Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2007
Physics 351

MATLAB: A Brief Introduction

DUE DATES FOR EXERCISES P4-P6:
P4: Friday, Oct. 5, 2007; 2.30 pm.
P5,6: Friday, Oct. 12, 2007; 2.30 pm.

There are many programming languages available. In this course we'll use **MATLAB**, a programming language and "environment" that is used by many scientists and engineers (including the Parthasarathy Lab). *If you're already familiar with some other language, e.g. C++, Java, IDL, etc., feel free to use it*, but don't ask Professor Parthasarathy for help with your programs. MATLAB is easy to learn, has many good built-in functions, and is especially powerful for calculations involving matrices (arrays of numbers), hence its name.

The rest of this document is a very brief introduction to some aspects of MATLAB. It should be read as a companion to "experimenting" with MATLAB, not as a substitute. The best way to learn to program is to dive in and try it! In this document, MATLAB commands are indicated by **`boldface Courier text`**.

1 MATLAB Availability

The Physics Department has licenses of MATLAB for student use installed on computers in 17 Willamette Hall. All Physics 351 students have access to this room – I'll give out the key code in class. You **do not** need to purchase MATLAB for this course. (If you would like to, the student version is $99, I believe.)

In addition, several "freeware" programs that mimic the basic structure of MATLAB are available. One of these is **FreeMat**, which can be downloaded at http://freemat.sourceforge.net . I have used FreeMat very little, but the course teaching assistant or I will check that my assignments are compatible with it, or comment on any differences between FreeMat and MATLAB commands. We have found so far one notable difference between FreeMat and MATLAB, which will be described in Section 7 below. Another popular free MATLAB mimic is **Octave**; I have never used it.

2 MATLAB's environment

Upon starting MATLAB, you should see the "Desktop" and "Command Window." In the command window you'll see the "prompt," which looks like ">>." You can type commands here. It is convenient to write programs ("m-files," described below) using MATLAB's editor, which you can open by typing "edit." (In FreeMat, you can open an editor window under Tools / Editor in the menu bar.) To run a program, you can hit F5 from the editor, or type the program name (without the .m extension) from the command window. (Be sure to be in the correct directory or to put the program directory in MATLAB's "path." I'll write more about this below.) To exit MATLAB, type "quit" at the command prompt, or use the menus.

3 MATLAB's tutorials

You may wish to start by examining some of MATLAB's built-in tutorials. (In general, MATLAB's built-in help features are very good.) Go to "Help / Full Product Family Help" to open the Help Window, if it's not open already. Click 'begin here,' and you should see a window with "What is MATLAB?", "Matrices and Arrays," etc. Go through a few of these, typing each displayed command in the Command Window.

In general, you can find information on any function by entering it in the Help window, or by typing "help [function]" from the Command window.

Also, "Wikibooks" has an entry on MATLAB programming. Overall it's not very good, but some parts may be useful – see the sections on "Data storage and manipulation" and "Graphics." The URL is: http://en.wikibooks.org/wiki/Programming:MATLAB .

4 Elementary operations

As with nearly all programming languages, *variables* store numerical values.

Type at the ">>" prompt:

`a=3  [Enter]`

"[Enter]" means press the Enter key. Matlab will respond: "*a = 3*". Enter **`a+2`** – MATLAB will respond "*5*." Enter **`a+3`** – MATLAB will respond "*6*" – note that the value assigned to *a* has not changed from the "3" we assigned it. Type **`a = a + 4`** – MATLAB will respond "*7*". Now what will happen when you enter **`a+3`**? **Try it.**

Next, define a new variable b by **`b = 2*a`**. The "*" indicates multiplication. You should find that *b = 14*. Next, enter **`c = 3*a;`** *including the semicolon at the end.* MATLAB sets *c* to equal 3 times *a*, but does not display the value on the screen. Enter "**`c`**" by itself to display the value.

Matrices are arrays of numbers. The number "3" is a 1x1 matrix. Create a 1 (row) x 4 (column) matrix containing the numbers 1, 2, 5, and 3.1 by entering at the ">>" prompt:

`d = [1 2 5 3.1]`

Matlab will respond: "*d = 1 2 5 3.1*," and you can note from the workspace window, or by typing "**`whos`**" ("**`who`**" in FreeMat) that there is a variable called "d" that is a 1x4 matrix of numbers. Now type:

`e = 2*d - 3;`

Note the semicolon. As above, if you want to see the value of *e*, type **`e`**.

Now type each of the following lines, and observe what happens:

`e(3)`

`e(4)`

`f = d.*e`

This last operation, " .* " is very useful – the period in front of the multiplication symbol indicates an element-by-element multiplication of the two arrays, rather than a matrix multiplication. Both factors must be the same size, and the product also has the same size. This element-by-element multiplication is usually what we want. Try:

`d*e`

Note that this last command gives an error message – if you're familiar with matrix algebra, you should see why. If you're not, don't worry about it – we won't make use of matrix algebra in our course.

`g = (1.1 + 2.0)*[1:10]`

The colon creates an array whose elements are evenly spaced between the endpoints, 1 and 10; the command multiplies this by 3.1. Note the parentheses.

`h = 1:4:38`

The colon creates an array whose elements are evenly spaced by 4.

`g(3:6)`

Hopefully you've deduced from the lines above that numbers in parentheses following a variable name select those elements of the array. We can use a colon to select a particular range of array elements.

5 A list of operators (from "Help")

(For your reference)

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`\`	Left division [*We won't use this*]
`^`	Power
`'`	Complex conjugate transpose [*We won't use this*]
`( )`	Specify evaluation order
`.*`	Element-by-element multiplication
`./`	Element-by-element division
`.\`	Element-by-element left division [*We won't use this*]
`.^`	Element-by-element power
`.'`	Unconjugated array transpose [*We won't use this*]

6 if-then statements

We often need to perform various logical operations. For example, we test whether some condition is met. Predict beforehand the result of the following code, then try running it (typing each line). The command "disp" displays a "string" (an array of characters).

```
if (g(3)>5)
   disp('happy');
else
   disp('sad');
end
```

7 for loops

A "for" loop increments an "index" variable over some range. Try the following:

```
for k=1:5,
   k
   m(k) = k/5;
end
```

(If you're using FreeMat, you'll get an error message – see the footnote[1].) The items between "for" and "end" are repeated five times, each time with a different (displayed) value of k. Note that the semicolon will prevent the value of m from being displayed. What do you suppose the array "m" looks like? Examine it, by typing: "`m`". Next, try the following, first trying to predict what the result will be:

```
n = 1:10;
n
for k=1:5,
   n(2*k) = 10-k;
end
n
```

[1] Unlike MATLAB, FreeMat strangely requires that lines in *For* loops be "statements" assigning values to variables, e.g. "`a=3`." Therefore we can't just type "`k`" to see the value of k. We can, however, type "`k=k`" – this assigns to k the same value it already has, and displays it.

The index variable doesn't have to run sequentially from 1 to its end. For instance, the loop in the previous example could have been written

```
for k=2:2:10,
   n(k) = 10-k/2;
end
```

with the same result. (Try it.)

8 Plots

MATLAB is both powerful and convenient for making plots. Let's plot x^2-3x as a function of x over the range x = -2 to 3 in increments of 0.1.

```
x = -2:0.1:3;
plot(x, x.*x - 3*x, 'ro-')
```

"ro-" means the plot color is red ("r"), points are indicated by circles ("o"), and a line ("-") connects the points.

```
hold on
```

"hold on" means keep this plot in this figure window when we add another plot, this time of y = 0.2x^3 - 4x^2 +10 as a blue dotted line:

```
y = 0.2*x.^3 - 4*x.*x + 10;
plot(x, y, 'b:')
xlabel('x')
```

This last command labels the horizontal axis. "ylabel" does the same for the vertical axis, and "title" adds a title.

Note that we could also have calculated y, less elegantly, with a *for* loop:

```
for j=1:length(x)
     y(j) = 0.2*x(j)^3 - 4*x(j)*x(j) + 10;
end
```

The command "length" returns the number of elements in x.

9 User inputs

Many sorts of user inputs can be constructed, including "push-button" user interfaces. The simplest input uses the "input" command. Try:

```
z = input('Enter your favorite number: ');
z
```

10 m-files

Instead of typing at the command prompt, we write programs and functions – text files of MATLAB commands. Save these as [filename].m .

It's important to keep track of the directory you're working in, which you can see from the top of the command window, or by typing "**`pwd`**". You can change directories by "**`cd '[directory]'`**" or by using the menu bar at the top of the command window. When looking for programs, MATLAB will look in the present directory and in the "path," which can be set from the File Menu ("Set Path"). One of several ways to list all the files in the present directory is "**`dir`**".

m-files contain MATLAB commands and also *comments*, which are not executed but which are very important in making the program readable. Comments are indicated by a "**`%`**" – MATLAB ignores everything on a line that follows a "**`%`**".

Type and save the following, which calculates $\sum_{j=0}^{N} \frac{1}{2^j}$, as an m-file called "**simpleseries.m**":

```
% simpleseries.m
% Raghuveer Parthasarathy
% April 19, 2007
```

```
% A series which should quickly converge

nloops = input('Number of iterations: ');
s = 0.0;
for j=0:nloops,
     s = s + 1/(2^j); % increment "s" at each iteration
end
disp(s)
```

Run the program by typing **simpleseries** at the command prompt. Run it for 3, 10, and 50 iterations.

11 Misc.

Two additional useful commands: "**clear all**" clears the values of all the variables, and "**close all**" closes all the open figure windows.
Control-C will abort whatever process is running – useful, for example, if you've set up some slow or infinite loop.

Exercises

(P4, 5 pts.**) Reading.** Start using MATLAB or FreeMat – find an appropriate computer. Read the above text, enter all the MATLAB commands, examine the output, and make sure you can get all the commands work as they should. Send Prof. Parthasarathy an email that says that you've done this. You don't need to send printouts or other records of the output.

(P5, 6 pts.**) Sine wave.** Recall that a sinusoidal oscillation has the form $x(t) = A\sin(\omega t - \phi)$, where the period $T = 2\pi/\omega$. (MATLAB, by the way, has a pre-set value of pi called, appropriately, "**pi**".) Consider an oscillator with a period of 2 seconds, an amplitude of 1 meter, and a phase offset ϕ=0. Plot x vs. t for 5 cycles of oscillation, and label the axes. *Hints:* Choose an appropriately dense array of time points – entering "**t=0:20**" is bad, and "**t=0:2:20**" is even worse – *think about why*. You can make a "for" loop to calculate the x value corresponding to each t value, or just calculate the entire "x" array in one step. Plot x and t. If you're stumped, first do this exercise by hand, i.e. without a computer, and then return to MATLAB.

(P6, 7 pts.**) Modifying "simpleseries" – partial sums of a simple series.**
(a, 4 pts.**)** Modify *simpleseries.m* (above) to calculate and plot the partial sum of its series at each term, up to 31 terms. Hand in your plot. *In more detail:* By "partial sum" we mean the sum at each "step" of adding an extra term to the series. Instead of "s" being just one element, make it an array, setting **s(1) = 0.0;** before the "for" loop. If you still make the for loop run from 0 to *nloops*, it should set s(2) to equal s(1) + $1/2^0$ when j=0, s(3) to equal s(2) + $1/2^1$ when j=1, s(4) to equal s(3) + $1/2^2$ when j=2, etc., up to the final, 31st term (j=30). Plot the output using either "**plot(s)**", or "**plot(1:31, s)**".

(b, 2 pts.**)** The series $\sum_{j=0}^{N} \frac{1}{2^j}$ is a simple geometric series, whose sum you should know how to analytically calculate. (Yes, I expect you to have learned things in your math classes.) For N=∞, what is the sum?
(c, 1 pt.**)** How many terms of the sum did your program need before s was within 10% of its asymptotic limit of part (*b*)?