Numerical Analysis.
Beginning MATLAB commands.

`MATLAB` stands for matrix laboratory and has been designed specifically for matrix computations. You need only a small number of commands to start using `MATLAB` effectively. This tutorial introduces you to basic `MATLAB` commands.

Good general purpose commands:

`help` This command is extremely useful, particularly at the beginning.

`lookfor` Want to find a command? Use this. For example, *lookfor plot* returns all commands that contain the word ‘plot’ in their one line summary.

`demo` This is a good command to get an idea of what MATLAB can do.

↑ gives you the last command. You can use this repeatedly.

% a comment line.

; a semicolon suppresses output to the command window.

`clf` clears the current figure .

`clear` or `clear all` clears the workspace variables.

Basic programming commands:

MATLAB has all the basic programming commands. The easiest way to access a list of these commands and their syntax is to use the `help` command. For example, *help lang* will give you a list of programming constructs and you can then type *help if* or *help break* to learn how to use those commands.

Similarly, MATLAB has the usual operators.

`=`	assignment operator
`==`	Boolean equals
`&`	Boolean and
`|`	Boolean or
`~`	Boolean not

Again, if you need more information, just use the help command.

Basic plotting commands:

MATLAB has good graphics and there are many ways to plot functions in MATLAB. By far and away the easiest is:

`ezplot` For example, *ezplot(‘x^2’,[-3,3])* plots $y = x^2$ for x values from -3 to 3.

More generally, MATLAB plots vectors of x-values versus vectors of y-values. For example,

`plot([1 2 3 4], [-1 2 0 20], ‘ro’)` plots the points $(1, -1)$, $(2, 2)$, $(3, 0)$ and $(4, 20)$ in red dots.

`plot([1:4], [-1 2 0 20], ‘ro’)` plots the same points. Here the colon means 1 through 4.

`hold on` holds the previous plots on the figure.

`hold off` a new plot will replace the current figure.

Inputting and accessing arrays:

`u = [2 4 5]` creates the 1×3 array $\begin{pmatrix} 2 & 4 & 5 \end{pmatrix}$

`u = [2; 4; 5]` creates the 3×1 array $\begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$. The semi-colon indicates the end of the row.

`u = [2 4 5]’` creates the 3×1 same array as above. The apostrophe is called the *transpose* operator.

`u = [3:6]` creates the array $\begin{pmatrix} 3 & 4 & 5 & 6 \end{pmatrix}$. This is useful in loops.

`u = [3:2:7]` creates the row vector $\begin{pmatrix} 3 & 5 & 7 \end{pmatrix}$. The syntax is u = [beginning number : step size : end number]. In the example above, the step size is two.

`A = [1 2 3; 4 5 6]` creates the 2×3 array $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$. The semi-colon separates the rows.

`A(2,3)` accesses the entry a_{23} of the array B above. For example, A(2,3) = 6.

`A(:,2)` returns *all* the rows of A in column 2. A(:,2) = $\begin{pmatrix} 2 \\ 5 \end{pmatrix}$. The colon ':' means *all* rows and the '2' means the second column.