COSMOS: Making Robots and Making Robots Intelligent

Lecture 2: Intro to MATLAB® *

Jorge Cortés and William B. Dunbar

March 6, 2006

Abstract

MATLAB® [1] is an interactive, numerical computation and graphics program. MATLAB® will serve as a tool for illustrating some of the mathematical concepts and solutions that we will see throughout the course. Think of MATLAB® as the ultimate calculator!

This session begins once one has figured out how to start up MATLAB®, and one is sitting in front of a computer screen on which the MATLAB® command window is displayed. This has a prompt "≫" and a cursor awaiting instruction.

Contents

*Part of the exposition in this lecture builds upon teaching material developed by Neil Balmforth and Hongyun Wang at the Department of Applied Mathematics and Statistics, University of California, Santa Cruz.

[1] You can download from http://www.mathworks.com/access/helpdesk/help/techdoc/Matlab.html the useful reference "Getting Started with MATLAB®".

1 Create a Working Directory

The first thing you should do is create a working directory on the machine at your desk. Once the machine is on and you are logged in, there should be a shortcut on the desktop that will open the D: drive. Double click this shortcut to show the contents of the D: drive. You will see a folder called "Cosmos Files." Double click this folder to open it. Once open, the easiest way to create a sub-directory would be to right click inside the folder, hover the mouse over "New..." and in the resulting dialog box, select folder. This will create a sub directory within that the "Cosmos Files" folder, that you can name as you wish. Clicking on the name of the folder once will allow you to rename it, and we suggest renaming the folder with your last name. If your name is *Lisa M. Simpson*, then name this new folder "Simpson." Now, within your folder, create another sub-directory with the name "matlab." This directory is where you should keep all your work.

Task 1.1 (Start MATLAB® and go to your Working Directory) Double click the MATLAB® icon on the desktop to start MATLAB®. When the command window opens and you see the command prompt ≫, type

≫ cd D:\Cosmos Files\(your last name)\matlab

From now on, at the beginning of every lecture, begin by following this procedure of getting into your working directory. Also, all MATLAB® related files that you save should be kept in this directory.

2 Help!

MATLAB® has an extensive help facility and a number of demonstration programs. The command "help (*function-name*)" is the syntax. For example

≫ help cos

provides a description of the cosine function "cos(·)". The help facility is also divided into categories, and a list of categories can be found by typing "help" alone with no function-name argument. To access a list of functions under a category, simply type "help (*category-name*)". For example,

≫ help elfun

produces a list of the elementary functions that are available in MATLAB® (and each is described by executing help again with the relevant function-name).

Task 2.1 (Help Yourself) Type "help" for the list of categories, choose a category (we suggest "elfun"), and type "help (*category-name*)" for the chosen category-name. Within the list of functions, pick one, and type "help (*function-name*)" for the chosen function-name. Given the help information, try to implement the chosen function.

Task 2.2 (Built-in Demonstrations) Type "demos" for the demonstrations. These are often more complicated than you bargained for, *e.g.* check out the cool "peaks" and "modes", which give some idea of the graphics that MATLAB® is capable of.

3 Mathematics with MATLAB®

3.1 Numbers and operations

In MATLAB®, numbers are what you expect: 12.4 entered after the prompt, followed by a carriage return ("Enter"), means exactly that to MATLAB®:

$$\gg \quad 12.4 \quad (\text{"Enter"})$$
$$\text{ans} = \quad 12.4$$

with answer, being the number 12.4 abbreviated as "ans" by MATLAB®. MATLAB® won't do anything other than interpret the command as a number unless you tell it to do more. Note that "Enter" tells MATLAB® to execute the command contained on that line.

Simple arithmetic operations are also straightforward:

$$\gg \quad 12.4 * (48.5 + 342/39) \quad (\text{"Enter"})$$

means $12.4(48.5 + \frac{342}{39})$, and produces the result:

$$\text{ans} = \quad 710.1385$$

The arithmetic operations recognized by MATLAB® are:

$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
$\wedge$	raise to power

(From now on, we will omit the "Enter" which executes the command.)

3.2 Output

If you get sick of MATLAB® always printing the output "ans = ...", you can suppress printing the output of a particular command by ending the line with a semi-colon ";" so

$$\gg \quad 6;$$

outputs nothing, and

```
≫  x = 6;
```

assigns the number 6 to the variable x without printing the output. We'll learn more about assigning values to variables in the next section.

MATLAB® also uses customary scientific notation (is anyone unfamiliar with scientific notation?):

```
≫  cos(1.57)
ans =  7.9366e−04
```

which means $7.9366 \times 10^{-4} = 0.00079366$.

MATLAB® lets you know if you screwed up too. Suppose the variable y has no value assigned to it. Then, you can verify that the follow occurs:

```
≫  cos(y)
Error :  Value of y unassigned.
```

(Some versions were also programmed to respond to rude comments.)

3.3 Variables

You can create variables in MATLAB® by assigning numerical values to variables names:

```
≫  x = 3
x =  3
```

assigns the value of 3 to x, and MATLAB® remembers this until x's value is re-assigned by another command. Hence, for example,

```
≫  2.5 ∗ x
ans =  7.5
```

Also, simply typing the variable name produces the value MATLAB® thinks it has:

```
≫  x
x =  3
```

Variables can be assigned to the results of computations. For example,

```
≫  y = (2 + x) ∧ 3
y =  125
```

assigns the value $(2 + x)^3 = 5^3 = 125$ to y.

Some variables are already assigned in MATLAB®. For example, pi stands for $\pi = 3.14159....$ So

```
≫  z = 2 ∗ pi + 6
z =  12.2832
```

3.4 Mathematical functions

MATLAB® has a large number of built-in mathematical functions. Some of them are:

abs(x)	absolute value of x, $\|x\|$
sqrt(x)	square root of x, $\sqrt{x}$
sign(x)	sign of x
exp(x)	$\exp(x) = e^x$
log(x)	$\ln(x) = \log_e(x)$
log10(x)	$\log_{10}(x)$
sin(x)	$\sin(x)$
cos(x)	$\cos(x)$
tan(x)	$\tan(x)$
asin(x)	$\sin^{-1}(x)$
acos(x)	$\cos^{-1}(x)$
atan(x)	$\tan^{-1}(x)$

MATLAB® assumes that the argument is expressed in radians for the trigonometric functions. e.g.

```
>> x = pi
x =
    3.1416
>> z = sin(x)
z =
    0
```

3.5 Vectors

MATLAB® handles vectors also. For us, a vector will just be a finite, ordered sequence of numbers, separated by commas. For example, consider the vector $x = (1, 3, 2, 4)$. In MATLAB®, you write this like

```
>> x = [1, 3, 2, 4]
x =
     1     3     2     4
```

The numbers in the vector are referred to as elements of the vector, so 3 is the 2^{nd} element of the vector x in this example. You can access the i^{th} element of a vector y by typing "y(i)". In the example, then,

```
>> x(3)
ans =
     2
```

When typing the vector, the commas separate elements, whereas MATLAB® outputs the vector with spaces between elements rather than commas.

Amazingly, MATLAB® can handle arithmetic defined on the elements of the vector. For example, let's take the vector $x = (1, 3, 2, 4)$ and multiply every element by 5:

```
>> 5 * x
ans =
     5    15    10    20
```

The output is another vector with the each element multiplied by 5. Now suppose we wanted to square each element of x:

```
≫   x.∧2
ans =   1   9   4   16
```

For exponential operations on vectors, you need to add a period before the "∧" symbol, so that MATLAB® knows that it must perform the operation element-by-element. There are other operations where this is also the case. Typically, if you try an operation without the period and it turns out that the period is required, an error will be printed indicating that you need the period.

Task 3.1 With the artifice of the period, we can also multiply the elements of two vectors together if they have the same length. For instance, multiply the vectors $x = (1, 2, 3, 4)$ and $z = (17, 12, 23, 17)$. If you get an error from matlab, re-check that your are using the period correctly.

The period is not necessary for sums and subtractions

```
≫   x+z
ans =   5   5   5   5
≫   x−z
ans =   −3   −1   1   3
```

We can sometimes apply functions on each element of a vector by simple operations on the vector as a whole: the exponentiation $e^{(\cdot)}$ of each element in x is obtained by

```
≫   exp(x)
ans =   2.7183   7.3891   20.0855   54.5982
```

and the sine of each element is

```
≫   sin(x)
ans =   0.8415   0.9093   0.1411   −0.7568
```

Now, a useful shorthand way to generate the vector x is

```
≫   x=[1:1:4]
x =   1   2   3   4
```

The notation $p : n : q$ instructs MATLAB® to assign the ordered list of elements that starts with p, ends with q, and increments in steps of size n.

Task 3.2 Construct a vector z vector whose elements are the even integers from 2 to 1000 using the previous construction (try it with and without the semi-colon). Once you have constructed z, construct the vector $x = z/1000$.

4 Plots

Finally we are ready for some simple plotting. Let $y = \sin(2\pi x)$, where x is the long vector constructed above in Task 3.2. We may plot x against y with the simple plot command:

$$\gg \quad \text{y} = \sin(2 * \text{pi} * \text{x});$$
$$\gg \quad \text{plot}(\text{x}, \text{y})$$

We can plot more than one curve at the same time:

$$\gg \quad \text{plot}(\text{x}, \text{y}, \text{x}, \cos(2 * \text{pi} * \text{x}))$$

Note how this plots a function directly, without first creating a new vector like y.

We can even change the style of the curves:

$$\gg \quad \text{plot}(\text{x}, \sin(2 * \text{pi} * \text{x}. * \text{x}), '--', \text{x}, \cos(2 * \text{pi} * \text{x}. * \text{x}), ':')$$

which plots the sine curve with a dashed line and the cosine curve with a dotted one. The three plots are shown in Figure 1. More plotting styles can be found by executing "help plot".

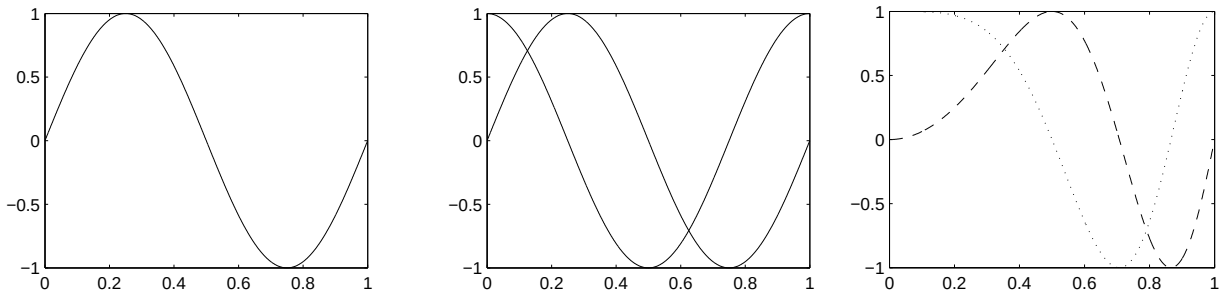

Figure 1: Sample plots

Task 4.1 (Plotting the evolution of the US population) Let us start using what we have learned so far by playing with the US Census data from 1900 to 2000, as depicted in the following table:

Year	Population (mil)
1900	75.995
1910	91.972
1920	105.711
1930	123.203
1940	131.669
1950	150.697
1960	179.323
1970	203.212
1980	226.505
1990	249.633
2000	281.422

Define two vectors, t and p. The vector t should contain the years in the first column (remember the shorthand way to generate vectors in Task 3.2), and the vector p should contain the second column with the population for each year. Then, use the plot command to plot the evolution of the population throughout the century. Put the title "Population of the U.S. 1900-2000" to your plot, and the label "Millions" in the y-axis. You should get something that looks like Figure 2. Try to get the circles to show up at the data points on the plotted line also.

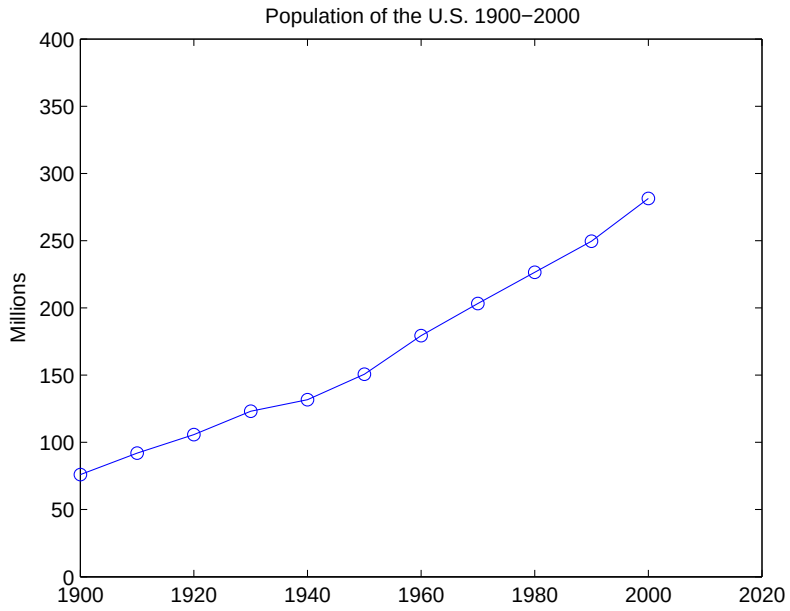

Figure 2: US population 1900-2000.

5 Programming

5.1 Loops

In some calculations it is useful to do things recursively. We may repeat commands by placing them in a loop; the construction is as follows. We choose a loop variable n and then increments n from 1 to N:

```
≫   for n=1:N
blah, blah
end
```

"blah, blah" denotes the commands we want to repeat N times, and "end" terminates the loop. The "blah, blah" commands can be just about anything you want. For example, let us say we want to plot a the function $f(x) = 1 - ax^2$ 9 times, each time with a different value for the parameter a. Using 'for' we can do it as follows:

```
≫ x=[0:100]/100;
≫ a=[1.1,1.2,1.4,1.6,1.9,2,2.5,3,4]/4;
≫ plot(x,x)
≫ for n=1:9
fx=1-a(n)*x.^2;
hold on, plot(x,fx)
end
≫ hold off
```

The initial plot command "plot(x,x)" gets everything going here (see Figure 3, left). The instruction "hold on" makes sure that previous plots are not erased by plotting the current one.

After typing these commands, you should get something similar to Figure 3, right.

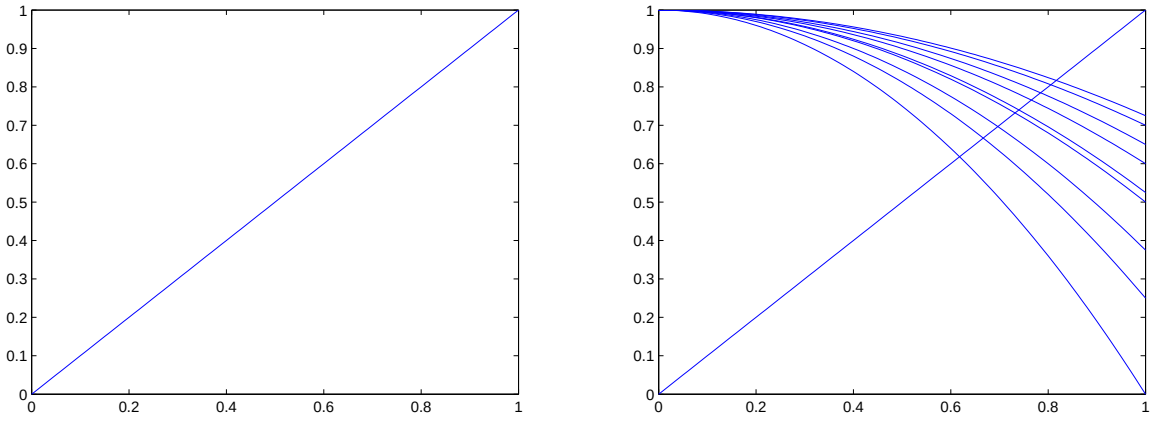

Figure 3: Plot of the function $f(x) = x$ (left) and $f(x) = 1 - ax^2$ (right) for different values of the parameter a.

If you already have previous knowledge of a programming language, the idea of the loop should be familiar; what's different is the MATLAB® format and the fact that it can deal with plotting within the loop.

Task 5.1 Consider the rule

$$x_{n+1} = \cos x_n$$

If we start with x_0 and use the rule, we get $x_1 = \cos x_0$. Now, using the rule again, we can compute $x_2 = \cos x_1$. And so on. Use the command 'for' to compute the value of $x_1, x_2, \ldots, x_{10}$ starting with $x_0 = \pi/6$.

5.2 m-files (not to be confused with the popular X Files :-)

Frequently, it becomes tedious to keep on repeating the same sets of commands, or re-entering a bunch from a previous MATLAB® session. A far more efficient approach is to place the MATLAB® commands into a text file that we call something like `rubarb.m` (rubarb is, of course, not the only word one can use here, but the .m is essential). The format should be identical to how the commands would appear on the MATLAB® command line. Whenever we want to execute the commands contained in the "m-file", all we need to is enter "rubarb" on the command line and MATLAB® will read and execute them.

To create, edit and modify an m-file, you need to be able to open the text file with a suitable editor. There is an editor in MATLAB® that you can use, but any other editor is also acceptable. To call it up, simply type

$$\gg \quad \text{edit}$$

This provides you with another window containing the editor; in this window you can happily type commands. You must save the file before you can execute the commands in it.

Task 5.2 Write an m-file that when invoked from the MATLAB® command window automatically generates the plot in Figure 2.

Using m-files, you can define your own functions. For example, typeset the following lines in your editor and name the file `myfunction.m`,

$$\begin{array}{l} \text{function output} = \text{myfunction(x)} \\ \text{output} = \exp(x.\wedge 2).*x; \end{array}$$

The first word, 'function', tells MATLAB® that we are defining a function. The name of the new function is 'myfunction'. It takes as argument the variable x, and spits out 'output' - which is defined in the second line (what would have happened if we do not use the period?). Much more complicated function definitions are possible.

Task 5.3 Once saved, you can invoke the function `myfunction` from the MATLAB® command window. For instance, `myfunction`(2) should give you 109.1963. Plot this function from -1 to 1 using an increment step size of 0.1.

Task 5.4 Let us combine our knowledge of the instruction 'for' with the definition of new functions. Create a new function (call it `until`) that takes as an argument an integer n and spits out the vector $x = (1, 2, 3, \ldots, n)$.

6 Homework

Make sure you have completed all tasks in this lecture. Re-read the notes and let us know if there is anything you like us to revise in the next lecture.