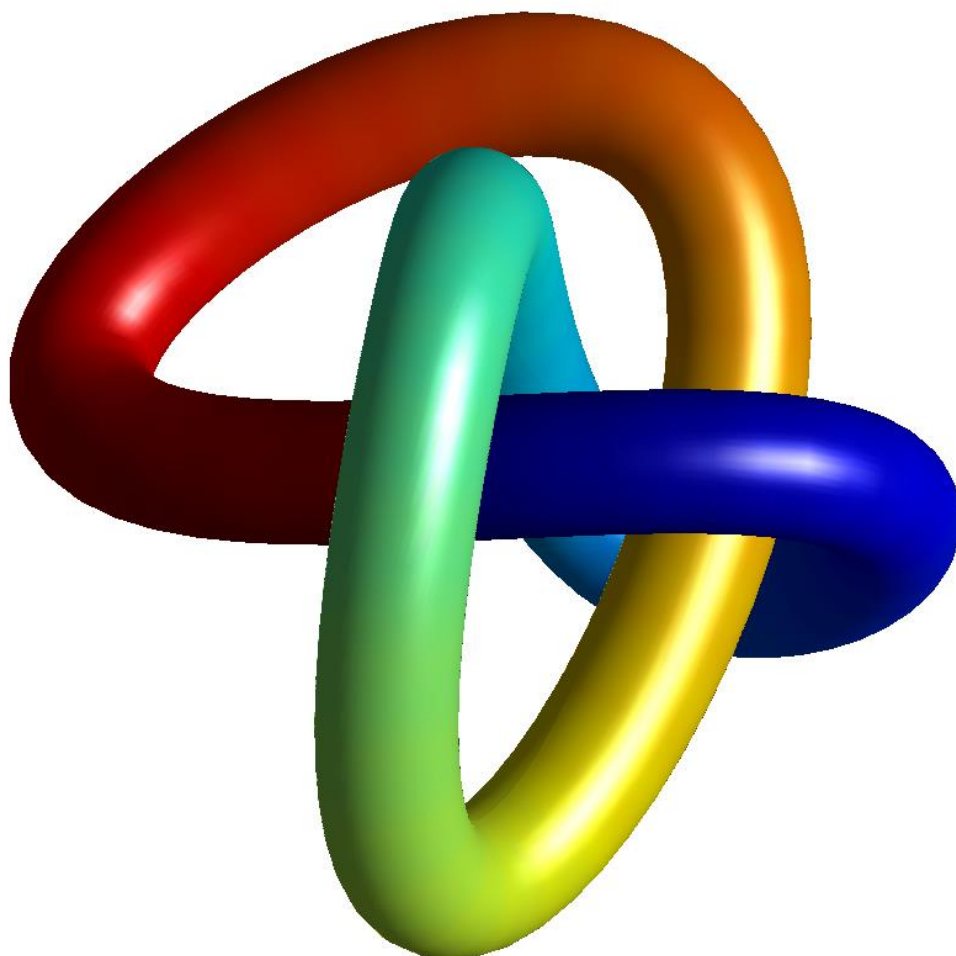**UNIVERSITY OF TORONTO**
FACULTY OF APPLIED SCIENCE AND ENGINEERING
Department of Electrical and Computer Engineering

ECE221H1S
ELECTRIC AND MAGNETIC FIELDS

Lab #1:
Introduction to MATLAB Part 1
Working With Arrays, and
Two-Dimensional Plotting

INTRODUCTION

One of the purposes of the computer labs in this course is to allow you to learn how to use MATLAB, which is a very powerful tool for mathematical analysis. It is widely used in both industry and academia due to its simple programming environment and flexibility. Through these computer lab experiences you will learn how to:

1. Use MATLAB's built-in functions to perform basic computations using real and complex numbers,
2. Perform basic computations using vectors and matrices,
3. Create two-dimensional and three-dimensional plots of functions, with proper labeling and formatting,
4. Create and run scripts and functions,
5. Work with variables and save and load data and figure files,
6. Use MATLAB to solve electric and magnetic field problems *numerically* rather than *analytically*. For example,
   a. Calculating the electric field due to a continuous charge distribution, and
   b. Visualizing electric and magnetic fields through two- and three-dimensional plots,
7. Use MATLAB for many other purposes, which might be helpful in your other courses. For example,
   a. Determining graphical solutions of transcendental equations,
   b. Creating Bode plots to visualize the frequency response of an electric circuit,
   c. Visualizing the frequency spectrum of a time-domain signal using stem plots, and
   d. Calculating discrete-time Fast Fourier Transforms (FFTs).

This introduction briefly describes the essentials of the MATLAB programming environment and language. The first two labs in this course will help you to understand the fundamentals of how to use MATLAB.

Getting Started

The MATLAB tutorial and the computer labs will be completed in the faculty's computer labs. You do not need to purchase a copy of MATLAB, as this program is available in both the ECE computer labs (Windows – BA3128, Linux – GB251 and SF2102) and the ECF computer labs (Windows – WB316, GB144, GB150, and SF1106, Linux – SF1106, SF1012, and SF1013). **If you have trouble logging into your account on the ECE Workstation Labs machines (GB251 and SF2102), please check with Tim Trant in GB249**.

If you do have access to MATLAB elsewhere, you should be able to complete these labs with any basic version of MATLAB 7 (R2011, R2013, etc.). As far as we are concerned, there is little difference in these versions, i.e., R2013a is very much the same as R2011, R2010, etc. As well, these labs do not require any of the special toolboxes for MATLAB, so the basic version is sufficient.

To run MATLAB in the ECE Workstation Labs (Linux – GB251 and SF2102), open a terminal window and type "matlab".

In the ECF computer labs, MATLAB is accessible through the Start menu (Windows), or under the "ECF Applications" menu (Linux).

MATLAB can also be accessed off-campus through ECF's **Remote Desktop** at:
https://ssl.ecf.utoronto.ca/ecf/services/rd

Introduction to MATLAB

MATLAB is a software package which is very versatile and can be used in many applications. Many engineering disciplines take advantage of the MATLAB software for its capabilities. It is just one of many computer software tools that will become useful to a student in their engineering career.

When you first open MATLAB, you will see 3 windows (see figure below): the *Workspace* window, the *Command Window* and the *Command History* window.
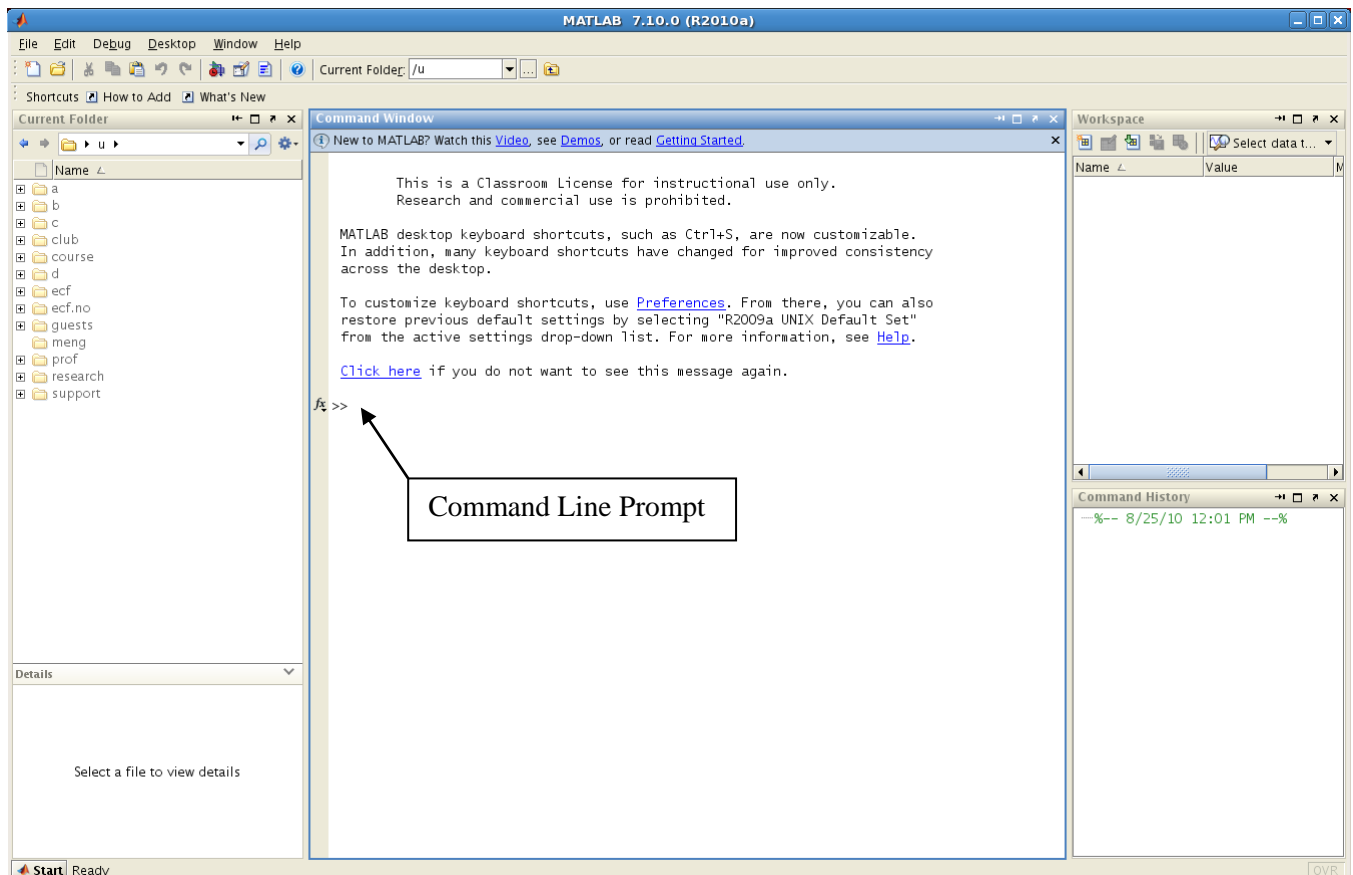
The *Workspace* window, located at the top-right corner, provides an inventory of all the items in the workspace that are currently defined, either by assignment or calculation in the *Command Window* or by importation with a *load* or similar command from the MATLAB command line prompt. These items consist of the set of arrays (including 1×1 scalars) whose elements are variables or constants and which have been constructed or loaded during the current MATLAB session and have remained stored in memory. The *Workspace* window shows the name of each variable and the relevant information about this variable.

The *Command Window* is in the center of the screen and it is where all the commands are executed and outputs displayed. The MATLAB command line prompt on the command window consists of two adjacent right angle brackets, i.e. **>>**. Results of command operations will also be displayed in this window unless the command line is terminated by a semicolon, ";", in which case the display of results is suppressed. If a command or script specified on the command line is questionable or cannot be executed because of invalid syntax, undefined variables, etc., a diagnostic message will be displayed in the

Command window. The current value of any saved variable is also displayed in this window if its name is entered at a prompt.

Finally, the *Command History* window, at the lower right in the default desktop, contains a log of commands that have been executed within the *Command* window. This is a convenient feature for tracking when developing or debugging programs or to confirm that commands were executed in a particular sequence during a multi-step calculation from the command line.

The Help Window

Separate from the main desktop layout is a Help desktop with its own layout. This utility can be launched by selecting *Help → Product Help* from the Help pull down menu. This Help desktop has a right side which contains links to help with functions, help with graphics, and tutorial type documentation. The left side has various tabs that can be brought to the foreground for navigating by table of contents, by indexed keywords, or by a search on a particular string.

Help can also be accessed by typing `help` at the command line prompt.

The Figure Window

There is a Figure window that floats independently from the main desktop. If not already present, it is launched when command execution results in graphical output, such as the `figure` command. From the *Edit* menu on the main toolbar, there are selections for editing figure properties, axis properties, and properties of objects within figures. On the Tools menu of the main toolbar there are selections for further manipulation such as zooming and perspective rotation.

MATLAB *File Types*

There are a variety of file types in MATLAB, but we will focus on the three which have the extensions: "**.m**", "**.mat**" and "**.fig**". Type `help fileformats` for more detailed information.

.m: The MATLAB program files. These contain all the MATLAB code that is needed to run a particular script or function. These are standard ASCII text files and thus can be edited in any basic text editor (MATLAB has such an editor available.) These are discussed further in a later module.

.mat: These are binary data files which are created when you save data in MATLAB by running the `save` command. The data contained within a **.mat** file can be loaded into MATLAB by running the `load` command (discussed further in a later module.)

.fig: These are binary figure files that can be loaded into MATLAB as a new figure (discussed further in a later module.)

Running a program

To run a program, the working directory from which the files are to be used must be selected in the "*Current Directory*" bar near the top of the screen (also shown in the *Current Folder* window located in the left-hand corner). The directory should contain the files that are needed to run the program, including the **.m** file.

Note that **.m** files may be also run by typing the file name without its extension at the command line.

Reference Information

For more details about the MATLAB programming environment and the description of MATLAB commands it is easiest to consult the built-in help feature. In addition, the following online resources may be helpful:

a) *Online MATLAB tutorials at Mathworks.com (producer of MATLAB)*:
   http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

   This contains a wide variety of videos, examples, interactive tutorials, and links to external resources.

b) *The official MATLAB Primer*:
   http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf

c) *The official online documentation*:
   http://www.mathworks.com/help/matlab/index.html

LAB #1

The purpose of this lab is to allow you to become comfortable with the basic interface and syntax of MATLAB. In this lab, you will learn how to define and manipulate vectors and arrays, as well as how to generate a number of different kinds of two-dimensional plots.

PREPARATION - Individual

Required to do before you come into the lab

1) Carefully read through the Introduction notes above, and the in-lab exercises described below.
2) Watch the 5 minute *Getting Started with MATLAB* video demonstration which is available at:
   a. http://www.mathworks.com/videos/getting-started-with-matlab-68985.html, *or*
   b. If you are running MATLAB, it can be accessed by selecting Demos from the MATLAB Start button, ⬥ Start.
3) Watch the video demonstration *Using Basic Plotting Functions (5:30),* available at:
   a. http://www.mathworks.com/videos/using-basic-plotting-functions-69018.html, *or*
   b. If you are running MATLAB, it can be accessed by selecting Demos from the MATLAB Start button, ⬥ Start.

IN-LAB WORK – Groups of Two

Working With Arrays (Matrices and Vectors)

Input via the command-line

Get yourself setup at one of the computers in GB251 or SF2102 and run MATLAB. As you read through the following examples, enter these in MATLAB to become comfortable with the basic interface.

Commands in MATLAB are executed by pressing **Enter** or **Return**, with the output being displayed on the screen immediately. As an example, try

```
>> 3 + 7.5
```

This will give a result in the command window as follows

```
ans =
 10.5000
```

The *up* and *down* arrow keys allow you to recall previous commands, which can be very helpful for making small changes to commands you have already entered. As well, the *Page Up* and *Page Down* keys allow you to navigate through the command window. **_Press the up arrow to recall the command_**: `3 + 7.5`. Edit it to evaluate `3 * 7.5`.

Previous commands are also accessible through the *Command History* window. **_Try to drag and drop_** your one of your earlier commands now. **Note:** You can clear your workspace from all variables by using the `clear` command.

The result of the last performed computation is ascribed to the variable `ans`, which is an example of a MATLAB built-in variable. It can be used in the next command. For instance,

```
>> 14/4
ans = 3.5000

>> ans *2
ans =
7.0000
```

You can also define your own variables. *Type* in the following commands to create the variables `a` and `b`:

```
>> a = 14/4
a =
3.5000

>> b = a+1
b =
4.5000
```

The command `whos` displays a list of variables, their sizes, and their data class. *Try it now*.

The entry of very large and very small numbers can be simplified using scientific notation. For example, *enter the numbers* $\varepsilon_0 = 0.000000000008854 = 8.854 \times 10^{-12}$, and $\mu_0 = 4\pi \times 10^{-7}$, by typing:

```
>> epsilon0 = 8.854e-12
```

and

```
>> mu0 = 4*pi*1e-7
```

Do the following exercises:

1. *Evaluate the following expressions* using MATLAB. Where useful, use the up arrow to recall your previous expression to edit for the next calculation. Fill in the table as you go. Verify that these make reasonable sense. Are they exactly (or approximately) what you might expect?

Expression	Result from MATLAB
`8 * 5 + 3`	
`8 * (5 + 3)`	
`10 / 2 / 5 - 3 + 2 * 4`	
`3 ^ 2 ^ 3`	
`exp(j*pi/4)`	
`c = a^39 where a = 3`	
`1/sqrt(epsilon0*mu0)` where epsilon0 and mu0 are defined as above.	

2. When the command is followed by a semicolon ';', the output is suppressed. ***Check the difference between the following expressions***:

```
>> 3 + 7.5    % without semicolon
>> 3 + 7.5;   % with semicolon
```

Note that the last stored answer can always be viewed by typing `ans` in the command window. Also note that the text following the `%` sign is ignored by MATLAB. It is useful for inserting comments, explanations or reminders.

Working with Matrices

A matrix of size $n \times k$ is a rectangular array of numbers having n rows and k columns. Row and column vectors are special types of matrices, with size of $n \times 1$, and $1 \times k$, respectively. Commas or spaces are used to separate elements in a row, and semicolons are used to separate individual rows. For example, ***define the matrix***

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

by typing

```
>> A = [1 2 3; 4 5 6; 7 8 9]   % row by row input
A =
1 2 3
4 5 6
7 8 9
```

Try this example of how to specify a matrix in MATLAB:

```
>> A2 = [1:4; -1:2:5]
A2 =
 1 2 3 4
-1 1 3 5
```

Notice how the colon ':' is used to specify a range of numbers as elements. `1:4` simply means from 1 to 4 in steps of 1; `-1:2:5` means from -1 to 5 in increments of 2.

Transposing a matrix can be achieved with a simple command, where transposing interchanges rows with the corresponding columns, as shown in the following example:

```
>> A2
A2 =
 1 2 3 4
-1 1 3 5

>> A2'   % transpose of A2
ans =
1 -1
2  1
3  3
4  5
```

Standard matrix multiplication is done with the `*` command, so the calculation given by:

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

would be executed in MATLAB by typing:

```
>> [1 2;3 4]*[2 0;1 3]
ans =
 4  6
10 12
```

While the "dot product", or *element by element multiplication*, is done by using the `.*` command. For example, try

```
>> [1 2;3 4].*[2 0;1 3]
ans =
 2  0
 3 12
```

Try these examples:

Expression	Result from MATLAB
`A2*A2'`	
`A2.*A2'`	
`A2.*A2`	

Building Matrices and Extracting Parts of Matrices

It is often necessary to build a larger matrix from a smaller one; for example

```
>> x = [4; -1], y = [-1 3]
x =
 4
-1
y =
-1 3
```

```
>> z = [x y']  % z consists of the columns x and y'
z =
 4 -1
-1  3
```

Right after the above example, try

```
>> z^2

>> z.^2
```

What is the difference between the two results? _____

_____

Next type in

```
>> z = x + y'

>> z^2 % does this work?

>> z.^2 % what about this?
```

What do you get as a result now? Why does one work and not the other? The `size` command may

help you answer these questions. _____

_____

Specific elements of a matrix can also be accessed. For example, to display the 2nd element of the first
row of the matrix A (as defined above), you would type:
```
>> A(1,2)
ans =
        2
```
$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and entire rows, or columns, are displayed through the use of the colon operator:
```
>> A(1,:)
ans =
        1    2    3
```

```
>> A(:,2)
ans =
        2
        5
        8
```

Partial rows, or columns, can be displayed by limiting the range:
```
>> A(1:2,2)
ans =
        2
        5
```

We can also make use of the colon operator to assign values to a part of a matrix:
```
>> A(1:2,1) = [1;1]
A =
        1    2    3
        1    5    6
        7    8    9
```

Click on `Help` in the top menu and search for `exp`, `for`, and `linspace` to get more information about these commands.

Do the following exercise:

Given the vectors (or row matrices) x = [1 3 7], y = [2 4 2] and the matrices A = [3 1 6; 5 2 7],   B = [1 4; 7 8; 2 2], determine which of the following statements will correctly execute (and if not, try to understand why) and provide the result or the explanation of why it doesn't work:

Expression	Result from MATLAB or Explanation
`B * A`	
`A * B`	
`2 * B`	
`B(:,2)`	
`A.^2`	
`exp(B(1,1))`	
`cos(pi*A)`	
`sin((pi/2)*A(1,2))`	
`exp(j*pi/4*B)`	

Applications – Arrays

Solving sets of simultaneous equations

In Electric Circuit Analysis you often need to solve sets of simultaneous equations which result from the basic techniques of circuit analysis (nodal and mesh analysis). Doing this by hand can be a time consuming process, but in MATLAB the solutions to these equations can be easily found.

This is only useful if we are dealing with cases where the coefficients of the equations are numbers. In cases where the coefficients contain one or more variables we must know how to solve this by hand. This might occur in a design-type problem where the value of a resistor is not known and the purpose of the problem is to determine the correct resistor value for a desired circuit operation.

To start, consider the set of equations given by:

$$5I_1 - 6I_2 = 0$$
$$-10I_1 + 14I_2 = 2$$

This can be converted to matrix form as:

$$\begin{bmatrix} 5 & -6 \\ -10 & 14 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

With the solution given by:

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 5 & -6 \\ -10 & 14 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1 \end{bmatrix}$$

In MATLAB this can be solved using the `inv` command (see `help inv`):
```
>> inv([5 -6;-10 14])*[0;2]

ans =

   1.2000
   1.0000
```

Note, that the matrix multiplier, `*`, is used rather than the dot multipler, `.*`, since we want real matrix multiplication rather than element-by-element multiplication.

Use MATLAB to solve the following set of equations:

$$9V_1 - 2V_2 - 2V_3 = 4$$
$$-2V_1 + 10V_2 - 4V_3 - V_4 - V_5 = 6$$
$$-2V_1 - 4V_2 + 9V_3 = -6$$
$$-V_2 + 8V_4 - 3V_5 = 0$$
$$-V_2 - 3V_4 + 4V_5 = -6$$

Make sure to discuss your final results with your TA before you leave the lab.

Plotting In MATLAB

Two-Dimensional Plots

The `plot` command can be used to produce a graphical display of a set of data points. For a vector (or row matrix) y, the command `plot(y)` draws the points [1, y(1)], [2, y(2)], . . ., [n, y(n)] and connects these points with straight lines. The command `plot(x,y)` does the same for the points [x(1), y(1)], [x(2), y(2)], . . . , [x(n), y(n)]. Note that x and y must be both either row or column vectors of the same length (i.e. must have the same number of elements).

Type the following commands *after predicting the results*:

```
>> x = linspace(0,10,100);
>> y = 2.^x;
>> plot(x)
>> plot(x,y)   % to get a graphic representation
```

More than one plot can be made in the same figure by specifying multiple arguments in the `plot` command. Try:

```
>> clf       % This clears the current figure window
>> plot(x,2*x.^2,x,4*x.^2)
```

Or, the `hold` command can be used. Try:

```
>> figure   % This opens up a new figure window
>> plot(x,2*x.^2)
>> hold
>> plot(x,4*x.^2)
```

Multiple plots can also be created on different axes in the same window, using the `subplot` command:

```
>> close all   % This closes all open figure windows
>> figure
>> subplot(2,1,1) % Breaks the figure window into two axes (top & bottom)
>> plot(x,2*x.^2) % Plots the function in the top set of axes
>> subplot(2,1,2) % Switches the current axes to the bottom set
>> plot(x,4*x.^2) % Plots the function in the bottom set of axes
```

Often it is useful to plot a function on a semilog or a log-log set of axes. For example, try the following commands:
```
>> figure
>> subplot(3,1,1)
>> plot(x,x.^2)
>> subplot(3,1,2) % Switches the current axes to the middle set
>> semilogx(x,x.^2) % Plots the function with the x-axis as a log scale
>> subplot(3,1,3) % Switches the current axes to the bottom set
>> loglog(x,x.^2) % Plots the function with both axes set to a log scale
```

Do the following exercises:

1. Create *two new plots* of the two exponentially decaying sinusoidal voltage functions, given by

$$v_1(t) = e^{-0.5t}\sin(4t) \quad v_2(t) = e^{-0.25t}\cos(4t)$$

over the range $0 \le t \le 10$. You can use the `linspace` command to create the required `t` vector and you will need to use the dot operator, `.*`, to properly multiply the `exp` and `sin` and `cos` functions.

Type `help plot` following the MATLAB prompt, or access the main MATLAB help menu and search for "plot". After reading about the `plot` command, generate these two plots:

a. *Your first plot* should have both functions on the same axes in the same figure window. Add an appropriate grid, title, legend, and x and y labels to your plot. For example, you can use the commands:

```
>> grid on
>> title('Two Decaying Sinusoids')
>> legend('v1','v2')
>> xlabel('Time (s)')
>> ylabel('Voltage (V)')
```

Note that these commands must follow rather than precede the `plot` command.

You can adjust the range of the axes of this plot by using the command axis. For example try the command:
```
>> axis([0 5 -1 1])
```

What are the new ranges of the x and y axes?

Note, the command `axis equal` sets the ranges of the x and y axes to be the same.

b. *Your second plot* should plot the two functions on two separate axes in the same figure window, i.e., through the use of the `subplot` command as described above.

2. Decide on a function you would like to plot and the range over which you are interested. Use MATLAB to generate a plot of this function. Use appropriate labeling, add in a grid, and choose your own line and/or symbol style for the plot.

Make sure to discuss your final plots with your TA before you leave the lab.