A Guide to Using Some Basic MATLAB Functions

UNC Charlotte

This document provides a brief overview of some of the essential MATLAB functionality. More thorough descriptions are available in a number of books. In addition, the Mathworks has excellent documentation available on the web. It is my hope that the combination of this document and the Mathworks help on the web will provide you with the tools that you need to be successful in starting to explore MATLAB. Once you get the hang of the basics, you can easily begin to figure things out on your own.

1.0 Getting Help with MATLAB

If you know the name of a function which you would like to learn how to use, use the '**`help`**' command:

```
>> help functionname
```

This command displays a description of the function and generally also includes a list of related functions. If you cannot remember the name of the function, use the '**`lookfor`**' command and the name of some keyword associated with the function:

```
>> lookfor keyword
```

This command will display a list of functions that include the keyword in their descriptions. MATLAB also contains a variety of demos that can be with the '**`demo`**' command.

A variety of additional resources are available via the Mathworks web site. The web materials often contain more specific examples of how to use each of the functions.

2.0 MATLAB Variables — Scalars, Vectors, and Matrices

MATLAB stores variables in the form of matrices which are M ×N, where M is the number of rows and N the number of columns. A 1 × 1 matrix is a scalar; a 1 × N matrix is a row vector, and M×1 matrix is a column vector. All elements of a matrix can be real or complex numbers; $\sqrt{-1}$ can be written as either '**`i`**' or '**`j`**' provided they are not redefined by the user.

A matrix is written with a square bracket '**`[]`**' with spaces separating adjacent columns and semicolons separating adjacent rows. For example, consider the following assignments of the variable **`x`**:

- Real scalar **`>> x = 5`**
- Complex scalar **`>> x = 5+10j (or >> x = 5+10i)`**
- Row vector **`>> x = [1 2 3] (or x = [1, 2, 3])`**
- Column vector **`>> x = [1; 2; 3]`**
- 3 × 3 matrix **`>> x = [1 2 3; 4 5 6; 7 8 9]`**

There are a few notes of caution. Complex elements of a matrix should not be typed with spaces, i.e., '**`-1+2j`**' is fine as a matrix element, '**`-1 + 2j`**' is not. Also, **`'-1+2j'`** is interpreted correctly whereas '**`-1+j2`**' is not (MATLAB interprets the '**`j2`**' as the name of a variable. You can always write '**`-1+j*2'`**.

2.1 Complex Number Operations

Some important complex number operations are shown below:

- Assume we create a complex scalar **`>> x = 3+4j`**
- We can then ask MATLAB a number of different questions about **`x`**:
    - Real part of x **`>> real(x)`** (This will return 3)
    - Imaginary part of x **`>> imag(x)`** (This will return 4)
    - Magnitude of x **`>> abs(x)`** (This will return 5)
    - Angle of x **`>> angle(x)`** (This will return 0.9273)
    - Complex conjugate of **`x >> conj(x)`** (This will return 3 - 4i)

2.2 Generating vectors

Vectors can be generated using the '**`:`**' command. For example, to generate a vector **`x`** that takes on the values 0 to 10 in increments of 0.5, type the following which generates a 1×21 matrix:

```
>> x = [0:0.5:10];
```

Other ways to generate vectors include the commands: '**`linspace`**' which generates a vector by specifying the first and last number and the number of equally spaced entries between the first and last number, and '**`logspace`**' which is the same except that entries are spaced logarithmically between the first and last entry.

2.3 Accessing vector elements

Elements of a matrix are accessed by specifying the row and column. For example, in the matrix specified by **`A = [1 2 3; 4 5 6; 7 8 9]`**, the element in the first row and third column can be accessed by writing

`>> x = A(1,3)` which yields 3.

The entire second row can be accessed with

`>> y = A(2,:)` which yields [4 5 6]

where the '**`:`**' here means "take all the entries in the column". A submatrix of **`A`** consisting of rows 1 and 2 and all three columns is specified by

`>> z = A(1:2,1:3)` which yields [1 2 3; 4 5 6]

3.0 Matrix Operations

MATLAB contains a number of arithmetic, relational, and logical operations on matrices.

3.1 Arithmetic Operations

The basic arithmetic operations on matrices (and of course scalars which are special cases of matrices) are:

- **+** addition
- **-** subtraction
- ***** multiplication
- **/** right division
- **** left division
- **^** exponentiation (power)
- **'** conjugate transpose

An error message occurs if the sizes of matrices are incompatible for the operation. Division is defined as follows: The solution to **A * x = b** is **x = A\b** and the solution to **x * A = b** is **x = b/A** provided **A** is invertible and all the matrices are compatible.

3.2 Element-by-Element Arithmetic Operations

Many times, you wish to multiply or divide each element in a vector by each element in another vector. That can be a very confusing operation for many students. Let's look carefully at this case.

Addition and subtraction involve element-by-element arithmetic operations; matrix multiplication and division do not. However, MATLAB provides for element-by-element operations as well by prepending a '**.**' before the operator as follows:

- **.*** multiplication
- **./** right division
- **.** left division
- **.^** exponentiation (power)
- **.'** transpose (unconjugated)
- Note that the '.' is not needed for element-by-element addition or subtraction

The difference between matrix multiplication and element-by-element multiplication is seen in the following example

```
>>A = [1 2; 3 4]
A =
1 2
3 4
>>B=A*A
B =
7 10
15 22
>>C=A.*A
C =
1 4
9 16
```

3.3 Math Operations

MATLAB comes with a large number of built-in functions that operate on matrices on an element-by element basis. These include:

- **`sin`** sine
- **`cos`** cosine
- **`tan`** tangent
- **`asin`** inverse sine
- **`acos`** inverse cosine
- **`atan`** inverse tangent
- **`exp`** exponential
- **`log`** natural logarithm
- **`log10`** common logarithm
- **`sqrt`** square root
- **`abs`** absolute value
- **`sign`** signum (i.e. is the number in question either positive or negative?

Example: Let's say I have

```
>> theta = [0:1:9999]*(2*pi/10000);
>> x = sin(theta);
```

The result will be a 10000 point vector (or 1x10000 matrix) containing the sine of each of the angles contained in the vector **`theta`**.

4.0 MATLAB Files

There are several types of MATLAB files including files that contain scripts of MATLAB commands, files that define user-created MATLAB functions that act just like built-in MATLAB functions, files that include numerical results or plots.

4.1 M-files

MATLAB is an interpretive language, i.e., commands typed at the MATLAB prompt are interpreted within the scope of the current MATLAB session. However, it is tedious to type in long sequences of commands each time MATLAB is used to perform a task. There are two means of extending MATLAB's power — scripts and functions. Both make use of m-files (named because they have a .m extension and they are therefore also called dot-m files) created with MATLAB's built-in editor. The advantage of m-files is that commands are saved and can be easily modified without retyping the entire list of commands.

4.1.1 Scripts

MATLAB script files are sequences of commands typed with an editor and saved in an m-file. To create an m-file using MATLAB, click on "New Script" on the "Home Tab" in newer versions of MATLAB. If your version is older, you can gain access to the MATLAB editor using the "File" menu.

Let's say that you have created a script file called **`filename.m`** The instructions contained in this file are executed by typing the file name in the command window at the MATLAB prompt, i.e., the m-file **`filename.m`** is executed by typing

```
>> filename
```

Execution of the m-file is equivalent to typing the entire list of commands in the command window at the MATLAB prompt. All the variables used in the m-file are placed in MATLAB's workspace. The workspace, which is empty when MATLAB is initiated, contains all the variables defined in the MATLAB session.

4.1.2 Functions

A second type of m-file is a function file which is generated with an editor exactly as the script file but it has the following *general* form:

```
function [output 1, output 2] = functionname(input1, input2)
%
%[output 1, output 2] = functionname(input1, input2) Functionname
%
% Some comments that explain what the function does go here.
%
MATLAB command 1;
MATLAB command 2;
MATLAB command 3;
```

The name of the m-file for this function is functionname.m and it is called from the MATLAB command line or from another m-file by the following command

```
>> [output1, output2] = functionname(input1, input2)
```

Note that any text after the '**%**' is ignored by MATLAB and can be used for comments. Output typing is suppressed by terminating a line with '**;**', a line can be extended by typing '**...**' at the end of the line and continuing the instructions to the next line.

4.2 Mat-Files

Mat-files (named because they have a **.mat** extension and they are therefore also called dot mat files) are compressed binary files used to store numerical results. These files can be used to save results that have been generated by a sequence of MATLAB instructions. For example, to save the values of the two variables, **`variable1`** and **`variable2`** in the file named **`filename.mat`**, type

```
>> save filename.mat variable1 variable2
```

Saving all the current variables in that file is achieved by typing

```
>> save filename.mat
```

A mat-file can be loaded into MATLAB at some later time by typing

`>> load filename` (or **`load filename.mat`**)

5.0 Plotting

MATLAB contains numerous commands for creating two- and three-dimensional plots. The most basic of these commands is 'plot' which can have multiple optional arguments. A simple example of this command is to plot a function of time:

```
t = linspace(0, 8, 401); %Define a vector of times from ...
0 to 8 s with 401 points
x = t.*exp(-t).*cos(2*pi*4*t); %Define a vector of x values
plot(t,x); %Plot x vs t
xlabel('Time (s)'); %Label time axis
ylabel('Amplitude'); %Label amplitude axis
```

This script yields the plot shown in Figure 1.

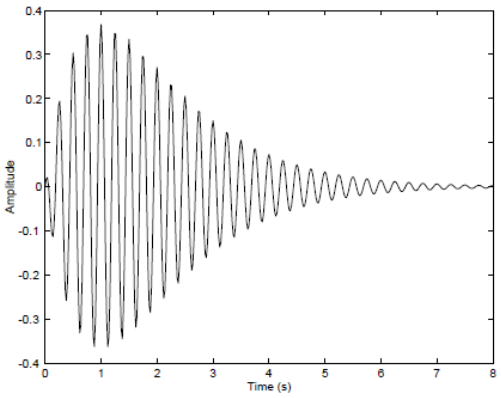

Figure 1: Example of the plotting of the function $x(t) = te^{-t}\cos(2\pi 4t)$.

5.1 Simple Plotting Commands

The simple 2D plotting commands include

- **`plot`** Plot in linear coordinates as a continuous function
- **`stem`** Plot in linear coordinates as discrete samples
- **`loglog`** Logarithmic x and y axes
- **`semilogx`** Linear y and logarithmic x axes
- **`semilogy`** Linear x and logarithmic y axes
- **`bar`** Bar graph
- **`errorbar`** Error bar graph
- **`hist`** Histogram
- **`polar`** Polar coordinates

5.2 Customization of plots

There are many commands used to customize plots by annotations, titles, axes labels, etc. A few of the most frequently used commands are the following:

- **`xlabel`** Labels x-axis
- **`ylabel`** Labels y-axis
- **`title`** Puts a title on the plot
- **`grid`** Adds a grid to the plot
- **`gtext`** Allows positioning of text with the mouse
- **`text`** Allows placing text at specified coordinates of the plot
- **`axis`** Allows changing the x and y axes

- **`figure`** Create a figure for plotting
- **`figure(n)`** Make figure number n the current figure
- **`hold on`** Allows multiple plots to be superimposed on the same axes
- **`hold off`** Release hold on current plot
- **`close(n)`** Close figure number n
- **`subplot(a,b,c)`** Create an a × b matrix of plots with c the current figure
- **`orient`** Specify orientation of a figure

6.0 Loading Data from a CSV File from the Tektronix Scope

MATLAB can load in various data files, including the CSV (comma-separated variable) files created by your Tektronix oscilloscope. The simplest way to do this is to use the **`csvread`** command.

As an example, let's say that I recorded a voltage waveform that contains 2500 samples contained between the cells E1 (which is location (0,4)) and E2500 (which is location (2499,4)). In my case, I loaded this data into a vector **`v1`** using the following command:

```
>> v1=csvread('TEK0000.csv',0,4,[0 4 2499 4]);
```

For more help on how to use this command, type **`help csvread`** at the MATLAB command prompt or use that vast treasure trove of information known as the internet.