I. Graphical User Interface  
a. Common to all versions (Unix, Linux, Mac, Windows), although some differences exist.  
b. We will focus on the Windows GUI

II. Program Anatomy  
a. Command Window  
b. Command History  
c. Workspace Window  
d. Launch Pad  
e. Online Help

III. Working with M-files  
a. Although you can use MATLAB by only entering commands into the Command Window, it is much more convenient to use the M-file interface, as it allows you to edit, test, re-edit your program, and save it for the future. Also, all homework and Lab assignments will require that you print the M-file and submit it for grading.  
b. Jump to the end of this document to find out the step-by-step process for using M-files.

Matlab is an array-oriented language uniquely suited for linear algebraic manipulations. Let's look at the syntax for entering arrays, vectors and matrices.

IV. MATLAB syntax  
a. Matlab is an array-oriented language uniquely suited for linear algebraic manipulations.

Entering an array (row vector) \( x \)

\[ x = [1 \ 2 \ 3] \]

Converting to a column vector

\[ x = x' \]

Now \( x \) is a column vector in MATLAB memory

Since memory can be a scarce commodity, you can destroy the vector \( x \)

`clear x`
x is cleared in MATLAB memory
Now let's see a syntax for directly constructing the column vector $x$

$$x = [1; 2; 3]$$

So, the semicolon, used this way, signals the end of a column.

The same syntax is used to denote the end of a row for matrices:
entering a 3 x 3 matrix $A$

$$A = [1 0 0; 0 1 0; 0 0 1]$$

MATLAB will display the result of each program statement, unless you
use a semicolon at the end of the program statement:

$$C = [1 1 0; 2 3 1; 4 3 7];$$

This is good programming practice, as it avoids cluttering your output
with intermediate results. Now let's solve the matrix equation $AX=b$

$$b = A \times x$$

Note that we need to use a column vector for $x$, following the rules of
linear algebra. If we try to use an array (row vector) for $x$:

```matlab
clear x
x = [1 2 3];
b = A * x
```

We get an error and MATLAB halts. Instead you can execute the
following:

```matlab
b = A * x'
```

("""" means to calculate the transpose of $x$) and everything works okay.

You can also output descriptive text using 'disp':
```matlab
disp ('Solving the Matrix Equation Ax = b:')
```

```matlab
b = A * x'
```
Finally, if you want to clear all memory and variables allocated so far, issue the following command:

```
clear all
```

**Standard Arrays**

There are a number of standard arrays used in MATLAB programs

First, let's look at how you can fill an array with evenly spaced values. Let's say you want to create a row vector \( x \) consisting of the numbers from 1 to 10:

\[
x = 1:10
\]

Or, you might be interested in creating a vector of all the even numbers from 0 to 20:

\[
y = 0:2:20
\]

The general syntax for this row vector creation command is

\[
x = \text{first}:\text{increment}:\text{last}
\]

In fact, you can create any linearly spaced row vector \( x \) using, e.g.,

\[
x = \text{linspace}(1, 20, 10)
\]

Or a logarithmicall-spaced vector:

\[
xl = \text{logspace}(1, 20, 10)
\]

The general syntax here is

\[
x=\text{linspace} (\text{firstElement, lastElement, numElements})
\]

or

\[
x=\text{logspace} (\text{firstElement, lastElement, numElements})
\]
For creating standard matrices, the following functions are useful:

```matlab
A = eye(3)
```

creates a square (3x3) identity matrix

```matlab
B = ones(3)
```

creates a square (3x3) matrix of ones

```matlab
C = ones(3,2)
```

creates a 3 x 2 matrix of ones

The same can be done with `zeros()`, with the same syntax, except that matrices consisting completely of zeros are created:

```matlab
D = zeros(3)
D = zeros(3,7)
```

To access an individual array or matrix element, use the following syntax:

To get the third element of the array `x`:

```matlab
x(3)
```

To get an element from a matrix at row 2, column 3:

```matlab
D(2,3)
```

### Matrix functions

You can have MATLAB calculate several important properties of matrices. The following functions will do what would otherwise require a lot of analytical calculation. Most of these functions require a square matrix.

```matlab
clear all
```
\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 7 \end{bmatrix} \]

disp('The rank of A is:')
rank(A)

disp('The nullspace of A is:')
null(A)

disp('The determinant of A is:')
det(A)

disp('The eigenvalues of A are:')
eig(A)

There are many other matrix functions. Consult Online Help for others

**Vector operations**

Being an array-oriented language, MATLAB is happiest when handling arrays. There are many functions in MATLAB that will work on either scalars or row vectors (not column vectors). For example, you can calculate the sine of \( \pi/2 \) using the following syntax:

\[ d = \sin(\pi/2) \]

'\pi' is a predefined variable in MATLAB - see online help
Or you can fill an array with values, say from 0 to \( \pi \), and calculate the sine of each value in one step:

\[ x = \text{linspace}(0, \pi, 10) \]
\[ y = \sin(x) \]

Plotting results

To plot some data in MATLAB, the plot function is used. Let's plot the sine function over the interval 0 to 2*\( \pi \):

\[ x = \text{linspace}(0, 2\pi, 100); \]
\[ y = \sin(x); \]
\[ \text{plot}(x,y); \]
Or we could move the evaluation of the function into the plot function call:

\begin{verbatim}
plot(x, cos(x));
\end{verbatim}

**Using M-Files**

To keep all your MATLAB code in a single file (required for homework and Lab submissions), you should create an M-file. On the main MATLAB menu, choose File->New->M-file. This creates a blank document. Enter your code into this file. As an exercise, try entering the code below:

\begin{verbatim}
% Example of using an M-file
% Start a comment line using a '%'
% clear all memory (usually, you want this as the first line in your code)
clear all
% print some descriptive text
disp ('Testing M-files');
% initialize some vectors
x = [1 ; 2 ; 3];
y = [3 ; 4 ; 5];
% get ready for output
disp ('x + y is:');
% calculate the vector sum, and output the results to the command window
z = x + y
% all done
\end{verbatim}

To run your code, use the menu command Debug->Save and Run. Matlab asks you where to save your M-file. Important: save the file somewhere in 'My Documents'. Matlab will then ask you if you want to change the Matlab working directory to that directory where you saved your file - answer yes.

Your code is run, and any output (or errors) are found in the command window of Matlab.