to max except that it returns the smallest value, so min(z) will return -9.

**Sum** and **prod** are two more useful functions for matrices. If z is a vector, sum(z) is the sum of all the elements of z. Similarly, prod (z) is the product of all the elements of z.

Often, it is useful to define a vector as a subset of a previously defined vector. This is another use of the colon operator. If we have z defined as in the min/max examples, z(2:5) will be the vector [2 -9 3 -3]. Again, remember that MATLAB indexes vectors such that the first element has indexed one, not zero.

The **size** function will return a two element vector giving the dimensions of the matrix it was called with. If x is a 4 row by 3 column vector, size(x) will return the vector [4 3]. You can also define the result to be two separate values as shown in the max example. If you type [m n] = size(x), m will be assigned to

the value 4, and n to the value 3. The **length** operator will return the length of a vector. If z is defined as 14 3 91, length (z) will return 3. Basically, length (z) is equivalent to max(size(z)).

## 2.6. Element wise Operations

We will often want to perform an operation on each element of a vector while doing a computation. For example, we may want to add two vectors by adding all of the corresponding elements. The addition (+) and subtraction (-) operators are defined to work on matrices as well as scalars.

For example, if

```
>> x = [1 2 3];
```

```
>> y = [5 6 2];
```

```
>> y+x
```

will return

Ans

[6 8 5]. Again, both addition and subtraction will work even if the elements are complex numbers.

Multiplying two matrices element by element is a little different. The \* symbol is defined as matrix multiplication when used on two matrices.

To specify element-wise multiplication, we use .\* So, using the x and y from above, x.\*y will give [5 12 6]. We can do exponentiation on a series similarly. Typing x.^2 will square each element of x, giving [1 4 9]. Finally, we can't use / to divide two matrices element wise, since / and \ are reserved for left and right matrix "division." So, we use the ./ function, which means that y./x will give [5 3 0.6666]. Again, all of these operations work for complex numbers.

## 2.7. Other operations

A complex number can be written as a+b\*i in MATLAB. The **abs** operator returns the magnitude of its argument. If applied to a vector, it returns a vector of the magnitudes of the elements. For instance, if x = [2 -4 3-4\*i -3], typing

```
>> y = abs(x)
```

will return

y=

2 4 5 3

The **angle** operator will return the phase angle of its operand in radians. The angle operator will also work element wise across a vector. Typing

```
>> phase = angle(x)
```

will give

phase =

0 3.1416 -0.9273 3.1416

The **sqrt** function is another commonly used MATLAB function. As you might well expect, it computes the square root of its argument. If its argument is a matrix or vector, it computes the square root of each argument. If we define x = [4 -9i 2-2\*i], then typing

```
>> y = sqrt(x)
```

gives

y=

2.0000 2.1213 - 2.1213i 1.5538 - 0.6436i

MATLAB also has operators for taking the real part, imaginary part, or complex conjugate of a complex number. These functions are **real**, **imag** and **conj**, respectively. They are defined to work element wise on any matrix or vector.

MATLAB includes several operators to round fractional numbers to integers. The **round** function rounds its argument to the nearest integer. The **fix** function rounds its argument to the nearest integer

towards zero, e.g. rounds "down" for positive numbers, and "up" for negative numbers. *ceil* rounds its argument to the nearest integer towards positive infinity, e.g. "up", and *floor* rounds its argument to the nearest integer towards negative infinity, e.g. "down." All of these commands are defined to work element wise on matrices and vectors. If you apply one of them to a complex number, it will round both the real and imaginary part in the manner indicated. Typing

```
>> ceil(3.1 + 2.4*i)
will return
ans =
    4.0000 + 3.0000i
```

MATLAB can also calculate the remainder of an integer division operation. If  $x = y * n + r$ , where  $n$  is an integer, then `rem(x,y)` is  $r$ .

The standard trigonometric operations are all defined as element-wise operators. The operators *sin*, *cos* and *tan* calculate the sine, cosine and tangent of their arguments. The arguments to these functions are angles in radians. Note that the functions are also defined on complex arguments, which can cause problems if you are not careful. For instance,  $\cos(x+iy) = \cos(x)\cosh(y) - i \sin(x)\sinh(y)$ . The inverse trig functions (*acos*, *asin* and *atan*) are also defined to operate element-wise across matrices. Again, these are defined on complex numbers, which can lead to problems for the incautious user. The arctangent is defined to return angles between  $\pi/2$  and  $-\pi/2$ .

MATLAB also includes functions for exponentials and logarithms. The *exp* operator computes  $e$  to the power of its argument. This works element-wise, and on complex numbers. So, to generate the complex exponential with a frequency of  $\pi/4$ , we could type

```
>> n = 0:7;
>> s = exp(i*(pi/4)*n)
s =
Columns 1 through 4
    1.0000    0.7071 + 0.7071 i    0.0000 + 1.0000i   -0.7071 + 0.7071 i
Columns 5 through 8
   -1.0000 + 0.0000i   -0.7071 i - 0.7071 i    0.0000 - 1.0000i    0.7071 - 0.7071 i
```

MATLAB also has natural and base-10 logarithms. The *log* function calculates natural logs, and *log10* calculates base-10 logs. Again, both operate element-wise for vectors. Both also come complete with the now-familiar caveat that they are defined for complex values, and that you should be careful about passing them complex arguments if you don't know how complex logs are defined.

## 2.8. Plotting Elementary Functions

Plotting is one of the most useful applications of a math package used on the computer, and MATLAB is no exception to this rule. Often we need to visualize functions that are too hard to graph "by hand" or to plot experimental or generated data. In this chapter we will introduce the commands and techniques used in MATLAB to accomplish these kinds of tasks.

### 2.8.1. Basic 2D Plotting

Let's start with the most basic type of plot we can create, the graph of a function of one variable. Plotting a function in MATLAB involves the following three steps:

1. Define the function
2. Specify the range of values over which to plot the function
3. Call the MATLAB `plot(x, y)` function

Let's plot the function  $y = \cos(x)$  over the range  $0 \leq x \leq 10$ . To start, we want to define this interval and tell MATLAB what increment to use. The interval is defined using square brackets `[]` that are filled in the following manner:

```
Start: interval: end
```

```
>> x = 0:1:10;  
>> y = cos(x);
```

Notice that we ended each line with *semicolons*. Remember, this suppresses MATLAB output. It's unlikely you would want MATLAB to spit out all the x values over the interval onto the screen, so we use the semicolon to prevent this. Now we can plot the function. This is done by entering the following command:

```
>> plot(x, y);
```

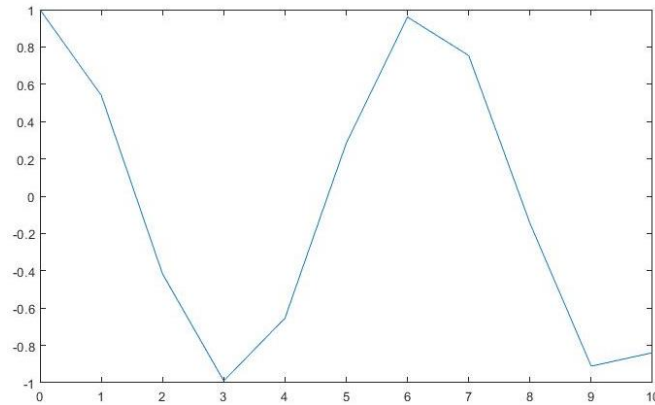


Figure 2-2

The plot of the function in this case is choppy. Let's try going the other way. We will make the increment ten times smaller than our original attempt, setting it to 0.01. Remember we need to redefine y again, therefore the commands we need to type in are:

```
>> x = 0:0.01:10;  
>> y = cos(x);  
>> plot(x, y);
```

Or in M-file, you can write below code, then save and run to get the below figure:

```
clc, clear, close all;  
x = 0:0.01:10;  
y = cos(x);  
plot(x, y);
```

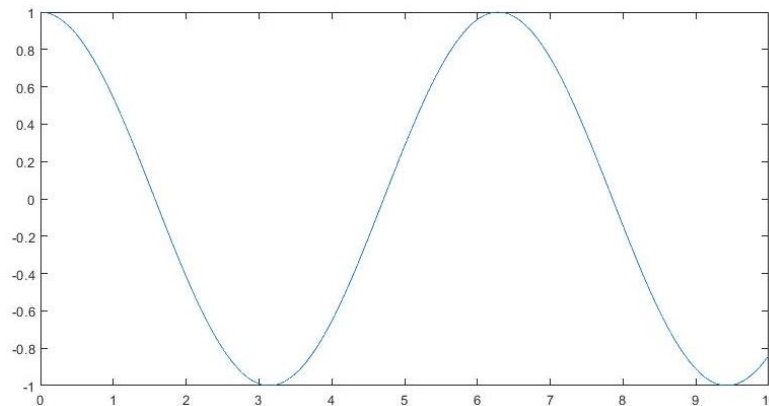


Figure 2-3

The first line of the above code-script is optional but useful. “clc” clears the command window; “clear” deletes all the variables from the current workspace and releases them from system memory and “close all” closes the figure windows that are currently open.

### 2.8.2. More 2D Plotting Options

Let's look at some other options we might consider with plots. If you're going to use a plot in a presentation or homework assignment, you might want to give the plot a title. You can use title('Name of Laboratory manual

the plot') command to do that. The title is printed just above the plot. Let's say that we were plotting some force data that happened to vary as  $f(t) = e^{-2t} \sin t$  for  $0 \leq t \leq 4$ , where  $t$  is time in seconds with data at intervals of 0.01 seconds.

```
>> t = 0:0.01:4;  
>> f = exp(-2*t).*sin(t);  
>> plot(t, f)
```

When generating a function, which is formed by the product of two or more other functions, make sure to tell MATLAB that we are doing element-wise multiplication between two matrices by including a `.'` character.

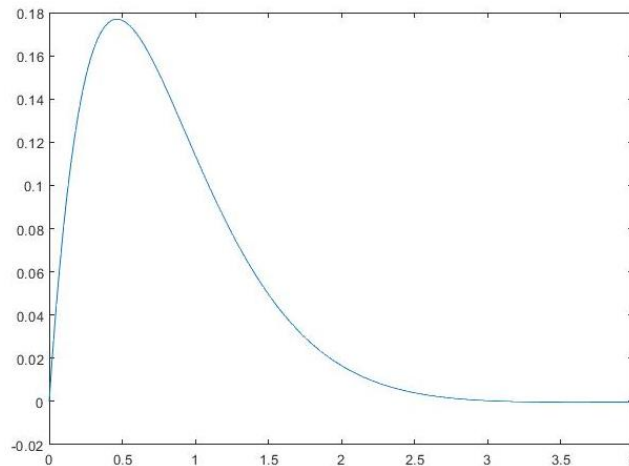


Figure 2-4

What are some other ways we can spruce up your basic two-dimensional plot? One way is to add a *grid* to the plot. This is done by adding the phrase `grid on` to your plot statement. For the next example, we will plot  $y = \text{square}(x)$  over the range  $-6 \leq x \leq 6$  with a grid display.

```
>> x = -6:0.01:6;  
>> y = square(x);  
>> Plot(x,y),grid on;
```

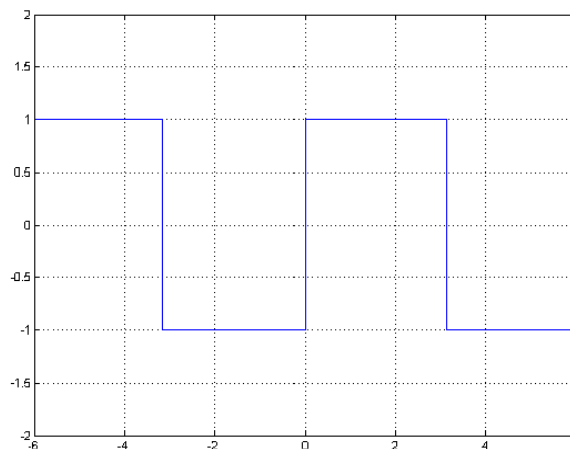


Figure 2-5

### 2.8.3. The Axis Commands:-

MATLAB allows you to adjust the axes used in two-dimensional plots in the following way. If we add `axis square` to the line containing the plot command, this will cause MATLAB to generate a square plot. If we type `axis equal`, then MATLAB will generate a plot that has the same scale factors and tick spacing on both axes.



Let's take a look at the following example  $y = \tanh(x)$ , which we plotted on the left of Figure 2-6. If you run this plot with axis square, you will get the same plot that we did using the default settings. But suppose that we typed the commands below, the graph will look like the one on the right of Figure 2-6:

```
>> x = -6:0.01:6;
>> y=tanh(x);
>> plot(x,y),grid on, axis equal
```

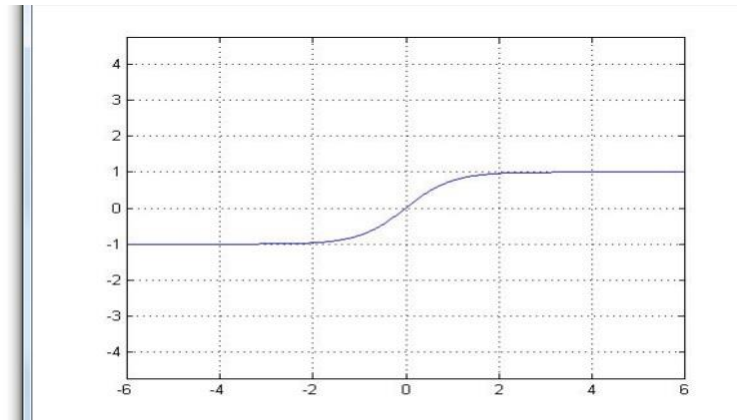
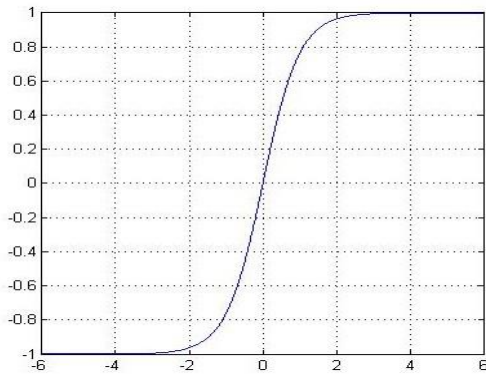


Figure 2-6

#### 2.8.4. Showing Multiple Functions on One Plot

In many cases, it is necessary to plot more than one curve on a single graph. The procedure used to do this in MATLAB is fairly straightforward. Let's start by showing two functions on the same graph. In this case let's plot the following two functions over  $0 \leq t \leq 5$ :

$$f(t) = e^{-t}$$

$$g(t) = e^{-2t}$$

We will differentiate between the two curves by plotting  $g$  with a dashed line. Following the usual procedure, we first define our interval:

```
>> t = 0:0.01:5;
```

Next, we define the two functions:

```
>> f = exp(-t);
```

```
>> g = exp(-2*t);
```

To plot multiple functions, we simply call the `plot(x, y)` command with multiple pairs  $x, y$  defining the independent and dependent variables used in the plot in pairs.

```
>> plot(t, f, t, '--');
```

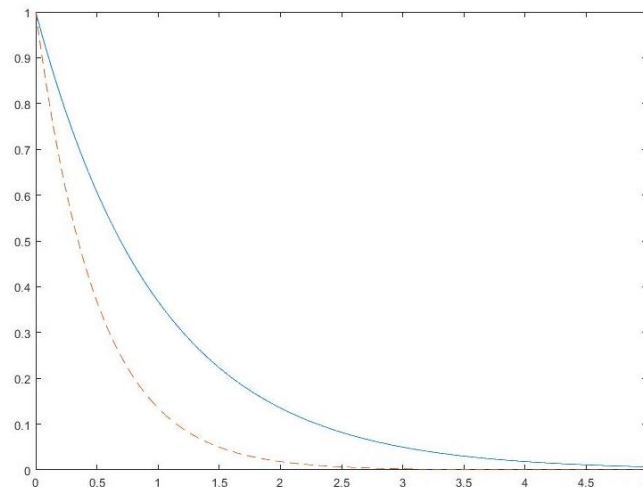


Figure 2-7

MATLAB has four basic line types that can be defined in a plot. These are, along with the character strings, used to define them in the plot command:

- ✚ Solid line '-'
- ✚ Dashed line '--'
- ✚ Dash-dot line '-.'
- ✚ Dotted line ':'

Let's generate the same graph with a dotted line. The command is  
`>>plot(t, f, ':', t, g, '--')`

### 2.8.5. Setting Colors and Linewidths

The color of each curve can be set automatically by MATLAB or we can manually select which color we want. Now we will generate the plot representing y with a red curve and z with a blue curve. The command looks like this:

`>> plot(x, y, 'r', x, z, 'b')`

Color	Specifier
White	W
Black	K
Blue	B
red	R
Cyan	C
Green	G
magnate	M
yellow	Y

The thickness of the curve can be set by 'linewidth' inside plot command. You can make the curves thicker for better visibility by using below command:

`>> plot(x,y,x,z, 'linewidth', '3')`

### 2.8.6. Adding Legends, Axes Labels and Titles

A professionally done plot has titles, axes labeled and often a legend that lets the reader know which curve is what. In the next example, let's suppose that we are going to plot two potential energy functions that are defined in terms of the hyperbolic trig functions  $\sinh(x)$  and  $\cosh(x)$  for  $0 \leq x \leq 2$ .

Xlabel and ylabel functions are used to label the axes. 'title' is used to set title. The 'legend' function includes a text string enclosed in single quotes for each curve you want to label. Even you can change the font sizes of all these texts using "fontsize" as shown in below code snippet, written in m-file.

```
clc, clear, close all;
x = 0:0.01:2;
y = sinh(x);
z = cosh(x);
plot(x,y,x,z, '-.', 'linewidth', 3);
xlabel('x', 'fontsize', 25);
ylabel('Potential', 'fontsize', 25);
title('Potential Energy Functions', 'fontsize', 25);
legend('sinh(x)', 'cosh(x)', 'fontsize', 25);
```

The code generates below figure:

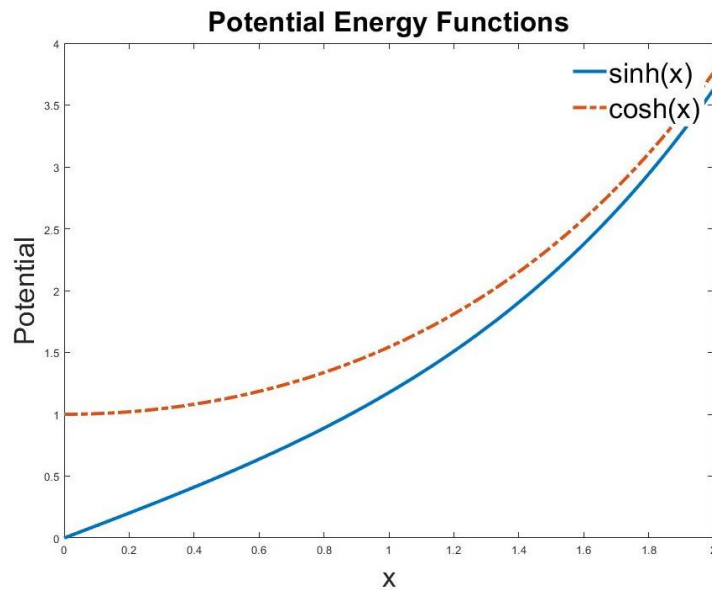


Figure 2-8

### 2.8.7. Setting Axis Scales:-

Let's take another look at the axis command and see how to set the plot range. This is done by calling **axis** in the following way:

```
axis ( [xmin xmax ymin ymax] )
```

Suppose that we want to generate a plot of  $y = \sin(2x + 3)$  for  $0 \leq x \leq 5$ . We might consider that the function ranges over  $-3 \leq y \leq 3$ . We can set the y axis to only show these values by using the following sequence of commands:

```
>> x = 0:0.01:5;
>> y = sin(2*x + 3);
>> plot(x,y), axis([0 5 -3 3])
```

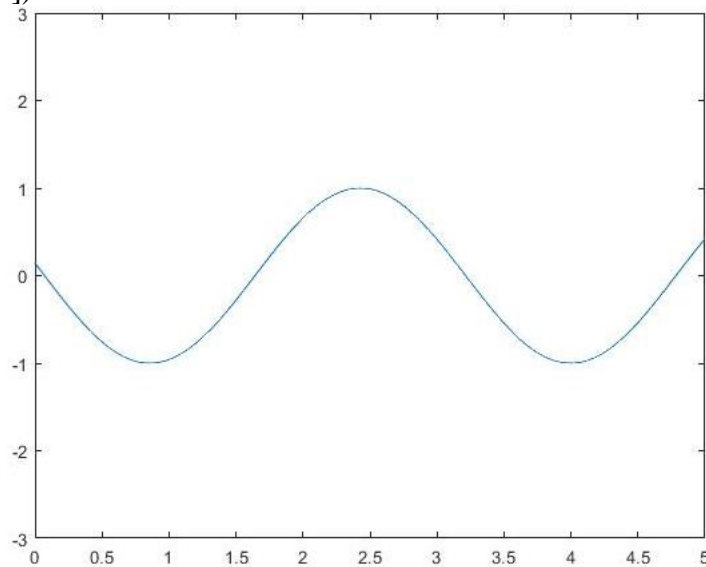


Figure 2-9

### 2.8.8. Subplots

A subplot is one member of an array of plots that appears in the same figure. The **subplot** command is called using the syntax subplot(m, n, p). Here m and n tell MATLAB to generate a plot array with m rows and n columns. Then we use p to tell MATLAB where to put the particular plot we have generated. Try below code snippet in M-file:

```

clc, clear, close all;
x = 0:0.01:1;
subplot(221), plot(x,sin(3*pi*x)), grid on;
xlabel('x'), ylabel('sin(3*pi*x)'), title('Plot 1');
subplot(222), plot(x,cos(3*pi*x)), grid on;
xlabel('x'), ylabel('cos(3*pi*x)'), title('Plot 2');
subplot(223), plot(x,sin(6*pi*x)), grid on;
xlabel('x'), ylabel('sin(6*pi*x)'), title('Plot 3');
subplot(224), plot(x,cos(6*pi*x)), grid on;
xlabel('x'), ylabel('cos(6*pi*x)'), title('Plot 4');

```

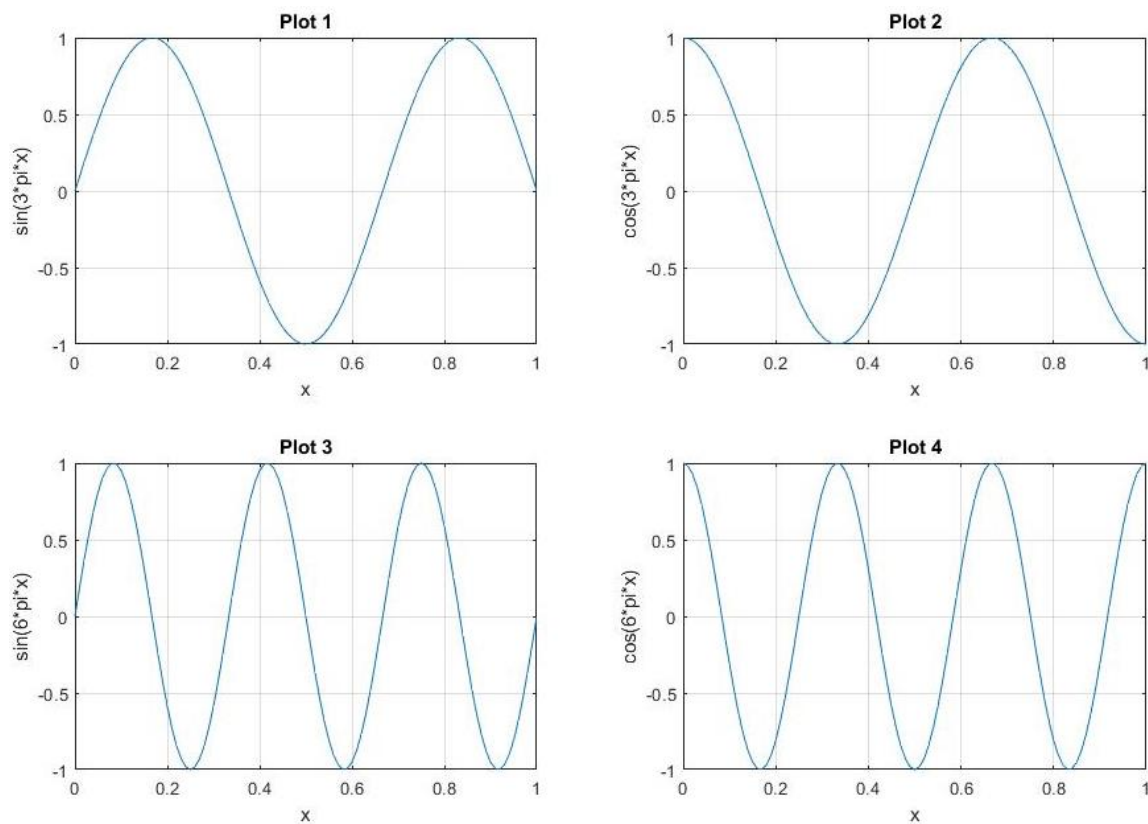


Figure 2-10

### 3. Laboratory Assignments

Complete below assignments individually and write report accordingly. Try to write MATLAB codes in M-files because .m files are more convenient than command-line. Each of the curves in the same plot should be distinguishable by color, line-marks, legends etc.

#### Assignment 1:

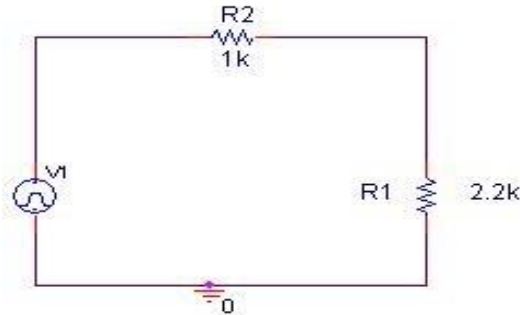


Figure 3.1

From the circuit shown in Fig. 3.1, It is straightforward to reach  $V_{R1} = V_1 \frac{R_1}{R_1 + R_2}$

- Find out how to represent or produce a square wave of 100Hz having a peak-to-peak voltage of 20V (-10V to 10V) in MATLAB. Generate a MATLAB Code to calculate the voltage across the Resistor  $R_1$  using the equation above. If the  $V_1$  is the square wave you generated, show  $V_1$  and  $V_{R1}$  in the same graph.
- Find out how to represent or produce a sine wave of 100Hz having a peak-to-peak voltage of 20V (-10V to 10V) in MATLAB. Assign  $V_1$  as this sine wave, and plot  $V_1$  and  $V_{R1}$  in one graph.

#### Assignment 2:

- When a charged capacitor is connected to a resistor, the absolute value of the voltage across the capacitor changes as  $V_C(t) = V_1 e^{-\frac{t}{RC}}$ . Given that  $R=1k\Omega$ ,  $C=500nF$ ,  $V_1=10V$ , plot  $V_1$  and  $V_C(t)$  in the same graph. You should choose a reasonable duration of the time to clearly show the exponential decay pattern.
- When a capacitor with a zero initial voltage is charged by a voltage source through a resistor that is in series with the capacitor, the absolute value of the voltage across the capacitor changes as  $V_C(t) = V_1(1 - e^{-\frac{t}{RC}})$ , where  $V_1$  is the voltage of the source. Given that  $R=1k\Omega$ ,  $C=500nF$ ,  $V_1=10V$ , plot  $V_1$  and  $V_C(t)$  in one graph.
- Plot  $V_1=10V$ , the  $V_C(t)$  obtained in (a) --- denoted as  $V_{C1}$ , and the  $V_C(t)$  obtained in (b) --- denoted as  $V_{C2}$ , in one graph.
- Now let's use Fig 3.2 and assume that the capacitor has no initial charge. At  $t=0$  the  $V_1$  of 10V is ON and it lasts for 5ms, and at  $t=5ms$  the  $V_1$  becomes zero and it maintains at zero for 5ms. Plot  $V_1$ , voltage across the capacitor  $V_{12}$  (i.e. polarity is defined as 1-higher, 2-lower), voltage across the resistor  $V_2$ , for this 10ms duration in the same graph.

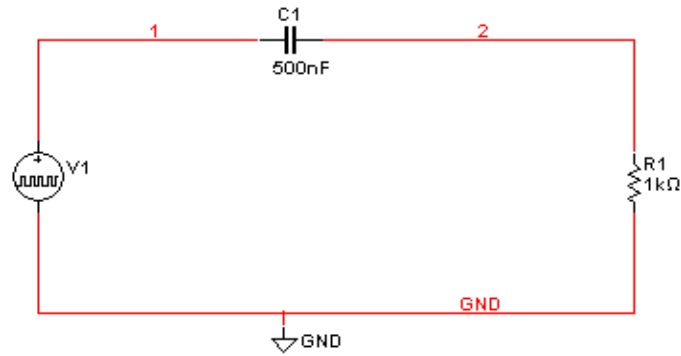


Figure 3.2

### Assignment 3:

- When a charged inductor is connected to a resistor, the absolute value of the current flowing through the inductor changes as  $I_L(t) = I_1 e^{-\frac{t}{L/R}}$ . Given that  $R=1k\Omega$ ,  $L=5H$ ,  $I_1=10A$ , plot  $I_1$  and  $I_L(t)$  in the same graph. You should choose a reasonable duration of the time to clearly show the exponential decay pattern.
- When an inductor with zero initial current is charged by a current source through a resistor that is in parallel with the inductor, the absolute value of the current through the inductor changes as  $I_L(t) = I_1(1 - e^{-\frac{t}{L/R}})$ , where  $I_1$  is the current of the source. Given that  $R=1k\Omega$ ,  $L=5H$ ,  $I_1=10A$ , plot  $I_1$  and  $I_L(t)$  in one graph.
- Plot  $I_1=10A$ , the  $I_L(t)$  obtained in (a) --- denoted as  $I_{L1}$ , and the  $I_L(t)$  obtained in (b) --- denoted as  $I_{L2}$ , in one graph.
- Now let's use Fig 3.3 and assume that the inductor has no initial current. At  $t=0$ , the  $V_1$  of 10V is ON and it lasts for 10ms, and at  $t=10ms$  the  $V_1$  becomes zero and it maintains at zero for 10ms. **During the ON state of input voltage**, the formula for voltage across resistor is  $V_2(t) = V_{peak}[1 - \exp(-t/\tau)]$  and voltage across inductor is  $V_{L2}(t) = V_1 \exp(-t/\tau)$ , where  $\tau = L/R$ . Use MATLAB to plot  $V_1$ ,  $V_{L2}$  and  $V_2$  in one graph **for this 20ms duration**.

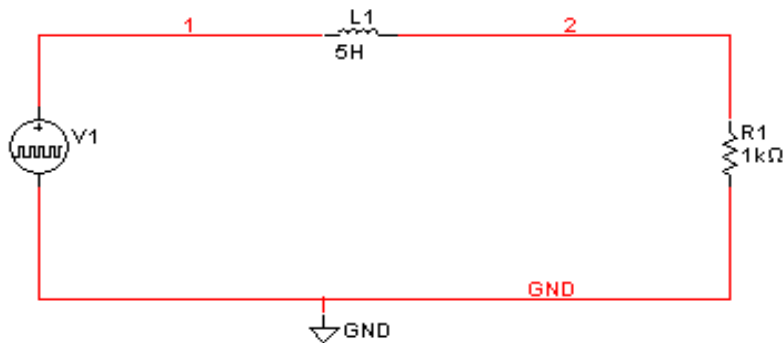


Figure 3.3

#### 3.1. Requirements for laboratory report

The lab report is to be completed individually, and due by the same lab session at the following week. The following contents constitute the minimum amount of materials that should be included in your report. There will be no pre-lab for this lab. You can provide your MATLAB codes in the report, but it is not mandatory.

- (a) **Cover Page:** Minus 5% points for not having one
- (b) **Introduction:** Describe the objectives of this lab, in your own words. (5% points)
- (c) **Assignments:**

**Assignment 1: (10%)**

- 1) The single graph showing  $V_1$  and  $V_{R1}$  for a square wave  $V_1$  obtained in (a). ----- 5%
- 2) The single graph showing  $V_1$  and  $V_{R1}$  for a sinusoidal wave  $V_1$  obtained in (b). -----5%

**Assignment 2: (40%)**

- 1) The single graph showing  $V_1$  and  $V_C(t)$  as obtained in (a). ----- 5%
- 2) The single graph showing  $V_1$  and  $V_C(t)$  as obtained in (b). -----5%
- 3) The single graph showing  $V_1$ ,  $V_{C1}$ , and  $V_{C2}$ , as obtained in (c). -----10%
- 4) The single graph showing  $V_1$ , voltage  $V_{12}$  across the capacitor, voltage  $V_2$  across the resistor, as obtained in (d). -----20%

**Assignment 3: (40%)**

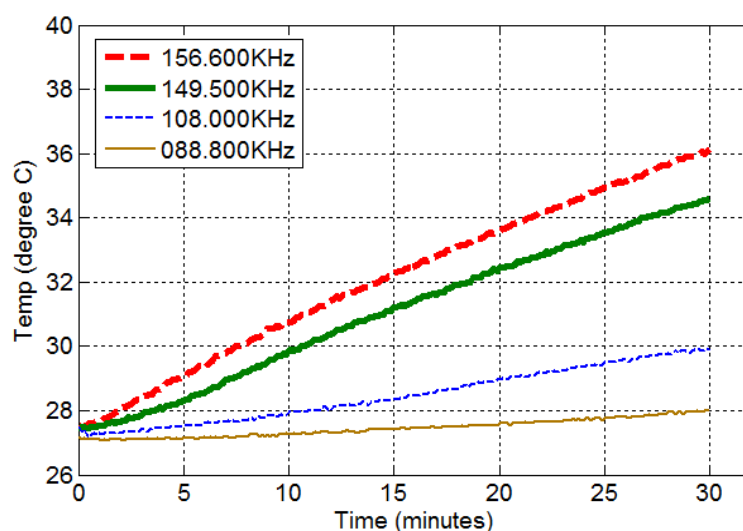
- 1) The single graph showing  $I_1$  and  $I_L(t)$  as obtained in (a). ----- 5%
- 2) The single graph showing  $I_1$  and  $I_L(t)$  as obtained in (b). ----- 5%
- 3) The single graph showing  $I_1$ ,  $I_{L1}$ , and  $I_{L2}$  as obtained in (c). ----- 10%
- 4) The single graph showing  $V_1$ , voltage  $V_{12}$  across the inductor, voltage  $V_2$  across the resistor as obtained in (d) ----- 20%

(d) **Discussion:** Discuss your observations and learnings from this lab. (5% points)

(e) **Reference:** If any.

**Caution:** All plots must have complete and comprehensive information including legend, axes, units, etc

**Example of a figure with comprehensive legend:** The following figure utilizes different line styles as well as line marks to represent different data, and a legend-box to detail the line-style and line-mark. The figure also contains legends to denote the variable and the unit of the variable for X, Y respectively. It is expected that the figure you presented in your report has similar level of detailed notation.



**Figure 3.4** The temperature rise of magnetic nanoparticle under an alternating magnetic field