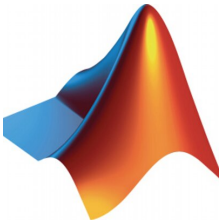
INM460 / IN3060 Computer Vision



This lab is an introduction to Matlab. Matlab is the programming environment we'll be using this term in this module. Matlab is installed on the university computers, and includes a variety of toolboxes that you will find useful for your work, including the Image Processing Toolbox, Computer Vision Toolbox, and Signal Processing Toolbox.

The university has an academic site license for Matlab. This license allows you to download and install Matlab on your personal machines. This is a great perk for being a City University London student, as buying Matlab and the toolboxes included in our subscription would be quite expensive if you did this as an individual.

It's recommended you install Matlab so that you can work at home or on your laptop, in particular the 2018b version or a newer one. If you haven't done so already:

- Go to https://www.city.ac.uk/current-students/it-support and click on "Resources and Facilities", log in, and click on "SPSS, Matlab, NVivo, OxMetrics" and then click on Matlab.

Here, you can read instructions on how to get an account on the Mathworks (the makers of Matlab) website and download the software. Note: this will require putting in a Service Request with City's IT department.

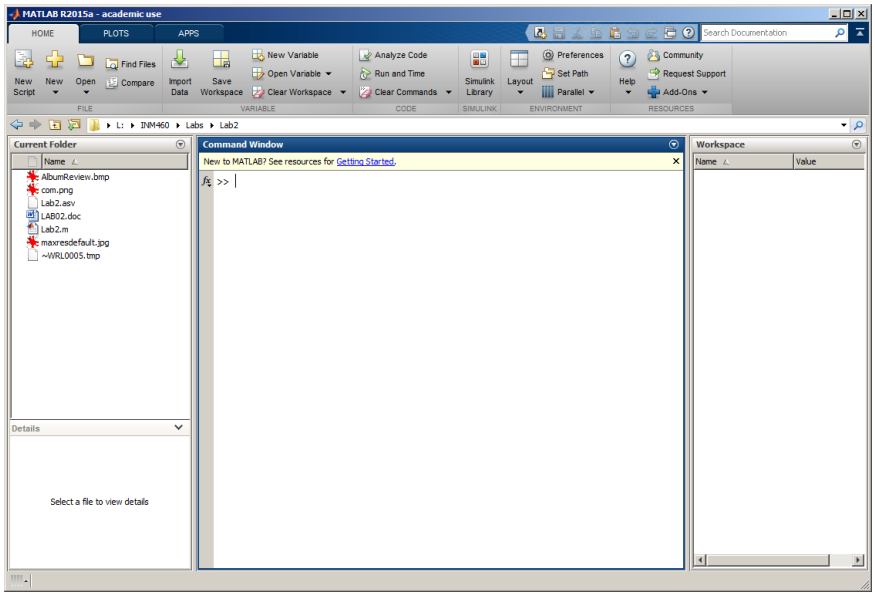
In addition, City University London used to have Matlab club where you can find more information on the basics of Matlab:
http://www.staff.city.ac.uk/~sbbk034/introduction.php?idMem=matlabC

Task 1: Hello Matlab

Matlab is a high level programming language initially released in 1984. Originally it was designed as an easy way to perform calculations on matrices for scientific computing. Over the years its influence has expanded, and it is now a popular language in many disciplines including image processing and computer vision.

Matlab consists of the core program, along with a series of *toolboxes* that provide additional functionality. The university academic license provides many toolboxes, and the key ones we'll need for this module. However, there are additional toolboxes that are not included.

Start Matlab. It is installed on the university computers under the programs staring with "M". When the program launches, you'll see the main user interface:



Three panes

The left pane is the Current Folder pane. This provides a listing of files that are in the current folder Matlab is running in. You can use the interface to change folders if you wish.

The middle pane is the Command Window. In this window you can enter commands, which typically are lines of code.

The right pane is the Workspace, which provides a listing of all variables currently loaded in to memory. The Workspace will list the variable's name as well as its value when possible.

Basic variables and datatypes

In the Command Window, you'll see a ">>" symbol. This is the command line where you can enter code. For example, type

```
x = 1
```

and press Enter. You'll see in the Workspace, a new variable called "x" has been allocated, and given the value 1. In the Command Window Matlab also shows you the value of x after it was assigned. If you wish, you can suppress this behaviour by using a semi-colon, i.e. entering

```
x = 1;
```

Matlab is a *typed* language. By default, variables in Matlab are assigned a double-precision floating point type. Conversions to other types are possible, such as single precision floating point, integer, unsigned integer, etc. For example,

```
y1 = single(x);
y2 = int32(x);
y3 = uint8(x);
```

will convert x to single, 32 bit integer, and 8 bit unsigned integer, and store in y1, y2, and y3, respectively. Whilst such representations can be more memory efficient, please bear in mind that *many functions in Matlab are limited to certain data types*.

You can use Matlab as a calculator. Basic arithmetic operators are + - * / ^ for addition, subtraction, multiplication, division, and exponentiation, respectively. For example, a command

```
x = 5+4/2
```

will assign 7 to x. One can use parentheses () to control precedence, for example

```
x = (5+4)^2
```

will result in 81, whereas

```
x = 5 + 4^2
```

will result in 21. You can also use variables; for example,

```
t  = 4
x = 5 + t^2
```

Strings can be composed in Matlab using single quotes. For example,

```
x = 'Hello Matlab'
```

will assign the string “Hello Matlab” to the variable x.

Exercise 1

Evaluate the function $x(t) = t^2 - 100t + 2500$ for t = 10.

Correct answer: 1600

Functions

One of the best features of Matlab is that it has a large set of built-in functions that are useful for scientific programming. Simple examples include trigonometric functions like sin, cos, tan, for sine, cosine, and tangent, respectively, as well as other elementary functions like sqrt, exp, log for square root, exponentiation (base e), and natural logarithm, respectively. At the command prompt, you can type

```
help elfun
```

and a list of built-in elementary functions supported by Matlab will be listed. For any particular function, you can type help followed by the function name to learn more about the function. For example,

```
help sin
```

will display text in the Console Window describing what the sin function is. If you'd like more detail, click on the blue hyperlink doc_sin (or "Reference page for sin") to open up the Help browser page for this function.

Exercise 2

Using Matlab, see if you can write the code to evaluate the function

$$f(x,y) = \frac{x^2 + 2xy - \cos(xy)}{\sqrt{x^2 + y^2}}$$

for x = 1 and y = 2, assigning the answer to a variable called f. The correct answer is 2.4222.

The function disp can be used to display a string. For example,

```
x = 'Hello Matlab';
disp(x);
```

will display the string “Hello Matlab” in the Command Window. This is the “Hello Matlab” program, which you’ll recognise as exceedingly simple.

Task 2: Vectors

A vector is simply an array of numbers. Vectors come in two forms: *row* vectors and *column* vectors. A row vector is an array of values arranged horizontally, whereas a column vector is an array of values arranged vertically. One can allocate a vector using square bracket notation, i.e. [].

Row vectors

When forming a row vector, you can provide a set of numbers, separated either by spaces or commas. For example, if you type any of

```
x = [1, 2, 3]
x = [1 2 3]
x = [1, 2 3]
```

you will produce a row vector with elements 1, 2, and 3. Matlab will display the vector in the Command Window as

```
x =
    1     2     3
```

Column vectors

A column vector is formed using semi-colons between numbers. For example,

```
y = [4; 5; 6]
```

will form a column vector with elements 4, 5, and 6. Matlab will display the vector in the Command Window as

```
y =
    4
    5
    6
```

You’ll recognise the numbers are arranged vertically. You can access elements of a vector using an index:

```
y(2)

ans =

     5
```

This returns the second element in the vector y, which is 5.

Vector arithmetic

You can do arithmetic on vectors of the same length, including addition, subtraction, element-wise multiplication and division, dot product, cross-product, norm, and more. For example, consider the following two column vectors:

```
x = [1; 2; 3]
y = [4; 5; 6]
```

The table below shows different arithmetic operations performed:

% Addition >>z = x+y z = 5 7 9	% Subtraction >>z = x-y z = -3 -3 -3	% Element-wise mult. >>z = x.*y z = 4 10 18
% Element-wise div. >>z = x./y z = 0.2500 0.4000 0.5000	% Element-wise power >>z = x.^2 z = 1 4 9	% Dot product >>z = dot(x,y) z = 32
% Cross-product >>z = cross(x,y) z = -3 6 -3	% Norm (length) >>z = norm(x) z = 3.7417	% Scalar mult. >>z = 3*x z = 3 6 9

One thing to be aware of is that if you try to do an illegal operation on vectors of different lengths, you will receive an error in the Command Window:

```
x = [1; 2; 3]
y = [4; 5]
z = x + y
```

Error using +
Matrix dimensions must agree.

Colon notation

Using colon notation, you can easily make a row vector. For example,

```
x = 1:4
```

will produce a row vector starting at 1 and ending at 4, i.e.

```
x =
    1     2     3     4
```

More generally, the syntax is a: b: c where a is the starting value, b is an increment, and c is the final value. Matlab will not let the vector extend beyond c. For example,

```
x = 0: .1: 0.5
```

produces the vector

```
x =
         0    0.1000    0.2000    0.3000    0.4000    0.5000
```

Using colon notation, you can specify a set of indices to extract a section of a vector. For example, you can extract elements 3 to 5 of the vector x as

```
y = x(3:5)

y =
    0.2000    0.3000    0.4000
```

Exercise 3

Make a row vector s that samples the function sin(t) for t=0 to 3.14 in increments of .2.

The correct answer is

s =

```
         0    0.1987    0.3894    0.5646    0.7174    0.8415    0.9320    0.9854    0.9996
0.9738    0.9093    0.8085    0.6755    0.5155    0.3350    0.1411
```

Transpose

The transpose operator turns a column vector into a row vector, and vice-versa. It is implemented with a single quote. For example,

```
x = [1, 2, 3]

x =
    1    2    3
```

We can take the transpose of x to convert the row vector into a column vector, and assign to y by entering

```
y = x'
y =
    1
    2
    3
```

Transposing y will restore x, i.e.

```
z = y'
z =
    1    2    3
```

Exercise 4

1. Create a row vector x containing integer numbers from 1 to 30. Create another row vector y containing numbers 1, 0.9, 0.8, 0.7, . . . 0.1, 0 in this order.

2. From x create:
    a. A new row vector a containing first 10 elements of x
    b. A new row vector b containing elements of x with indexes from 15 to 25
    c. A new row vector c containing elements of x with even indexes.

If you code this correctly, for part 2 of this question, you should get

```
a =
    1    2    3    4    5    6    7    8    9   10
b =
   15   16   17   18   19   20   21   22   23   24   25
c =
    2    4    6    8   10   12   14   16   18   20   22   24   26   28   30
```

Task 3: Matrices

Defining a matrix is similar to defining a vector. To define a matrix, you can treat it like a column of row vectors (note that the spaces are required!):

```
A = [ 1 2 3; 3 4 5; 6 7 8]

A =     1    2    3

        3    4    5

        6    7    8
```

You can also treat it like a row of column vectors:

```
B = [ [1 2 3]' [2 4 7]' [3 5 8]']

B =     1    2    3

        2    4    5

        3    7    8
```

(Again, it is important to include the spaces.)

You can get the size of a matrix using the size command.

```
size(A)

ans =

   3    3
```

states that A has 3 rows, and 3 columns. If you're only interested in the number of rows of A, you can use a command

```
size(A, 1)

ans =

   3
```

and the number of columns can be returned as size(A, 2).

You can access elements of a matrix in a similar way in which you access elements of a vector. However, since matrices are two dimensional, two indices are required

```
A(2, 3)

ans =
```

5

This returns the element in row 2, column 3 of matrix A.

The notation used by Matlab is the standard linear algebra notation. Matrix-vector multiplication can be easily done. You have to be careful, though, your matrices and vectors have to have the right size! This will be discussed further at the mathematics primer.

```
v = [0:2:8]

v =     0     2     4     6     8

A*v(1:3)   Error using * Inner matrix dimensions must agree.

A*v(1:3)'

ans =
   16
   28
   46
```

Matrix Functions

Once you are able to create and manipulate a matrix, you can perform many standard operations on it such as +, -, ^, inv, etc.

```
A + B

    ans =

    2     4     6

    5     8    10

    9    14    16

A – B

    ans =

    0     0     0

    1     0     0

    3     0     0

 A^2 = A * A
```

```
ans =

25    31    37

45    57    69

75    96   117
```

```
A .* A

ans =

 1     4     9

 9    16    25

36    49    64
```

Note that **Matlab is case sensitive**. This is another potential source of problems when you start building complicated algorithms.

Finally, sometimes you would like to clear all of your data and start over. You do this with the *clear* command. Be careful though, it does not ask you for a second opinion and its results are final.

```
clear all;
```

will clear all the variables from memory.

```
clc;
```

will clear all the commands from the Command Window.

> ***Exercise 5***
>
> Create a 3 by 3 matrix with all ones. Create an 8 by 1 matrix with all zeros. Create a 5 by 2 matrix with all elements equal to 0.37.

Task 4: Control flow

Matlab has conditional statements that are similar to other programming languages.

if, elseif, else

Use if, elseif and else to execute functions only if a certain condition applies.

```
if (condition)
   statement1
else
   statement2
end
```

An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

Note: the elseif and else blocks are optional, but the end is required to indicate the end of the conditional statement.

Example:

```
A = [1 2 3 4 5];
if size(A, 2) > 4
   B = 1;
else
   B = 0;
end
```

for

Matlab supports different types of loops, like for and while loops. A for loop will execute statements specified number of times, and has the following structure:

```
for index= firstIteration : lastIteration
    statement
end
```

Example

This example initializes a matrix H to be a 10x10 matrix of zeros. It then uses a doubly-nested for loop to set individual elements in the matrix.

```
s = 10; H = zeros(s);
for c = 1:s
    for r = 1:s
        H(r,c) = r*c;
    end
```

```
end
```

while

---

while keeps running a loop while a condition is valid.

Here is an example that uses a counter to control execution of a while loop:

```
counter = 10; A = 0;
while counter > 0
    A =  A + 10;
    counter = counter -1;
end
```

if you forget the line counter = counter - 1; in the above example, the code will never terminate. If your code is stuck in an infinite loop, press CTRL + C in the Command Window to stop the program.

A general programming tip: Whenever you open a loop, parenthesis or if/elseif, immediately close it, then write the code in between. This will help you not to forget where your loop ends and will avoid programming errors.

Example:

```
if i>10

end
```

then go back between if and end, and start typing your statements. Also use increased indent in your code to make it more readable:

```
if i>10
   j = 1;
end
```

Exercise 6

1. Use a for loop to calculate the sum 1+2+3+...+300. To do this, create a variable (e.g. sumAll) that is initialized to 0 before the loop, and increases its value inside the loop. After the for loop, output the value of the sum.

   If you code this correctly, the answer is 45150

2. Modify your code to calculate the sum $1^2 + 2^2 + 3^2 + \ldots 400^2$.

   If you code this correctly, the answer is 21413400

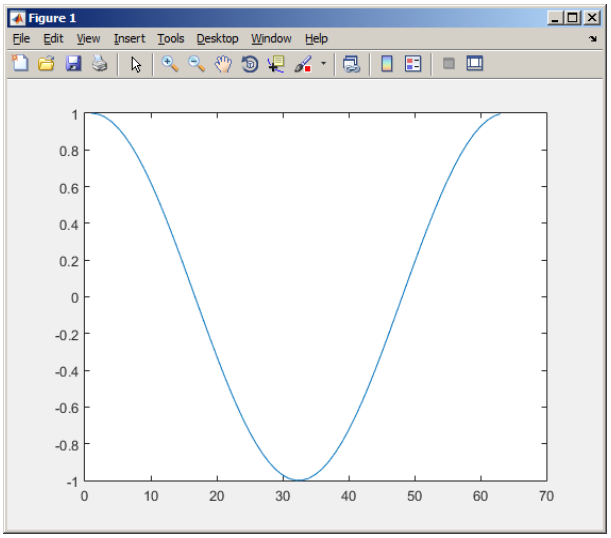
Task 5: Plotting

Another great feature of Matlab is its sophisticated plotting functions. With these, you can simply generate visualisations of your data by making a figure. The function plot is used to generate graphs of 1D functions.

For example, we can generate a plot

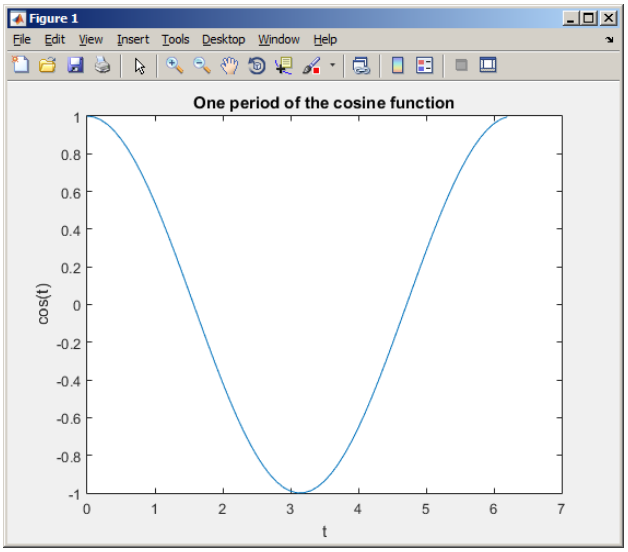
```
t = 0:0.1:2*pi;
y = cos(t);
plot(y);
```

And a new figure, Figure 1, will show the plot of y. Note Matlab is aware of pi, as 3.141527… The figure is shown below.



You may notice the x axis goes from 0 to 70. The reason is that the length of y is 63 elements. To plot y vs t, we can pass the t vector into the plotting function as well, i.e.,

```
plot(t, y);
```



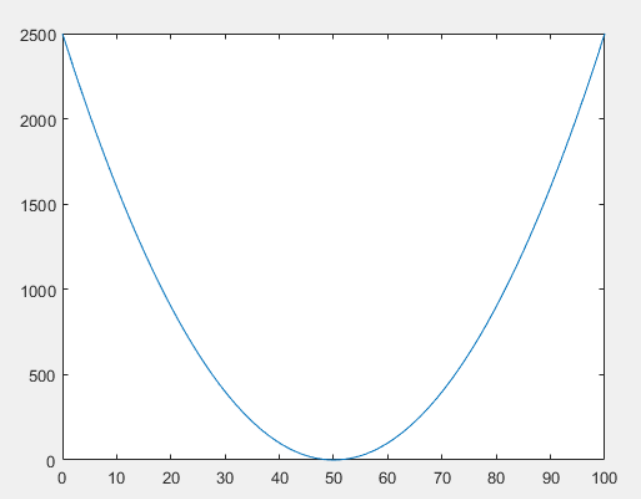
Here, we have also labelled the plot, using title, xlabel, and ylabel.

```
title('One period of the cosine function');
xlabel('t');
ylabel('cos(t)');
```

Exercise 7:

Plot the function f(t) = t^2 – 100t + 2500 for the range t = [0, 100]. *Hint: you can define t as a row vector, and use an element-wise multiplication for the t^2 term.*

The correct answer looks like this:



Task 6: Basics of images in Matlab

Matlab was originally designed for matrix manipulation. Since a standard image is a 2D array of numbers (i.e., a matrix), it is well suited to Matlab. Matlab allows you to load your own images (in a variety of standard formats, like .bmp, .png, .jpg) and video (.avi, .mp4), use a webcam, etc. For now, let's try loading one of the images that comes with Matlab. At the Matlab prompt, type

```
I = imread('peppers.png');
imshow(I);
```

This will load and display the peppers image. If you call the size function on I, it will return three arguments:

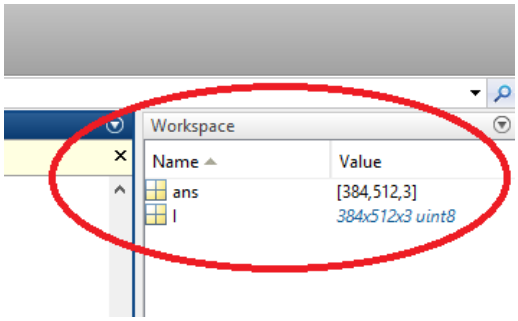
```
size(I)

ans =
  384   512     3
```

These correspond to the height, width, and number of colour channels, respectively. Note this is **not** the width, height, and number of colour channels. The reason why

the height is returned first is because when describing the size of a matrix, it is customary to provide the number of rows, *then* the columns. Matlab follows this convention.

If you look in the Matlab Workspace, you'll see the variable I is in memory and has a size [384, 512, 3].



Also, you'll notice that I is of type uint8. These are unsigned, 8 bit values that are in the range [0, 255]. You can convert the image to double as

```
J = double(I);
```

Now if you now call imshow(J), the image will not be correct. When given a uint8 image, imshow interprets the data so that 0 is the minimum intensity (in a colour channel) and 255 is the maximum. However, when given a double image, imshow interprets the data so that 0 is the minimum intensity and 1 is the maximum. However, you can normalise the image by dividing by 255.

	
imshow(J);	imshow(J/255);

Matlab is a powerful programming environment, and this lab has provided a (hopefully) gentle introduction. To learn more about Matlab, you can read the handout provided on Moodle. As mentioned earlier, at the Matlab prompt you can type "help" with the name of any function and you will receive a description of what it

does. Also, Matlab has documentation online and a large user community. These can be explored on the Mathworks website.