Author: Alex Dubec
Edited By: Alissa Cooperman

Introduction to MATLAB and Useful MATLAB Commands

1 What is MATLAB?

Simply put, MATLAB is a "numerical computing environment", and is an extremely powerful computational tool. MATLAB is particular adept at matrix operations, and is commonly used in engineering applications in industry. MATLAB also includes Simulink, which we can use to model dynamic systems. We'll use this tool later in the semester for testing the limits of the linearity/small-angle assumptions that we have been making; for now, though, you won't need to use it. For the purposes of our class, we'll use MATLAB to perform computations that would be taxing to perform by hand (like the Fourier transform), and to model dynamic systems. You can also program in MATLAB; the labs in ES 125 will ask you to write short programs.

MATLAB is available for use on all Windows and Mac computers in the Science Center and house computer labs, as well as for download from FAS IT.[1] The latest version is R2008b; that said, R2008a (which is the version available from FAS IT), and basically any other version of MATLAB you might be able to find, will have the tools you'll need for ES 125.

If you've never used MATLAB before, or are stuck, there is extensive Help documentation available from the Help menu within MATLAB. There are also several demos included in MATLAB, which may help you understand some of the program's features, as well as on-line tutorials available at the MathWorks website (The MathWorks is the company that produces MATLAB).[2] Of course, if you're really stuck, you can always ask Mike, Alex, or Professor Howe for help.

2 MATLAB Interface

When you open MATLAB, you will be greeted by a paneled display, consisting of the Command Window, Current Directory , Workspace, and Command History windows. If any of these windows are not available, you can open them from the Desktop menu. Each of these windows can be undocked and resized. You should also see the MATLAB and Shortcuts toolbars at the top of your screen; the MATLAB toolbar in particular has useful shortcuts as well the option to change the current directory. Should either of these toolbars not be displayed, they too are available from the Desktop menu.

- Command Window

  The Command Window, intuitively, is the window into which you will type commands. Pressing the up directional key on the keyboard will bring up the previous command (and every press of up will bring up successively older commands; you can also use the down key to return to newer commands). MATLAB's command prompt looks like this:

  ```
  >>
  ```

  and you can type commands whenever the prompt is available (if you're running a script or a particularly intensive command, the prompt will not be displayed).

- Current Directory

  Current Directory lists the contents of the current directory in which you are operating. You can select the current directory from the MATLAB toolbar at the top of your screen. This is particularly important for running scripts and importing data - you won't be able to run a script that you've written or import a data file from a directory other than the current one.

[1] http://downloads.fas.harvard.edu/download; you'll have to log in with your Harvard ID and PIN.
[2] http://www.mathworks.com/academia/student_center/tutorials/launchpad.html.

- Workspace

  The Workspace displays the variables that you have created and some basic information about them, such as their size and type. You can also create or delete variables from the Workspace (although you may find that this is most easily done from the Command Window; the commands for this are explained later).

- Command History

  The Command History window displays a list of commands that you have run since the last time the window was cleared (it will retain commands even if you close and re-open MATLAB). It's primary use is that you can drag previous commands from it into the Command Window.

3 .m Files and the Editor

The Command Window is great for running one or two commands, but if you wanted to create a process that requires more than a couple of commands, or if you want to re-run your commands later, retyping commands would get tedious. Instead, you can create a script (called an .m file) which you can save, edit, open, and re-run. .m files are created from File - New - M-File, and will open the Editor on creation. The Editor interface will allow you to create scripts - which are essentially collections of commands. No special heading or ending lines are required; just put each new command on a different line. For example, if you wanted to create a variable A and set its value equal to 3, then multiply A by 2, and then display it, you could do this in the Command Window by typing:

```
>> A = 3;
>> A = 2*A;
>> A
```

or you could do it in an .m file by typing:

```
% Create variable A
A = 3;
% Multiply A by 2
A = 2*A;
% Display A
A
```

Note that the `%` character (without quotes) is used to create comment lines - lines where you can type English-language comments without having them interfere with your program. These lines will turn green, to distinguish them from the rest of your script. Try to use these liberally to explain what your code is doing; both you and your TF's will thank you later!

To run .m files, you can either press Run from the Editor Toolbar, press F5, or type in the name of the .m file in the Command Window.

4 Variables, Notation, and Operators

You'll often use variables in MATLAB. With some exceptions, you don't need to explicitly create variables; that is, you can create them on the fly. If you want a variable V to contain the number n, you can simply say `V = n` (no initialization of `V` is required). The exception to this rule is symbolic variables; if you want to create a function of a symbolic variable, and do algebra with it, you'll need to use `syms`[3] to create that variable first.[4] Remember that there is also a listing of current variables in the Workspace.

[3]The `syms` function is explained in §5.4.

[4]MATLAB generally does numerical computation; when it can avoid it, it doesn't do algebra. Symbolic variables and functions will force it to algebra, and these variables and functions can be subjected to the usual mathematical tricks (differentiation, integration, and the like).

4.1 Creating Variables

- Variables

  Scalar variables are very easy to create. Simply use the equals sign (`=`) to assign a quantity to a variable. Note that the variable name is to the left of the sign, and that the quantity to be assigned is on the right. For example, a variable V would be defined as follows: `V = 3*exp(4);`.

- Symbolic Variables

  In order to create symbolic variables, you'll need to initialize them with the `syms` command first. For example, if you wanted to create the symbolic function `sin(x)`, and then do some math with that that function, you would need to first initialize `x` by calling `syms x`.

- Vectors and Matrices

  Vectors and matrices are defined similarly to variables, except that their definitions require the use of square brackets (`[` and `]`). Spaces separate columns within a row; semicolons separate rows from one another. For example, in order to define the following matrix $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, you would type `M = [a b; c d]`.

4.2 Indices

MATLAB uses standard matrix notation when describing locations in a matrix (that is, an n by m matrix will have n rows and m columns). This is useful if you're interested in a specific value in a matrix or a vector. Should you want to access the a by b'th cell in a matrix M, you would call `M(a,b)`.

4.3 Frequently Used Operators

- `:`

  The `:` operator has two frequently used functions:

  - As "everything in a column or row"

    `:` is useful for saying 'for all values of this index'. If you have a n by m matrix, where n is large (such as what might occur in a data file from lab, where you've taken many data points), but are interested in only the mth column of that data, you'd use the `:` operator as a placeholder for the other index. For example, consider a matrix `data`, which is a 1000 x 2 matrix. If you were interested in only the second column, you could call `data(:,2)`.

  - As a tool for defining vectors

    If the need should arise to create a vector of evenly-spaced numbers, `:` can be used to great effect. For example, if you need to create a vector V which goes from 1 to 10 with spacing of 0.1, you could type `V = 1:0.1:10`, where we have *lowbound*`:`*spacing*`:`*highbound*. V would then contain the numbers 1, 1.1 …9.9, 10. This sort of vector is particularly useful when evaluating some function at many points.

- Element-wise Operators

  Element-wise operators are very commonly used in MATLAB. Because MATLAB deals so often with matrices, the basic math operations ($+$, $-$, $*$, and %) as well as exponents (^) are defined in their matrix-math sense. Should you instead want to multiply every entry in a matrix or vector by a scalar, you'd use the element-wise operator `.`. For example, if you wanted to multiply every element in a matrix M by some integer i, the command `M*i` would return an error; instead, you'd use `M.*i`.

- `'`

  The apostrophe (`'`) character gives a matrix' or vector's transpose. For example, if you wanted the transpose of some matrix M, you'd call `M'`.

- `<`, `<=`, `>`, `>=`, `==`

  These are the inequality and equality operators. Note that equality is denoted by `==`; `=` is used for variable assignment.

- `&&`, `||`, `~`

  These are the logical operators for and, or, and not respectively.

5 Useful Commands

5.1 Commands that Tidy Things Up (or Make Your Life Easier)

The following commands will either tidy up your Command Window, or are shortcuts, or are useful for making your scripts more efficient.

- `format`

  The `format` command tells MATLAB to how many decimal places it should go when displaying numbers; `format short` will limit it to 4, whereas `format long` will give you 15.

- `clc`

  `clc` clears the Command Window.

- `close`

  `close` closes figures; this command is usually used as `close all`, which closes all open figures.

- `clear`

  `clear` clears variables. To clear a specific variable, type `clear name`; to clear all variables in use, type `clear all`.

  It's *usually* helpful to run this command at the beginning of a script, so long as you're not clearing something useful[5]

- `;`

  The `;` operator suppresses output, and is one of the most handy tools in MATLAB. That is, if you want to perform some calculation, but don't want the output printed in the command window (like if you were importing a data file or performing a manipulation on a large matrix), using the `;` operator at the end of a command will prevent output from being displayed.

- Control-C

  Pressing Control-C will abort whatever operation you're currently running. This is useful if you've done something long in error - like having MATLAB display the entire contents of one of the data files you took in lab.

- Tab

  Tab brings up a completion menu, and can be used to complete half-written commands. That is, if you want to use the clear command, but don't want to type it all out, you could type `cle` and press Tab; all MATLAB commands beginning in `cle` would be displayed, and you could select `clear`.

- Up/Down Keys

  The Up and Down keys can be used in the Command Window to place old commands at the prompt without having to retype them.

[5] When dealing with matrix and vector variables with the same name, MATLAB does not overwrite all previously existing data - it only overwrites the area that the new variable needs. For example: if you're using the 4x4 matrix M, and you want to overwrite M with a 3x3 matrix, M will still be a 4x4 matrix with data in the last column and row. If you then perform some operation with M, that extra data is still used, and your output can get very confusing.

- F5

  F5 is useful in the Editor; it runs your script without having to return to the Command Window.

- F9

  F9 is also useful in the Editor; when a portion of a script is selected, it runs that selection (as opposed to the entire script).

- `%`

  `%` is the comment operator; use it to denote comment lines in .m files.

- `for`, `while`, `end`

  `for`, `while`, and `end` are used to create for and while loops. This is very handy when performing the same operation many times. For example, if you wanted to display the first five powers of two, you could create the following for loop:

  ```
  A = 1;
  for i = 1:5
       A = 2*A
       i = i+1;
  end
  ```

  In this case, `i` is the index variable - the variable that tells the for loop how many times it should run. In this case, the for loop is set to run for values of `i` from 1 to 5. You can also create while loops that perform the same function, or if you didn't know how many times the loop needed to be run, you could use equality or inequality operators with a while loop. Feel free to use whichever loop you are most comfortable with.

5.2 Commands that Perform Operations with Variables

The following commands are useful for working with variables, vectors, and matrices.

- `find()`

  The `find()` function, when given a vector and an equality, returns the index where the equality is true. For example, `find([1 4 3 2] == 4)` returns 2.

- `load('')`

  When run with the name of a data file as its argument, the `load('')` function imports that data into MATLAB. This function is almost always used to assign information into a variable.

- `length()`

  `length()` returns the length of a vector.

- `size()`

  `size()` returns the size of a matrix.

5.3 Commands that Do Interesting Things

The following commands are useful for image processing

- `imread()` The import image function. The function's input is the filename of the image to be imported surrounded by single quotations. The output is the matrix representation of the image.

- `rgb2gray()`

  The convert RGB image or colormap to grayscale image.

- `imhist()`

  The function takes an image as an input. It displays a histogram for the image.

The following commands perform mathematical operations.

- `fft()`

  The discrete Fourier transform function. This function's input is a vector with size nx1. The output of this function is part of the definition of the power vector (see Lab 1 Handout for details).

- `round()`

  Rounds its argument to the nearest integer.

- `log()`

  The logarithm function. Note that this is the natural log.

- `exp()`

  In MATLAB, the exponent function `exp` is called with arguments (that is, e^x is `exp(`x`)`. Should you just need the value of e, call `exp(1)`.

- `pi`

  Pi, should you need it.

5.4 Commands that Do Symbolic Things

The following commands are useful for symbolic operations (that is, algebra).

- `syms`

  `syms` is required for initializing symbolic variables. For example, if you wanted to create the symbolic function `sin(x)`, and then do some math with that that function, you would need to first initialize `x` by calling `syms x`.[6]

- `int()`

  `int()` is used for the integration of a symbolic function.

- `diff()`

  `diff()` is used for the differentiation of a symbolic function.

- `eval()`

  `eval()` evaluates symbolic functions, or vectors and matrices with symbolic contents.

5.5 Figures and Plots

The following commands are useful for plotting graphs and figures.

- `figure`

  Opens a figure window.

- `plot()`

  `plot()`, when called with two vector arguments, will plot the two vectors against each other. `plot()` sometimes has funky behaviour when the two vectors are of different length; check to make sure that they are the same length if you're getting weird results!

[6]This is opposed to calling `sin()` on some vector V of closely-spaced values - that would be numerical, rather than symbolic, computation.

- `semilogx()`, `semilogy()`, `loglog()`

  `semilogx()`, `semilogy()`, and `loglog()` are the logarithmic plot tools.

- `scatter()`

  `scatter()` creates a scatterplot (no lines connect data points).

- `axis([])`

  `axis([xmin xmax ymin ymax])` forces a plot's axis to its arguments (MATLAB plots usually automatically size their axes).

- `title(‘’)`

  To title a plot, use the `title(‘’)` command, with the plot's title between the quotes.

- `xlabel(‘’)`, `ylabel(‘’)`

  Similar to `title(‘’)`, `xlabel(‘’)` and `ylabel(‘’)` label the x and y axes of a plot respectively.

- `hold`

  If plotting multiple plots on the same figure, you'll need to use `hold on`, which tells MATLAB to continue to plot on the same figure, or `hold off`, which prevents the same.

- `legend(‘’)`

  Creates a legend for your plot. `legend(‘’)` takes multiple text arguments, where each line's description in the legend is its own argument.

- `subplot(a,b,c)`

  `subplot` is useful for putting multiple plots in the same figure. It creates an `a` by `b` matrix of plots in a single figure window; the `c` argument tells MATLAB into which location to plot. `c` is counted in the same way that English is read; it incrementas across lines from left to right, and then down successive lines from top to bottom.

5.6 Error Messages

If at some point you make a mistake in your code, and MATLAB runs into an error, the program will beep and place a red error message in the Command Window. If the error occured while running a script, the error message will also tell you on which line the error occured, and will sometimes also tell you where on that line the problem is (if you're missing an argument to a function, or have too many or too few parentheses) - which is very useful for debugging.

6 Other Stuff, and Getting Help

These functions barely scratch the surface of what you can do with MATLAB. If there are any other functions that you wish existed, they very well might; there are other functions that you may find useful for ES 50 (and there are definitely other functions that you will use if using MATLAB in other classes!). MATLAB is very well documented (although it can be somewhat cryptic), and the Help interfaces should be your first stop for examples if you're stuck. The above descriptions of functions are by no means exhaustive; there are many other ways to use the functions listed, and the Help menus contain examples for each of their uses. Also, remember that if you're really stuck, the teaching staff is here to help.