

# Introduction to MATLAB

Matlab is a program that allows you to carry out computations in a straightforward manner, removing much of the tedium involved in programming. It is extremely useful for creating simulations of neural networks, as well as for general types of data analysis and visualization.

## Getting started

You start Matlab by simply clicking on the icon in the desktop. This will open a command window, in which you can start typing commands and doing computations. Within the command window there will appear a prompt, which is where it is waiting for you to type a command. The Matlab prompt looks like this:

```
>>
```

This document is intended as a gentle introduction to Matlab. Once you finish it, you should move on to the Matlab introduction provided in the helpdesk (type 'helpdesk' at the prompt).

## Manipulating vectors and matrices

Matlab is built around vectors and matrices. A vector is just a list of numbers, whereas a matrix is a 'box' of numbers. Vectors and matrices have geometrical interpretations, but for now we will just focus on algebraic manipulations of these quantities.

To create a vector of the numbers 4, 6, 9, 2, type (after the prompt):

```
x = [ 4 6 9 2 ]
```

After hitting return you will see Matlab return the result:

```
x =  
4      6      9      2
```

You will also get the same answer now if you simply type 'x' at the prompt.

Creating a matrix is quite similar. You just have to separate the rows with a semicolon. For example, typing

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

will return:

```
A =  
    16     3     2    13  
     5    10    11     8  
     9     6     7    12  
     4    15    14     1
```

Now let's say you're interested in the **SUM** of each column. Just type

```
sum(A)
```

and you get:

```
ans =  
    34    34    34    34
```

(yes, they are all the same, if you've entered "A" correctly).

NOTICE: The `sum( )` command worked "column-wise". Most matlab functions work this way. What this means is that the 'default' is to give an answer working through each column. It's just a matter of consistency – Matlab could have chosen to consistently give answers working "row-wise".

How to find this out?

**All Matlab functions have command line help. Just type "help" followed by the name of the function.**

For example,

```
help sum
```

Now let's assign a new variable, "S" to represent the sum of each column:

```
S=sum(A);
```

Notice the semicolon at the end—you do this if you want to suppress output from a command. I already know the answer (from above) and I don't want to see it again. If you want to double check, just type `S` at the prompt:

```
>> S
S =
    34    34    34    34
```

Now that you have several variables in the workspace, you may want to keep track of them all. A very useful command for doing this is “whos”.

```
>> whos
Name          Size          Bytes   Class
A             4x4            128    double array
S             1x4             32    double array
ans           1x1              8    double array
x             1x4             32    double array

Grand total is 25 elements using 200 bytes
```

This helps you keep track of the variables and is useful for de-bugging programs. The most helpful are “Name” and we see our variables “A” and “S” and “Size”. “A” is a 4 x 4 matrix and S is a 1 x 4 matrix (i.e., a **row vector**).

To find out the size of a single variable, use the size command:

```
>> size(A)
ans =
     4     4
```

[By the way, let's say that you notice that you typed in the matrix “A” incorrectly. Do you have to re-type everything... not really. The up-arrow and down-arrow allows you to scroll through previously typed commands.]

If you want to access the value of a single element of an vector or matrix, use **subscripts**. For example, `x(3)` refers to the third element of `x` (which is 9), and `A(2,4)` refers to the element in the 2<sup>nd</sup> row and 4<sup>th</sup> column of A. In general `A(i,j)` gives you the i<sup>th</sup> row and the j<sup>th</sup> column of A.

For example, to sum the elements of the first row of A manually, type:

```
A(1,1)+A(1,2)+A(1,3)+A(1,4)
```

which produces

```
ans =  
34
```

This demonstrates matrix subscripting, but obviously it is VERY inefficient for accessing large numbers of elements. A more efficient way is to use the colon notation. When you type a ':' where a subscript should be, it means "give me all elements of this row or column." For example

```
A(1,:)
```

returns

```
ans =  
16      3      2     13
```

Now we can do our sum easily by typing:

```
>> sum(A(1,:))
```

The parentheses should not be confusing. The values entered into a Matlab function are contained in parentheses. If you want the sum of a variable, you type `sum(variable)`. The "variable" that we want is expressed by `A(1,:)` so we enter this into `sum( )`, hence the double parentheses.

It is sometimes useful to create new variables from PARTS of other variables. Say we want a new variable "B" to equal the last row of our matrix "A". Knowing how to subscript, it should be easy:

Try:

```
B = A(4,:);
```

Do you remember all the variables we have? If not, remind yourself with a "whos" command.

## MORE FUN WITH COLONS

Colons can also mean "TO"

Try:

```
1 : 10
```

This means, obviously, all numbers from 1 to 10 inclusive. Getting even fancier, try:

```
0 : 5 : 50
```

The middle number is the “step size”. If you leave it out, a step size of 1 is assumed.

Now let’s say we just want the first three columns from the third row of A. So we type

```
A ( 3 , 1 : 3 )
```

Make sense? It says, “give me row 3, columns 1 to 3 of matrix A”.

## Mathematical operators

I won’t go into much detail. Some of the simple operators are

+	add
-	sub
*	mult
/	div
^	power
sqrt	square root

and it has lots of built in numbers-e.g., *pi*.

We’re done with all the variables that we’ve used. Typing “clear” will clear the matlab workspace. To prove it, type “whos” and you will see that everthing is gone. You can clear individual variables with

```
clear <name_of_variable>
```

To save the workspace before quitting, type

```
save <filename>
```

## Plotting

Let's say we want to plot one cycle of sinewave. First, define the numbers on the x-axis:

```
x = 0:pi/10:2*pi;
```

This looks a little confusing, but what does it mean? We are setting  $x$  equal to a list of numbers starting at 0 and going to  $2\pi$  in steps of  $\pi/10$ .

Now let's compute the sine function on each of these numbers:

```
y = sin(x);
```

The variable  $y$  is now a vector containing the result of the sine function applied to each of the numbers within  $x$ . To plot it out, we just type

```
plot (x,y)
```

This plot shows a smooth sine wave, but is slightly misleading. It's not really smooth but made of discrete points (remember 'step size'). To see this, type

```
plot(x,y, '*')
```

This command tells Matlab to plot each point with an asteriks, so you can see each of the data points making up the plot. If you want to see both the data points and the lines, just type

```
plot(x,y, '*-')
```

Normally, whenever you use a plotting function in Matlab, it will erase the current figure and replace it with the new one. If you'd like two graphs to be plotted with the same axis, use the "hold on" function ("hold off" to turn it off). Now, any subsequent plots will be drawn on this same axis. If you want a new figure window (for a new graph), type

```
figure
```

An easy way to get rid of figures: `close (figure#)`, to remove one figure or "close all" to remove all figures.

## Loading a File

It's easy to load data files in Matlab. For example, download the file 'astronaut.mat' from the class web page and bring it into your working directory. (The default working directory is the

work/ directory in the Matlab folder. To change the working directory, type 'cd *new\_directory*'.) Now load the file into the Matlab workspace by typing

```
load astronaut.mat
```

(“.mat” files are a Matlab-specific data file. To find out more see “help load” and “help save”).

So what have you just loaded in? Type `whos` to find out. You’ll see that it’s a variable named ‘astronaut’ which contains a 256x256 array of numbers

When I have a lot of data and I want to get a feel for what the numbers are like, I’ll plot a portion of it. For example,

```
plot(astronaut(128, :))
```

plots the data in row 128 of this array. You see the values for this row range between 0 and about 240.

This matrix is actually an “image”. To see what it looks like, type

```
imagesc(astronaut)
```

“imagesc” is one of Matlab’s image display commands that automatically “scales” (sc) the data between 0 and 1.

What are you looking at? Yes, it’s an astronaut, but you’re actually looking down on top of the matrix. The value at each position in the matrix is assigned a different color value. The way each value is assigned a color is determined by the “colormap” that’s being used (the mapping from number value to a specific color).

Matlab has a number of different colormaps. For this image the most appropriate is a simple grayscale colormap, which you obtain by typing:

```
colormap gray
```

The image should look more normal. Large values are assigned white, small values are assigned black.

Finally, we need to get the aspect ratio right. By default, Matlab stretches the image to fill the figure window, which by default is rectangular. But this image is square, so to display it properly, preserving the aspect ratio, type

```
imagesc(astronaut), axis image
```

Let's say that we're interested in looking at a smaller portion of the astronaut. For some reason we decide that only a portion of the image is relevant. We already know how to subscript a matrix (specific rows and columns). Let's say we only want to examine the image between coordinates 50 and 200 on the x-axis and 1 and 200 on the y-axis. We could do either

```
new_astronaut = astronaut(50:200,1:200);  
imagesc(new_astronaut)
```

or we could just go for it directly without creating a new variable:

```
imagesc(astronaut(50:200,1:200))
```

## Writing M-file Scripts

When you find yourself executing a series of commands over and over again, it is more efficient to write an m-file script which perform a series of Matlab commands by calling a single function call.

An m-file is a text file saved with a .m extension. Matlab executes each command in order that they appear in the text file, as if you were typing them into the command line. You execute the commands in the M-file by simply typing the name of the file (without the .m suffix) at the Matlab prompt.

On Unix systems: just use your favorite text editor. On PCs: Go to File → New → M-file in the command window (matlab has its own built-in text file editor).

For example, here's an m-file script that loads the astronaut file, "downsamples" it, and resizes it:

```
% my_astro.m - loads and displays astronaut  
  
load astronaut  
subplot(2,2,1)  
imagesc(astronaut);  
axis image off  
colormap gray  
title('Original image')  
  
% reduce resolution by a factor of 2  
astronaut2=astronaut(1:2:256,1:2:256);
```

```
subplot(2,2,2)
imagesc(astronaut2); axis off
colormap gray
title('Downsampled by factor of 2')

astronaut3=astronaut(1:4:256,1:4:256);
subplot(2,2,3)
imagesc(astronaut3);axis off
colormap gray
title('Downsampled by factor of 4')

astronaut4=astronaut(1:8:256,1:8:256);
subplot(2,2,4)
imagesc(astronaut4);axis off
colormap gray
title('Downsampled by factor of 8')
```

The lines beginning with % are comments – they are ignored and are just used to help remind you what your program does.

Save the script as the file `my_astro.m` in your working directory, and then execute it by typing `my_astro` at the prompt. Did it work?