Carnegie Mellon

DSP-I

Digital Signal Processing I (18-791)
Fall Semester, 1997

Department of Electrical and Computer Engineering

INTRODUCTION TO MATLAB

1. Introduction

In this course, you will be required to use the software tool MATLAB for a number of assignments. MATLAB is similar to a very easy-to-use, high level programming language that is tailored for use in communication and control systems and signal processing. It is used by industrial and academic professionals worldwide for research, development, and design. MATLAB will be extremely useful for this and many of your other courses for a number of reasons. In particular, MATLAB makes it very easy for you to perform matrix and vector operations, to operate with complex numbers, and to plot your results. In this course, you will find that MATLAB is very useful, not only for the specifically assigned MATLAB problems, but also for checking many of your solutions to the problem sets.The version of MATLAB available to you in ECE at CMU is the full research version, not the restricted student version, and thus is the same software that many of you will have access to in your future jobs after graduation.

2. Access and Documentation

MATLAB can be accessed by you through the workstations in HH 1107 (The DEC Lab), either by logging onto these machines directly or over the network. Alternatively, MATLAB also is available on the Macintosh computers in the Hunt Clusters. Mathworks (the company that makes MATLAB) also allows students to make a copy for use on their personal Mac or PC during a course. We are working to have this distribution run through the computer store, and will keep you informed. If you have a PC or a Mac of your own, you also can purchase a copy of the student edition of MATLAB (a slightly limited version of MATLAB that is sufficient for coursework) with a users' manual at the CMU bookstore. You also can purchase the user's manual, without a disk. You will find that MATLAB will be useful for many of your courses in ECE, *even those for which* MATLAB *is not required*.

An additional tool called SIMULAB is available on-line as well. SIMULAB is used with MATLAB to make it easy to simulate real control, communication, and signal processing systems and will be introduced to you later in the course. Also available on the workstations in the DEC Lab are copies of the signal processing, control systems, robust control, and optimization toolboxes. We encourage you to play around with these tools; they allow you to quickly check your intuition by seeing the result of small changes to systems of equations representing signals and systems.

You have been provided with the MATLAB Primer, Third Edition. This primer provides an *excellent* overview of and introduction to MATLAB. *BE SURE TO READ IT.* MATLAB also has an on-line help function and a demo. We recommend that you try the demo; it shows you some of interesting and useful functions that MATLAB can perform. The primer and the on-line help should provide you with the information that you need to effectively use MATLAB to complete your assignments. Complete sets of Manuals are available for your use in HH 1107 if you need further information.

3. Starting MATLAB

3.1. MATLAB on a workstation

• Log on to your machine.

• Open an xterminal by typing:

```
xterm &
```

• When the new window opens, activate it by either clicking the mouse in the window, or just moving the mouse into the window, depending on your system.

• Create a directory (call it matlab if you want), and go into it:

```
mkdir matlab
cd matlab
```

• If you're not on an ece machine (such as the clusters in Wean Hall) type:

```
klog user_id@ece.cmu.edu
```
(supply your password when prompted)

• Start matlab by typing:

```
/usr/local/bin/matlab
```
or
```
/usr/supported/bin/matlab
```
Also, some machines may already be configured to run MATLAB just by typing **`matlab`**. Try it!

• In your xterm window, you should now see some information about MATLAB, and a prompt

```
>>
```
Hereafter, this prompt will be referred to as the MATLAB prompt. Congratulations! You are now in the MATLAB environment and are ready to begin.

3.2. MATLAB on a Macintosh

MATLAB is already available on some campus Macs. Check the 'classes' folder on the applications disk. In Hunt library, the Apple cluster (downstairs) has three sections. The Macs in the outer sections have MATLAB already loaded. MATLAB will not function properly, especially when printing, if it is being run from a locked disk. Note that the applications disks on all campus Macs are usually locked. You must, therefore, create a folder on the user disk (call it **`matlab`**, if you like) and copy the MATLAB Program, **`MATLAB_Toolbox and Signal_Processing_Toolbox`**. Do not copy the whole MATLAB folder from the applications disk, as this will take several minutes.

Additional Mac notes:

MATLAB, on the Mac, differs from the workstation program in printing the graphics window. The **`print`** command will not work. You must use the print command from the file menu.

The Mac version of MATLAB has a nice editor built in for editing M-files. Use **`new`** from the file menu (or **`open`**) to start up the editor. Important: changes in the M-file will not take effect until you save the file you are editing.

4. Useful commands

4.1. General hints

To exit the MATLAB environment, simply type **`quit`** in response to the MATLAB prompt.(p. 1, sec. 1).[1]

• If you get into trouble (a runaway computation or display), **`Ctrl-C`** will stop the executing command (p. 3, sec. 4).

4.2. Listing variables

• MATLAB is case sensitive. Thus a variable called "**`A`**" is different than a variable called "**`a`**". (p. 3 section 4).

• You can check to see what variables you have defined by typing **`who`**:

```
>> who
```

If you are now interested in the variables that you have defined during this MATLAB session, and what types they are, you can use the command **`whos`**:

```
>>whos
          Name        Size       Total    Complex

           ans      1 by 1          2         Yes
             z      1 by 1          2         Yes


Grand total is (4 * 8) = 32 bytes.
```

You can check the current value of any variable by simply typing the variable name in response to the MATLAB prompt. Note that the size of each variable is listed. In MATLAB, all variables are considered to be $m \times n$ matrices. A scalar is simply a 1×1 matrix, and a vector of length N is simply an $N \times 1$ matrix. Thus all of the symbols for arithmetic operations actually designate matrix operations (See Primer, section 3).

4.3. Saving your Results

You may wish to have a transcript of your results for a given session so that you can review it or print it out later, rather than writing down each answer. This can be accomplished by using the diary MATLAB command: **`>> diary prob1.doc`** will save a transcript of your MATLAB session from this point on, excluding plots, until you enter **`>> diary off`**. Then the transcript will exist in the file **`prob1.doc`** (for this example) in the current directory.

If you wish to quit a MATLAB session, perhaps to go to class, but would like to save the existing variables and their current values, simply enter the command **`save`**:

```
>> save
```

1. All page numbers and sections in this document refer to the MATLAB Primer.

```
    Saving to:  matlab.mat
```

Note that the file '`matlab.mat`' will now appear in your current directory. When you restart you next MATLAB session, you can restore your variables to the values that existed at the end of your last session by simply entering `load` in response to the MATLAB prompt, *as long as you are executing* MATLAB *from the same directory where you were when you executed the* `save` *command*.

4.4. Editing and Executing Past Commands

A useful feature of MATLAB is that you can use the up and down arrow keys on your keyboard to display and edit past commands to quickly make small changes. You can hit the up-arrow key to go backward through your previous commands until you reach the desired previous expression. Once that expression is displayed, you can edit it and then hit carriage return to re-evaluate the expression.

5. Some examples

MATLAB makes it very easy to generate and save nice plots.

5.1. Generating Complex Functions

In order to plot a function, we will need to generate a number of values of t from *0* to *2*, for example, and evaluate and plot $f(t)$ at these values. This task is accomplished in MATLAB by first defining a vector `t` that has as its elements all of the values at which we would like to evaluate $f(t)$. Defining `t` can be accomplished in a number of ways (See section 2 of the Primer). For this example, the simplest method is shown below, where the notation means that the first element is *0*, each subsequent element is computed by adding *0.5* to the previous element, and the last element is *2*.

```
>> t = 0:0.5:2

t =
     0    0.5000    1.0000    1.5000    2.0000
```

Note that MATLAB responds by printing *ALL* of the elements of the vector to the screen. For this problem, we would like to plot many more that *5* points to get an accurate representation of the function $f(t)$. However, for this longer vector, we would like MATLAB to evaluate the expression, but not print all of the elements of the vector to the screen. *Printing to the screen is suppressed by including a semicolon at the end of the command line.* MATLAB will perform the operation (in this case, create the vector `t` as indicated), but will not display the result for you to see. Let's use increments of *0.01* to define t:

```
>> t = 0:0.01:2;
```

Now the objective is to compute $f(t)$ for each element in t. In MATLAB this can be accomplished by entering the expression:

```
>> f = 2*exp(j*(2*pi*t + pi/4));
```

Note that f is now a complex-valued vector of the same length as t where each complex--valued element in the vector f was computed using the above expression for the corresponding value of the vector t. Such operations on matrices and vectors are called element-wise or scalar operations (See section 7).

Now the objective is to plot the real and imaginary parts of $f(t)$. First we need to define a vector **`fr`** comprising only the real parts and a vector **`fi`** comprising only the imaginary parts of each element of f. To accomplish this task in MATLAB, simply enter

```
>> fr = real(f); >> fi = imag(f);
```

5.2. Plotting Complex Function Components

Now we can use the plot command in MATLAB. Typing **`plot(x,y)`** will generate a plot where the values of the vector **`x`** are along the horizontal axis and the values of **`y`** are along the vertical axis. *Note that the vectors* **`x`** *and* **`y`** *must have the same number of elements.* The N^{th} point plotted is the point corresponding to the N^{th} value of **`x`** and the N^{th} value of **`y`**. For this example, enter

```
>> plot(t,fr)
```

Note that a new graphics window appears, with the axes automatically scaled. You can plot both the real and imaginary parts on the same graph as follows:

```
>> plot(t,fr,t,fi)
```

You can also explicitly specify different colors and line types for the curves by typing, for example,

```
>> plot(t,fr,'r',t,fi,'g')
```

where **`‘r’`** stands for red and **`‘g’`** stands for green. Other line-types are described on p. 17 in the Primer. You can add labels and grid lines as well. Try the following sequence of commands and notice what happens to your plot.

```
>> grid >> title('Prob. 6a') >> ylabel('f(t)') >> xlabel('t')
```

You may wish to label the different curves or provide a key. You can experiment with the commands **`‘gtext’`** and **`‘text’`**. To find out more use the on-line help, for example:

```
>> help gtext

GTEXT  Place text on a graph using a mouse.
GTEXT('string') displays the graph window, puts up a
cross-hair, and waits for a mouse button or keyboard key to be
pressed.  The cross-hair can be positioned with the mouse (or
with the arrow keys on some computers).
Pressing a mouse button
or any key writes the text string onto the graph at the selected
location.  See also: GINPUT
```

Other commands for generating plots in MATLAB are listed on page 32 of the Primer.

5.3. Saving Graphics

You will need to save your plots in a file to print them out later. This can be accomplished in MATLAB using the **`print`** command. The command

```
>> print -dps ps1plots
```

will save the current contents of the graphics window in a file called **`ps1plots.ps`**, in PostScript format, in the current directory.

Files having a **`*.ps`** extension can be directly sent to the printer. To print out your file, simply enter the Unix command

```
>> !lpr -Ptolkien ps1plots.ps
```

This command will print out your plots on the printer **`tolkien`** in the DEC Lab. Note that the ! allows you to execute Unix commands from MATLAB.

Finally, see Section 3 if you are using a Macintosh.

5.4. M-files and Editing

Once you are comfortable with the MATLAB commands, it is often easier to create a file of the commands using emacs (or the editor of your choice). If you store this file in your **`matlab`** directory with a **`*.m`** extension, you can then execute these commands from MATLAB by simply entering the filename in response to the MATLAB prompt. Such files are called script M-files. It is also possible to define functions, similar to subroutines, that also are called function M-files. All of these files also have the extension **`*.m`** If these files are stored in your **`matlab`** directory, then you can run functions directly from MATLAB as if they were pre-defined MATLAB commands. We will illustrate this concept below.

While you are in MATLAB, you can edit a file by preceding the Unix editing command by the **`!`**. For example, to create and edit the file **`'ps16a.m'`** using emacs, simply type

```
>> !emacs ps16a.m
```

You now can edit this file. If your current directory is the **`matlab`** directory, then this file should be created in the **`matlab`** directory. After editing this file, you can check to see if it is in the current directory by typing **`ls`**. If the file is NOT listed, check your root directory.[2] After you create this file and ensure that it is in the **`matlab`** directory, you then can execute the commands present in the file by simply typing

```
>> ps16a
```

On the workstations in the DEC Lab, you can also edit files while in MATLAB by opening another X-window. You can edit the files in one window while running MATLAB in the other window.

2. The file *should* be located in your current directory. If it is not, check with the user consultant in the DEC Lab

5.5. Multiple Plots in the Graphics Window

The file **`ps16a.m`** simply contains MATLAB commands. Let's create the file **`ps16a.m`** that has the following commands (be sure to enter these lines EXACTLY as shown):

```
%Generates Magnitude and Phase plots
w = 0:0.01:1;
s = j*w; F = 1 + s.^2;
Fm = abs(F);
Fp = angle(F);
subplot(211)
plot(w,Fm)
title('magnitude of F')
xlabel('freq in radians')
subplot(212)
plot(w,Fp)
title('phase of F')
xlabel('freq in radians')
```

Note that the character **`%`** is used to designate comments.

The command **`subplot()`** is used to create an $m \times n$ array of plots. This command allows you to view up to 4 plots in a 2×2 array in the same graphics window. **`subplot()`** is particularly useful for plotting magnitude and phase functions, as the y-axis scales needed for the two plots may be significantly different. For more information on this command, simply use the help function:

```
 >> help subplot

SUBPLOT Graphing window control. SUBPLOT(mnp), where 'mnp' is a
three digit number, breaks the graph window into an m-by-n
matrix of small graph windows, and selects the p-th window
for the current plot. For example,

SUBPLOT(211), PLOT(1:3), SUBPLOT(212), PLOT(SIN(1:3))

plots 1:3 on the top half of the screen and SIN(1:3) on the
bottom half.  CLG or SUBPLOT(111) returns to the default
single-plot configuration.
```

Now to execute the sequence of commands contained in the file **`ps16a.m`**, simply enter

```
>> ps16a
```

The magnitude and phase plots should appear one above the other in the graphics window. In this course it will also be useful to generate log-log and semilog plots. In MATLAB these types of plots can be generated using the **`loglog`**, **`semilogx`**, and **`semilogy`** commands in a manner similar to the **`plot`** command. Use the on-line help to find out more about these commands.

5.6. Matrix vs. Scalar Operations

This example also can be used to illustrate another important point- that of matrix and vector operations as opposed to point-wise or scalar operations. Now that you have run **`ps16a`**, the vector s is still defined. Now see what happens when you enter the following command:

```
>> F = 1 + s^2
??? Error using ==> ^
Matrix must be square.
```

This error occurs because **`s`** is a vector and there is no operation that is defined as “squaring” a vector – a scalar or a square matrix can be squared, or each element of a square matrix can be squared. Operations on Vectors (or non-square matrices) are either defined as inner or outer products. The difference between this command and the similar line in the script file **`'ps16a.m'`** is that here the power operation is preceded by a period, i.e. **`.^`**. This notation denotes element-by-element operations. Thus the command

```
>> F = 1 + s.^2
```

is equivalent to saying that $F(n) = 1 + s(n)^2$, where $F(n)$ and $s(n)$ are the n^{th} elements of the F and s vectors, respectively.

5.7. On–line Help for M-files

A final important point: Comments are very useful in MATLAB. Once you create an m-file in your MATLAB directory, the opening comments are displayed if you enter **`help fn`** in response to the MATLAB prompt. For example, now type

```
>> help ps16a

Generates Magnitude and Phase plots
```

Note that MATLAB responds with the comment used in the script file. This feature is very useful for keeping track of your own script and function files. For more on these M-files, see section 12 in the Primer.

A very nice help program is available on the Macintoshes by hitting the help key.

6. Concluding Remarks

You are now ready to begin to use MATLAB more extensively. Be sure to read the MATLAB Primer as we have not covered everything in this simple tutorial. Section 12 is particularly important to learn more about function M-files. We did not introduce these types of M-files in this introduction, but you will find it useful to create and save your own MATLAB commands as the semester progresses. Also you should use the on-line help facility to find out more about the additional commands listed on pages 22-35 in the Primer.