# Matlab Databook

Tim Love

July 24, 2006

**Abstract**

This document is aimed at 3rd and 4th years who need to revise matlab for use in coursework. Many of the examples are taken from Prof J.M.Maciejowski's *Getting started with Matlab* and Dr Paul Smith's *Tutorial Guide to Matlab*. most of the material also applies to using `octave`. Suggestions and contributions for this document are welcomed. Mail `tpl@eng.cam.ac.uk`.

## Contents

## 1 How to avoid needing this document

Matlab has online tutorials and easily accessible lists of routines. Also CUED keeps a list of articles written here and elsewhere. So consider the following web links before reading further, especially if you're reading this document on paper.

- Getting Started[1]
- Using MATLAB[2]
- Mathematics[3]
- Graphics[4]
- 3-D Visualisation[5]
- MATLAB Functions Listed by Category[6]
- MATLAB Functions Listed Alphabetically[7]
- CUED's matlab page[8]

[1]http://www.mathworks.com/access/helpdesk/help/techdoc/learn_matlab/learn_matlab.shtml
[2]http://www.mathworks.com/access/helpdesk/help/techdoc/intro_tutorial.html
[3]http://www.mathworks.com/access/helpdesk/help/techdoc/math_anal/math_anal.shtml
[4]http://www.mathworks.com/access/helpdesk/help/techdoc/creating_plots/creating_plots.shtml
[5]http://www.mathworks.com/access/helpdesk/help/techdoc/visualize/visualize.shtml
[6]http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ref.shtml
[7]http://www.mathworks.com/access/helpdesk/help/techdoc/ref/refbookl.shtml
[8]http://www-h.eng.cam.ac.uk/help/tpl/programs/matlab.html

- Reading the online version of this document[9] saves you having to type examples in - you can just copy/paste. You won't have to type the WWW addresses either! A PDF version[10] is online if you need a paper copy.

## 2 Starting Matlab

On the teaching system you can type `matlab` at a Terminal window, or look in the `Programs/ Matlab` submenu of the `Start` menu at the bottom-left of the screen. Depending on the set-up, you might start with several windows (showing documentation, etc) or just a command line. This document will focus on command line operations.

## 3 Finding your way around matlab

### 3.1 Variables

- To list the variables currently accessible, use `who` (or for more details, `whos`). The variables will be arrays. To find the size of an array `A` use `size(A)`, which will return the number of rows and columns.
- To remove a variable `A` use `clear A`.

### 3.2 Functions and scripts

Matlab has many functions available

- If you know the name of a function but you don't know how to use it, try the `help` command - e.g. `help sum` tell you how to sum the elements of an array. Throughout this document details of commands will be omitted - using `help` or, better still, the WWW documents will help you find all there is to know.
- If you *don't* know the name of the function you need, look through the lists in the first section, or use the `lookfor` command - e.g. `lookfor sum` will tell you the names of all the functions that might have something to do with summing.

### 3.3 Directories and the `path`

When you type a command, matlab has to decide how to respond. First it checks to see if it has a *built in* function of that name. Then it checks through a list of directories looking for an appropriately named file.

- To see what's actually going to be run when you type a command, use `which` - e.g. `which sum` shows that the `sum` command is built into matlab.
- To see what directory matlab's in, type `pwd`. To change directories use the `cd` command.
- `what` lists the matlab-specific files in the current directory.

[9]http://www-h.eng.cam.ac.uk/help/tpl/programs/Matlab/matlabdatabook/
[10]http://www-h.eng.cam.ac.uk/help/tpl/programs/Matlab/matlabdatabook/matlabdatabook.pdf

- To see which directories are searched, use `path`. Most of the directories in the path belong to matlab. The final place it looks is the current directory. If you have a subdirectory called `matlab` in your home directory, matlab will look there before looking in any other directory. The ordering of these directories matters - if you have a file in the current directory that's the same name as a system file, the system's file will be run rather than yours.

## 4 Matrix operations

### 4.1 Creating Matrices

A 1x1 matrix can be created using something like `A=7;`. `A=7:10;` creates a matrix with 4 elements, namely 7, 8, 9 and 10. `A=7:0.5:10;` creates a matrix with element values running from 7 to 10 with a step-size of 0.5.

Between `[` and `]` commas (or spaces) are used to put things in rows, while semicolons (or line-breaks) put things in columns. So

```
b = [ 1 3 5; 2 4 6];
```

creates a matrix with 2 rows and 3 columns.

Matrices can contain strings - e.g. `A='bat'`.

There are commands to create particular types of matrices - `zeros`, `ones`, `eye`, `diag`, `rand`, etc.

### 4.2 Picking sub-matrices

Rows, columns and submatrices can be selected from a matrix. Here are some examples. Note how parentheses (rather than square brackets) are used when manipulating matrices (the % signs introduce comments)

```
b = [ 1 3 5; 2 4 6];
c=b(2,:) % row 2
d=b(:,1) % column 1
e=b(:,2:3) % columns 2 and 3
```

Matrices can be composed from submatrices, so `f=[d e]` would produce a matrix equal to the original `b`.

### 4.3 Matrix operators

Matrix operators work much as expected. In order of precedence they are listed in table 1.

| | |
|---|---|
| ′ | transpose |
| ^ | power |
| $*$ | multiplication |
| / | right division |
| \ | left division |
| $+$ | addition |
| $-$ | subtraction |

Table 1: Matrix operators

Note that the normal matrix rules apply, so matrices need to be appropriately shaped. Using `.*` (note the extra dot) multiples corresponding elements together.

`./` and `.\` work in a similar way. `.^` raises each element to the given power (rather than the matrix as a whole).

## 4.4 Matrix functions

Functions exist to perform common operations on matrices. A few are listed in table 2

| | |
|---|---|
| `inv` | inverse |
| `det` | determinant |
| `trace` | trace |
| `eig` | eigenvectors and eigenvalues |
| `rank` | estimate rank |
| `null` | calculate a basis for the null space |
| `rref` | Gaussian elimination on an augmented matrix |
| `lu` | LU decomposition |
| `qr` | QR decompostion |
| `svd` | SVD |
| `pinv` | Pseudoinverse |
| `rot90` | rotation of 90 degrees |
| `triu` | extract upper triangle |
| `tril` | extract lower triangle |

Table 2: Matrix functions

Some functions are especially useful when used with string matrices. E,g.

```
A='abcdefg'
A=fliplr(A)
A=sort(A)
circshift(A,[0 1])
```

# 5 Maths

As well as the usual operators there are many functions. See table 3

## 5.1 Complex Numbers

By default `i` and `j` have the value $\sqrt{-1}$ and complex numbers can be represented in the usual way - e.g `s = 2i+3`. The usual operators and functions can be used on complex numbers and there are some special functions too - `imag`, `real`, `conj`, `angle`, etc. Note that `abs` works with complex numbers, and that matrix transposition by default conjugates the elements (use `.'` rather than `'` to suppress this).

## 5.2 Numerical Analysis

It helps to know some numerical analysis terms.

**Inf :-** The IEEE representation of positive infinity.

**NaN :-** The IEEE representation of 'Not-a-Number', used when an operation has an undefined result (e.g. 0/0). All logical operations involving `NaN` return false, except for $\neq$. If you try to plot points with the value `NaN`, nothing appears. This can be useful for clipping, making holes in surfaces etc.

| | |
|---|---|
| `cos` | cos (angle needs to be in radians) |
| `sin` | sin (angle needs to be in radians) |
| `tan` | tan (angle needs to be in radians) |
| `exp` | exponential |
| `log` | natural log |
| `log10` | base 10 log |
| `sinh` | hyperbolic sin |
| `cosh` | hyperbolic cos |
| `tanh` | hyperbolic tan |
| `acos` | inverse cos |
| `acosh` | inverse hyperbolic cos |
| `asin` | inverse sin |
| `asinh` | inverse hyperbolic sin |
| `atan` | inverse tan |
| `atan2` | 2-argument form of inverse tan |
| `atanh` | inverse hyperbolic tan |
| `abs` | absolute value |
| `sign` | sign (either -1 or +1) |
| `round` | round to the nearest integer |
| `floor` | round down to the nearest integer |
| `ceil` | round up to the nearest integer |
| `fix` | round towards 0 |
| `rem` | remainder after integer division |
| `sqrt` | square root |

Table 3: Math functions

**Condition Number :-** The condition number of a matrix (calculated using `cond`) gives a fair indication of how well the matrices can be operated upon.

## 5.3 Sparse Matrices

If you know that your matrices will stay nearly empty, then make them sparse - they'll take up less space and can be operated on more quickly.

```
% comparison of sparse vs full matrices
% Create sparse and full versions of the same matrix.
% Note that the density is 0.05. The break-even point
% for this operation seems to be about 0.25.
S = sprandn(60,60,0.05);
F = full(S);
% Compare speed.
disp('Sparse')
t0=cputime;
B = S * S;
cputime-t0
disp('Full')
t0=cputime;
C = F * F;
cputime-t0
```

### 5.4 Algebra

Matlab stores polynomials as vectors. For example, $2x^2 - 3x + 1$ can be represented as `p=[2 -3 1]` and roots found by using `roots(p)`. `poly` and `polyval` are also useful.

### 5.5 Symbolic Maths

Toolboxes are adds-ons. One of the most useful and underused (the Symbolic Math toolbox) performs factorisation, integration, etc, so `int('1/(1+x^2)')` works, as does

```
dsolve('D2y = -a^2*y', 'y(0) = 1', 'Dy(pi/a) = 0')
```

which solves an ODE. The toolbox also makes function plotting easier - `ezplot('sin(3*t)',[-10 10])` becomes possible.

## 6 Screen textual output

- The `format` command controls how numbers are output. For example `format long g` will display 15 digits in either exponential or fixed format, which ever it thinks best.
- Finishing a command or variable initialisation with a semi-colon suppresses the output,
- You can display the contents of an array just by typing its name (without a semi-colon) but that will display the array name too. Particularly if the array's a string the `disp` command is useful - it suppresses the printing of the array name.

## 7 I/O

### 7.1 Saving and loading data

By default matlab saves values into *MAT-files* with a `.mat` extension, so `save results Z` would save the `Z` array into a file called `results.mat`. This array could be reloaded into matlab by using `load results`.

You can load files produced by other programs. If a file called *data* contains "1 2 3" then `load data` will create a matlab matrix called `data` which contains those numbers.

### 7.2 Saving graphics

If you want to save graphics in a particular format (postscript, jpeg, tiff, PNG, etc) use the `print` command. For later inclusion into a LaTeX document, `print -depsc` (colour encapsulated postscript) might be best. For Word documents, try `print -depsc -tiff`, which adds a TIFF preview.

### 7.3 Printing

Using `print` or the `print` menu option will print the current figure to the default printer. On the teaching system the `cuedpr` command will do this in a paper-efficient way.

## 8 Graphics

### 8.1 General

Before you display graphics you have to calculate coordinates - unlike some other programs you can't just print `sin(x)` and give a range. Typically you'd do something like the following

```
x=linspace(0, pi, 100);
y=sin(x);
plot(x,y);
```

The `linspace` command creates a vector of 100 elements equally spaced between 0 and pi inclusive.

Once you have a graph it's easy to add titles - use the menus on the graphics window or type commands like

```
title('Trig');
xlabel('Radians');
ylabel('Sin');
```

You can plot more than one function on a graph. If you use `hold on` before doing another plot, the first plot won't be erased. Alternatively, you can give the `plot` command 2 lines at once. The advantage of this approach is that the lines are automatically drawn in different colours and that legends are easily created - e.g.

```
x=linspace(0, pi, 100);
y=sin(x);
y2=cos(x);
plot(x,y,x,y2);
legend('Sin','Cos');
```

Adding a grid is easy - use `grid on`. The menu options let you change many features of graphs, but you can also use the `axis` command to change many axis features - range, aspect ratio, etc.

The `plot` command has many options to control colour, markers, etc. There's also `scatter`, `bar`, `pie`, `stairs`, `stem`, `polar`, `compass`, etc. Type `help graph2d` or (if you want to see some graphics!) see the Graphics tutorial[11]

Note that you can have many figure windows open (read about the `figure` command). You can also have many graphs in each figure. The `subplot` command creates axes in tiled positions. So for example if you want 3 rows each containing 2 graphs, and you want to plot vector `x` in the bottom left graph (the 4th graph, counting left-to-right), you type

```
subplot(3,2,4);
plot(x);
```

Use `clf` to clear the current figure.

### 8.2 3D plots

There are many 3D options too, but again you still need to calculate all the coordinates.

[11]http://www.mathworks.com/access/helpdesk/help/techdoc/creating_plots/intro_ne.html

```
 x=linspace(-1, 1, 10);
 y=linspace(-1, 1, 10);
 [X, Y]=meshgrid(x,y);
 Z= X.^2-Y.^2;
 mesh(Z);
```

The `meshgrid` command creates 2 2D matrices, each with as many elements as there are grid-points. The `X` matrix has the x coordinates and the `Y` matrix the y coordinates. Then the `z` coordinate is calculated for each point and put in a `Z` matrix. Instead of `mesh`, `surf` could be used to display a facetted surface. Then the following can be done to add graduated colour and a key.

```
shading interp;
colorbar
```

## 8.3 Plotting experimental data

Most of matlab's 3D facilities expect the data to be evenly spaced on the XY plane, but what if you have some scattered experimental data? First you need to interpolate the data. Suppose you have your datapoints in 3 arrays: `x`, `y`, and `z`. Then the following will extrapolate the data onto a coarse 10 by 10 grid and display a colour-coded surface.

```
 xi=linspace(min(x), max(x), 10);
 yi=linspace(min(y), max(y), 10);
 [X, Y]=meshgrid(xi,yi);
 Z=griddate(x,y,z,XI,YI)
 surf(Z);
 shading interp;
 colorbar
```

There's also `contour`, `scatter3`, `waterfall`, `slice` and `ezplot`. The `colormap` command controls palettes. Type `help graph3d` for details.

## 8.4 Movies

To create an animated sequence you first need to draw and save each frame. This example creates a 16-frame movie then runs it 5 times.

```
% movie demo
M=moviein(16);
for j = 1:16
   plot(fft(eye(j+16)))
   M(:,j)= getframe;
end
movie(M,5)
```

# 9 Graphics and Objects

Though it's not essential to know how matlab graphics are organised, it's likely to be useful sooner or later.

Figures are made of objects (lines, text, images etc). Objects have properties (like Color, Position, etc) which can be individually controlled. Each object has a *handle* (a unique identifier). To experiment with handles, create a simple graph by typing '`plot(1:3)`'. Typing '`gcf`' returns the handle of the current figure. '`get(`*`handle`*`)`' lists the properties of the given object, so `get(gcf)` lists all the properties of the current figure. You can use `set` to change the values of these properties once you know the handle of the object and name of the property. For example, `set(gcf,'Color','green')` sets the color of the current figure to green. If you type `set(gcf)` you get a list of the properties of the current figure, and alternative settings for these properties.

The axes are children of the figure - and consequently inherit some of the figure's properties. If you type `get(gcf,'Children')` you'll get the axes handle, which you could also get by typing '`gca`'. The axes in turn can have children.

'`set`' is a powerful mechanism. Often matlab has commands that spare you needing to use `set` directly, but particularly if you want to retrospectively fix something, the facility is useful.

## 10 Programming Constructs

Matlab is a language, not just a calculator. It has the usual programming constructs. Both `if` and `while` understand all the usual boolean operations (though note that `~` rather than `!` is used for negation - see table 4).

| | |
|---|---|
| == | equal |
| ~= | not equal |
| > | greater than |
| >= | greater than or equal |
| < | less than |
| <= | less than or equal |
| & | logical AND |
| \| | logical OR |
| ~ | logical NOT |

Table 4: Matrix functions

### 10.1 if

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

### 10.2 while

```
k=1;
while t(k) == 0,
  k=k+1;
end
disp('First nonzero entry in t is', disp(t(k))
```

### 10.3 for

This command is often not needed because matlab functions (`sin` for example) often work on whole arrays. The loop below adds an element to `back` on each iteration.

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

### 10.4 switch

This is like a multiple `if` statement.

```
  sport='tennis';
  switch sport
  case 'soccer'
    disp('you like soccer')
  case 'tennis'
    disp('you like tennis')
  otherwise
    disp('maybe you like some other sport')
  end
```

## 11 Writing your own scripts and functions

It soon gets tedious to type everything on the command line. If you put any valid sequence of Matlab statements into a file (using an editor) 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). For example, if you type `edit dothis`, then `matlab` will start an editor ready to edit a file called *dothis.m*. If you save the following text into it

```
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`).

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* 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.

It is also possible to pick up subroutines which have been written in other languages and pre-compiled. See the Interface Guide[12] for details.

### 11.1 Global variables

Ordinarily, each function has its own local variables, which are separate from those of other functions, and from those of the base workspace. However, if several functions, and possibly the base workspace, all declare a particular name as `global`, then they all share a single copy of that variable. Any assignment to that variable, in any function, is available to all the other functions then declare it as `global`.

## 12 Debugging

- Use `size`, `whos` and `which` to check on variables (or selectively remove semicolons from commands so that output isn't supressed).
- Be careful when choosing variable names
  - `i` begins with the value `sqrt(-1)` but typing `i=7` will change its value.
  - If you give a variable the same name as a function odd things can happen. E.g. the following causes an error on the last line

    ```
    A=3
    size(A)
    size = 1
    size(A)
    ```
- The `keyboard` command stops function/script execution so you can examine and change values. To continue, type `RETURN`.
- Sometimes the error in your program triggers an error message from a function that your program directly or indirectly calls. Use `depfun` to see the dependent functions.
- On some systems a Debugger is available as an item on the menu bar.

## 13 Miscellaneous

### 13.1 Stopping

Use `CTRL-C` to interrupt a command. Use `quit` to exit from matlab.

### 13.2 Program-generated names and commands

This shows how to generate variable names 'on the fly'. This is especially useful when you're running batch jobs and want to save results into several files.

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

```
% A1, A2, A3, and A4 are created and set to 3
for i=1:4
s=sprintf('A%d= 3',i);
eval(s);
end;

% This shows how to run a command that's stored in a string
str=input('Type in a command ','s')
eval(str)
```

## 13.3 Vectorising

`for` loops are slow and can often be eliminated, though the resulting code may be less readable. For example, to replace all values $< -1$ by $-1$ and all values $> 1$ by 1 in a matrix `A` you can just do `A = min(max(M, -1),1)`. See the Matlab vectorisation tricks[13] page for details.

## 13.4 Calling Fortran or C

To find out how to run Fortran or C code from matlab see the External Interfaces[14] page which includes tutorials and examples.

[13]http://www-h.eng.cam.ac.uk/help/tpl/programs/Matlab/tricks.html
[14]http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/matlab_external.shtml