ESSENTIAL MATLAB*

This summarises the MATLAB needed for completing the computational exercises associated with the CHEN30101 course.

1. Basics. Start MATLAB, you should see the prompt `>>`. This is where you type commands—it is called the **command window**.

- To check the version of MATLAB and list the toolboxes that are available, type `ver`.
- To find out what your current directory is, type `pwd`.
- To change the current directory, for example to access an external USB drive, type `cd g:` (assuming your USB is associated with drive `g:`). You can only access functions (also called M-files) which are in the current directory.
- To save an M-file from the web, open the **editor window** by typing `edit`. Then use the mouse to highlight the contents on the web page and copy into the **editor window**. Finally click on the save button to store the function as an M-file in the current directory (your USB stick, for instance).
- To check that you have successfully saved an M-file in the current directory, type `what`.
- To get help with the syntax of the built-in commands type `help <command>` where `<command>` is any built in MATLAB function. For example, `help ones` gives the syntax of the `ones` command that is described below.

You are now ready to start computing!

2. Scalars. To assign values to a real variable you use the `=` command.

- To set variable `a` to the number 2, and variable `b` to the number π, you simply type `a=2,b=pi,` If you use a semicolon to terminate the command instead of the comma, that is, type `a=2;b=pi;` then the assignments are not echoed to the screen. In this case, typing `a` or `b` will echo the current value of the variable to the screen.
- MATLAB internally holds numbers to an accuracy of 15 decimal places—this is called 64-bit arithmetic. To change the format of the output to the "full" precision type `format long e`; to change back type `format short e`.
- To list the current variables, and find out how much computer memory is needed to hold them, type `whos`.
- Variables can be combined using addition `+`, subtraction `-` , multiplication `*`, right division `/`, left division `\` and exponentiation `^`. For example, `a*b` gives 2π, `a/b` gives $2/\pi$, `a\b` gives $\pi/2$ and `b^a` gives π^2.

Note that MATLAB will compute results in complex arithmetic if required!

3. Vectors. To assign values to a real vector you typically need to use square brackets and a single argument delimited by commas (for a row vector) or semicolons (for a column vector) together with the `=` command.

- Type `a=[1,2,4,8],` to create the row vector `a` equal to $[1, 2, 4, 8]$, and then type `b=[1;2;4;8],` to create the corresponding column vector `b`. If you use a semicolon to terminate the command then the assignments are not echoed to the screen.
- To transpose a vector you can use the `'` command. For example `c = a'` gives the same vector as `b` above.

- A component of a vector is accessed using round brackets. For example `d=a(4)` sets the variable `d` to 8 (the fourth component of the vector `a`).
- Vectors of the same length and orientation can be combined using addition `+` , subtraction `-` , componentwise multiplication `.*` , right division `./` , left division `.\` and exponentiation `.^` . For example, `b-c` gives the 4×1 zero vector, and `b./c` gives the 4×1 vector of ones. The three componentwise operations `./` `.\` and `.^` are extremely useful in practical computation.
- Typing `zeros(k,1)` or `ones(k,1)` creates a column vector of zeros or ones, with length equal to `k` whenever `k` is a positive integer.
- To list the current variables and vectors, and find out how much computer memory is needed to hold them, type `whos`.

4. Colon notation. You can use the colon `:` to generate vectors. You can also use it to define a loop that is to be repeated a fixed number of times.

- To create a vector of integers from `1` to `k` type `1:k`
- The construct `for m=1:k, <commands>, end` defines a "loop" in MATLAB. The sequence of instructions between `for` and `end` is executed exactly `k` times. The first time has `m=1`, the second time has `m=2`, and the last time has `m=k`.
- Type `t0:dt:t1` to create a vector of numbers from `t0` to `t1` with "increment" `dt`. Typing `x=0:1/4:5,` generates the vector $[0, 1/4, 2/4, \ldots, 19/4, 5]$.

5. Matrices. To assign values to a matrix you follow the vector notation and use square brackets, with commas to delimit rows and semicolons to delimit columns.

- Type `A=[11,12,13;21,22,23;31,32,33]` to create a 3×3 matrix with entries `ij` in the `i`th row and the `j`th column. Type `I=[1,0,0;0,1,0;0,0,1]` to get the 3×3 identity matrix.
- To transpose a matrix you can use the `'` command. For example `B = A'`.
- Just like vectors, you can access any of the matrix entries using round brackets. For example `d=A(1,3),` sets the variable `d` to 13.
- You can extract a row or a column of a matrix using a colon. For example `y=A(2,:),` generates a row vector `y` which is the same as the second row of `A` and `z=I(:,3),` generates a column vector `z` which is the same as the third column of `I`.
- Matrices of the same dimension can be combined using addition `+` , subtraction `-` , componentwise multiplication `.*` , right division `./` , left division `.\` and exponentiation `.^` . For example, `M=A-2*I` generates a new matrix `M` by subtracting twice the identity matrix from the matrix `A`, and `N=A./I,` divides the entries of A by the corresponding entries of the identity. (Note that MATLAB can deal with division by zero[1] without breaking down!
- Matrices and vectors with consistent dimensions can be multiplied together using `*`. For example `y=A*z` is the column vector which results when you multiply `A` by the third column of the identity matrix. (This means that `y` should be equal to the third column of `A`.)
- You can solve linear systems of equations using the "backslash" `\` command. For example `x = M\z` is the solution to the linear equation system `Mx = z`. Typing `W = M\I` sets `W` to the inverse of the matrix `A`. (You can check by computing `W*M` and comparing with the identity matrix.) How cool is that?

The fact that matrices can be treated as "objects" in terms of basic arithmetic is one of the reasons why MATLAB is so incredibly useful.

[1] This is because it always uses so-called IEEE arithmetic.