Getting started with Matlab

J.M.Maciejowski
Cambridge University Engineering Department

October 13, 1990. Updated in 2000-2006 by Tim Love

1 Introduction

Although the name 'Matlab' comes from 'Matrix Laboratory', Matlab can be used as a very powerful calculator for all kinds of engineering, scientific and other calculations. It is not necessary to know much about matrices to use Matlab, though the more you know, the more effectively you can use it.

This document is intended to show you just how easy it is to use Matlab, and to give you some idea of what it can do. If you like what you see, you can find out much more by reading the online Tutorial[1]. You can also use the on-line 'help' system, as we will show later.

Before starting on some simple examples, we should mention that Matlab is much more than a 'calculator': as well as handling vectors and matrices, it can work with complex numbers, generate 2-D and 3-D figures, and it has the full power of a general programming language.

This document is intended to be read at one of the DPO workstations. Try everything as you read it. It should take you less than an hour to get through it.

Note that `octave` (a free program similar to `matlab`) is available on the DPO terminals and the MDP Resource CD[2]

2 Starting and stopping

Log in to a DPO terminal. From the `Start` menu, choose the `Programs` submenu and select the `Matlab` program. This will get Matlab started, and when you see the >> prompt Matlab will be waiting for input from you.

When you want to leave Matlab, type `quit` or `exit`. If you want to stop before completing something, simply type `save` before stopping, and all your current workspace will be saved to a file (called *matlab.mat* by default). When you want to continue just enter Matlab again (from the same directory as before) and type `load`. The file will then be read, your workspace restored, and you can go on from the point at which you stopped.

3 Basic use

First let's use Matlab just like a calculator. Type

```
1.2 * sin(3.4^2 + log10(5))
```

to see the value of $1.2 \times \sin(3.4^2 + \log_{10} 5)$. To keep this value for later use, calling it by the name 'result' (for example) type

[1] http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml
[2] http://www-mdp.eng.cam.ac.uk/CD/

```
result = 1.2 * sin(3.4^2 + log10(5))
```

Now you can use this value in a new calculation:

```
new = 3+result/2
```

(Note that this gives $3 + (\text{result}/2)$, not $(3 + \text{result})/2$; you can always use round brackets to specify exactly what you mean: `newnew = (3+result)/2`).

Matlab knows about complex numbers. You get $\sqrt{-1}$ by typing `sqrt(-1)`, but since most people use i to represent $\sqrt{-1}$ you can get it more easily by typing `i`. (To allow for the peccadilloes of electrical engineers you can also get it by typing `j`.) Try typing

```
exp(i*pi)
```

to check that Matlab also knows about the exponential function and π. (You have just evaluated $e^{i\pi}$.) You can enter complex numbers either in terms of real and imaginary parts ($z = x + iy$), or in terms of modulus and argument ($z = re^{i\theta}$). Try typing

```
z = 3 + 4*i
```

and

```
z = 5 * exp(i*0.9273)
```

Note that in both cases Matlab's response is in $x + iy$ form, and that the argument (θ) must be specified in radians, not in degrees. You can recover the real part, imaginary part, modulus (absolute value), and argument by typing `real(z) imag(z) abs(z)` and `angle(z)` respectively. To find $|z^2|$, for example, type `abs(z^2)`.

Warning: If you should type something like `i = 10` then the value of `i` will be redefined and will no longer be $\sqrt{-1}$.

One of the most useful features of Matlab is that functions like `sin`, `log`, `abs` and many others will work on whole lists of numbers simultaneously. Suppose, for example, that we wanted to evaluate the list

$$\sin\left(-\frac{\pi}{2}\right),\ \sin 0,\ \sin\left(\frac{\pi}{2}\right),\ \sin\pi$$

for some reason. We could do this by typing

```
sin([-pi/2, 0, pi/2, pi])
```

Notice that we used `(` and `)` to enclose the argument to the function `sin`, but `[` and `]` to enclose the list of numbers. If you make your list of numbers vertical instead of horizontal, then the answer comes out vertical too. Try:

```
vlist = [-pi/2; 0; pi/2; pi]
sin(vlist)
```

So, between `[` and `]` commas are used to put things beside each other, while semicolons put things on top of each other. (You can also use a space instead of a comma, and a line-break ('Return' key) instead of a semicolon — try it.)

In fact functions like `sin` will work on two-dimensional arrays of numbers, and the result will have the same shape as the original array. To find

$$\begin{bmatrix} \log_e 1 & \log_e 2 & \log_e 3 \\ \log_e(-1) & \log_e(-2) & \log_e(-3) \end{bmatrix}$$

type:

```
try1 = [1, 2, 3 ; -1, -2, -3]
log(try1)
```

4 Vectors and matrices

You have already started working with vectors and matrices. The array `try1` is an example of a *matrix*, with 2 *rows* and 3 *columns*. The horizontal list `[-pi/2, 0, pi/2, pi]` and the vertical list `vlist` are both examples of *vectors*; these can be regarded as special cases of matrices — a matrix with one row is called a *row vector* and a matrix with one column is called a *column vector*.

Matlab has lots of ways of building up matrices, and of extracting bits of matrices. For example, if you type

```
try2 = [try1 ; 2*try1]
```

you get a matrix with 4 rows and 3 columns. To pick out the number which is in row 3 and column 2 of `try2` you type

```
try2(3,2)
```

To pick out columns 1 and 3 of `try2` type

```
try3 = try2(:,[1,3])
```

`try3` is now a matrix with 4 rows and 2 columns. Here the colon `:` was used as an abbreviation to mean 'all the rows', so that we didn't have to type `try2([1,2,3,4],[1,3])`.

We can do arithmetic with matrices, but we have to be careful about their sizes. Addition and subtraction operators (`+` and `-`) can be used between any two matrices, providing that they have the *same numbers of rows and columns*. The meaning is that corresponding elements are added or subtracted. For example

```
gross_weights = [12.3 23.4 34.5]
crate_weights = [2.3 3.4 4.5]
net_weights = gross_weights - crate_weights
```

Multiplication and division of corresponding entries can also be obtained, using the operators `.*` and `./` respectively. Note the '`.`' here. For example

```
prices = [1 2 3 4]
quantities = [10 20 30 40]
item_costs = prices .* quantities
total_cost = sum(item_costs)
```

If you have not seen matrix multiplication or vector scalar products before, you can skip to the end of this section; all the material needed to read the rest of this section is covered in the first year Maths course.

The operator `*` (without the dot) is used for proper matrix multiplication. Recall that if matrix `A` has m rows and n columns, and `B` has p rows and q columns, then the product `A*B` is defined only if $n = p$, and the resulting matrix will have m rows and q columns. In particular, if `A` is a row vector ($m = 1$) and `B` is a column vector ($q = 1$) then the result is just a scalar number, and is in fact the scalar (or 'dot', or 'inner') product of the two vectors. For example:

```
weights = [3 1 7]
locations = [-2 0 5]'
cg = (weights * locations) / sum(weights)
```

Note the use of the transposition operator (`'`) here to make `locations` into a column vector (`locations = [-2;0;5]` would have done just as well). Do you see why the centre of gravity is given by using the scalar product in this way? Once you get used to it, lots of calculations can be written in this form. Matlab is particularly efficient at performing scalar product calculations.

Finding inverses of matrices is easy in Matlab. To find the inverse of a random 4×4 matrix:

```
M = rand(4,4)
Minv = inv(M)
```

To find AB^{-1} we can write `A*inv(B)`, but Matlab also allows us to write `A/B`, and to find $B^{-1}A$ it allows `B\A` as well as `inv(B)*A`. (Note that this use of `/` and `\` is not standard notation in linear algebra. Using these operators is not quite equivalent to the use of `inv`, and is in fact preferable on grounds of numerical accuracy — see the Matlab User's Guide for more details.)

There are many more fancy operations on matrices available in Matlab. To find the eigenvalues of `M`, for example, type:

```
eig(M)
```

If you want both the eigenvalues and eigenvectors, type:

```
[W,D] = eig(M)
```

In this case `W` is a matrix whose columns are the eigenvectors, and `D` is a diagonal matrix, whose diagonal entries are the corresponding eigenvalues. Check that `M*W` is the same as `W*D` — the smart way of doing this is to type `M*W - W*D`.

5 Polynomials

Matlab stores polynomials as vectors. For example $2x^2 - 3x + 1$ can be represented as

```
p = [2 -3 1]
```

Several operations on polynomials are available. To find the roots of `p`, for example, just type

```
roots(p)
```

6 Suppressing and controlling output

Matlab is frequently used with vectors and matrices containing hundreds or thousands of entries. In such cases it is a great inconvenience to have the answers written to the terminal. It is easy, however, to prevent Matlab from writing the answer to any calculation: simply terminate a statement by a semi-colon (`;`) and Matlab will execute it without writing out the result. For example, generate a vector of 100 evenly spaced points between 0 and π (remember the semi-colon, and distinguish it from the colons (`:`) which are used to define the vector conveniently):

```
t = linspace(0, pi, 100);
```

Now you can examine the first 10 values, say, by typing `t(1:10)` (without a semi-colon). If you just type `t` you will get all 100 points. We can also make a vector of the values of $\sin(t)$ for $0 \leq t \leq \pi$:

```
sint = sin(t);
```

Did you remember the semi-colon?

In engineering we often need to use 'scientific' notation, in which numbers are written in the 'mantissa $\times$ exponent' form $x.xxxx \times 10^x$. We can make Matlab output results in this form. First let's look at the values in the vector `sint` near to $t = \pi/2$, say the 49th, 50th, 51st and 52nd entries:

```
sint(49:52)
```

Now switch to scientific notation by typing

```
format short e
```

and look at those values again. You will see the values to 4 decimal places, with powers of 10 (labelled 'e' for exponent). If you would like to see the values to the full accuracy with which they are stored in Matlab, type

```
format long e
```

and look at them again. If you would like to restore the original format, type

```
format
```

on its own.

7 Graphical output

Obtaining graphical output is very easy. The basic syntax is `plot(x,y)` which plots vector `y` (ordinate) against vector `x` (abscissa). For example to plot $\sin(t)$ against t, using the vectors we have already created, type

```
plot(t,sint)
```

The plot appears in a separate graphics window. Now you can add some axis labels and a rectangular grid by using menus or by typing

```
xlabel('t'), ylabel('sin t'), grid
```

To get your graph in red rather than black, and with a title, type:

```
plot(t,sint,'r'), title('Half period sine wave')
```

To position text onto the graph using cross-hairs the `gtext` command is available. Type `help gtext` for more information.

There are many other possibilities for controlling graphical output. You can have log-log or semi-log graphs, 3-D graphs of surfaces, contour plots, you can overlay various graphs on top of each other, control the axis ranges, and so on.

You can obtain hard copy of a graph (in fact, of the current contents of the Graphics Window) by typing `print` (or `cuedpr` if you want to save paper).

8 On-line help

If you forget how to use a Matlab function, or what it does exactly, you can use the on-line help facility. At the matlab prompt try the following (one at a time):

```
help abs
help plot
help eig
```

To get fuller details you may need to consult the online Reference section of Matlab's Help Desk[3]

Typing `help` on its own will print a complete list of the "topics" (e.g `matlab/general`). To list the functions within a topic, use `help` again (e.g `help matlab/general`). The list will include the so-called 'built-in' functions which have been optimised and pre-compiled for efficiency, further functions which have themselves been written in the Matlab language (more on that shortly) and supplied as part of Matlab, and collections of Matlab functions which have been written for particular applications — mostly for signal processing, time series analysis and control engineering. There are several hundred of them altogether.

9 Programming

The following extracts give you an idea of the permissible programming constructs:

```
if all(abs(t)>0),
  lt = log(t);
else
  disp('There are zero elements in t')
end
```

If you type this in, Matlab will not do anything until you have typed the final `end`. The layout is just for readability; you could also type

```
if all(abs(t)>0), lt=log(t); else
disp('There are zero elements in t'), end
```

(but you need the comma before `end`).

Here is another example:

```
k=1;
while t(k) == 0,
```

[3] http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

```
  k=k+1;
end
disp('First nonzero entry in t is:'), disp(t(k))
```

(Note the use of == here to mean 'is equal to', whereas = means 'becomes'.)

You can write 'for loops' just as in other programming languages, though there is less need for them in Matlab, since you can do many operations on whole matrices simultaneously. In this example we shall count down from 10 to 1 by steps of -1. Note that the vector `back` is initially set to be the 'empty matrix', and how it grows on each pass through the loop:

```
back = []
for k = 10 : -1 : 1,
   back = [back, k]
end
```

10 Writing your own functions

If you put any valid sequence of Matlab statements into a file and give the file a name ending with the suffix *.m*, then Matlab will execute those statements if you type the name of the file (without the suffix). You can use any text editor to create the file. Matlab's `edit` command will try to use Matlab's own editor, which has a built-in debugger.

For example, if you have a file called *dothis.m* which contains the text

```
back = []
for k=10:-1:1, back=[back,k], end
```

then just typing `dothis` will cause the `for` loop to be executed. Such files are called *script* files.

If the first line of such a file has the form

```
function outputs = somename(inputs)
```

then it becomes a *function*. In this case `inputs` and `outputs` are formal parameters, and variables used inside the file will be local to that file, which means that they will not clash with other variables you may have used having the same names (unless these are explicitly declared to be `global`). (Matlab functions correspond to subroutines in Fortran or functions in C++.)

In an earlier section we wrote a Matlab statement for finding the centre of gravity of a set of weights, given their locations along a straight line. If you wanted to calculate centres of gravity repeatedly, then you could write a function which would perform the required calculation, and store it for future use. All you need to do is to create a file called *cofg.m* in your directory (using an editor) which contains the following text:

```
function cg = cofg(weights, locations)
% Comments are introduced by percent signs, as here.
% These initial comments will be written to the terminal
% if you type 'help cofg'. It is useful to remind the
% user how to call the function by putting in the line:
%        cg = cofg(weights, locations)

cg = weights * locations' / sum(weights);
```

and that's all there is to it! When this file exists in your directory, you can type the following in Matlab:

```
cg = cofg([3 1 7],[-2 0 5])
```

Of course it may be advisable to make your function more sophisticated, for example to check whether both input arguments are row vectors, or to make sure that it will work with either row or column vectors. (Note that this function as written here will in fact work in two or three dimensions. Can you see why the following works?

```
xlocations = [-2 0 5]
ylocations = [1 2 3]
xylocations = [xlocations ; ylocations]
cg2 = cofg([3 1 7], xylocations)
```

This gives the x and y coordinates of the centre of gravity.)

To see the whole of your function from within Matlab, type `type cofg`. Try this with some other functions, such as `type angle` or `type gradient`.

It is also possible to pick up subroutines which have been written in other languages and pre-compiled. See the Interface Guide[4] for details.

11 Getting data in and out

If you want to analyse or display a large amount of data in Matlab, or to export a large amount of results from Matlab, you will probably want to read the data from a file or send it to a file. We have already seen the commands `load` and `save`, which read and write files, respectively, but when used in their basic forms they will read or write files in a peculiar format understood only by Matlab. This is fine and efficient if the files are only going to be used by other Matlab users or programs, but of no use if you want them for other purposes.

The easiest thing to do is to issue a command such as

```
save myfile.dat sint -ascii
```

[4] http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/matlab_external.shtml

This saves the variable `sint` to a file called `myfile.dat` in the 'ASCII' format, which is understood by editors, for instance, and can be read by other software. The command

```
load myfile.dat
```

reads the file `myfile.dat` from your directory.

Consult the reference section of the User's Guide to get details of how incoming ASCII files should be laid out.

For loading moderate amounts of data (small enough to type in by hand, large enough for mistakes to be likely and irritating) you can just create a script file which defines the variables you want. For instance the file *data.m* may contain the text

```
weights = [1.23 2.34 3.45 .12 .234 .345 1.23456 32.1 234 ...
21 32 43 54 123 ]
locs = [ -2 3 12 -4 23 1.34 5432 34 56 98 67 234 53 45]
```

Typing `data` within Matlab gives the variables `weights` and `locs` these values.

12 On-line demos

To get a better idea of what is possible with Matlab, type `demo` and select one or more of the on-line demonstrations. The demos in the Matlab section introduce the basic capabilities, going slightly beyond this document, while the remaining ones show off unashamedly. There are video demos too.

13 References

Much the most complete source of information is the Matlab page[5] in CUED's help system. There you'll find further tutorials, local documentation (including a Matlab Databook[6], and more specialist material.

[5] http://www-h.eng.cam.ac.uk/help/tpl/programs/matlab.html
[6] http://www-h.eng.cam.ac.uk/help/tpl/programs/Matlab/matlabDatabook/