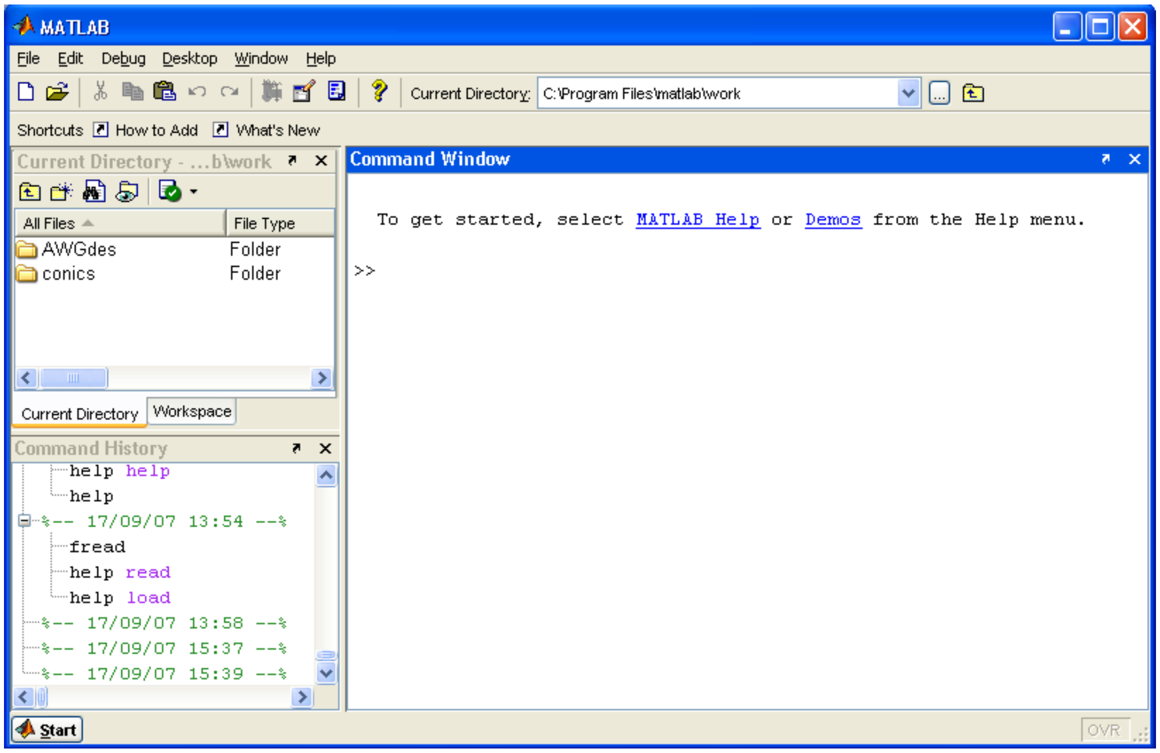
ELEC6021 An Introduction to MATLAB: Session 1

Lecturer: Rob Maunder rm@ecs.soton.ac.uk

Short for "Matrix Laboratory", MATLAB is a proprietary numerical computing environment. To perform numerical calculations MATLAB uses a *scripting* or *interpreted* programming language, meaning that the source code is not compiled such as in the C programming language but rather is interpreted each time the program is executed. MATLAB is commonly used in industry and academia for its ease of programming and vast libraries of functions. This session will guide you through the basics of the programming language and does not cover everything. On Windows systems, MATLAB can be started through the Start button. Once opened, the MATLAB environment should look similar to Figure 1.


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Program Files\matlab\work
Shortcuts How to Add What's New
Current Directory - ...b\work
All Files File Type
AWGdes Folder
conics Folder
Current Directory Workspace
Command History
Command Window
Start
OVR

```
help help
help
%-- 17/09/07 13:54 --%
fread
help read
help load
%-- 17/09/07 13:58 --%
%-- 17/09/07 15:37 --%
%-- 17/09/07 15:39 --%
```

```
  To get started, select MATLAB Help or Demos from the Help menu.

>>
```

Figure 1 The MATLAB environment.

1. Finding your way around

TASK 1 Identify components of the environment:

- *Command History Window*
- *Command Window*
- *Start Button and Launch Pad*
- *Help Browser*
- *Current Directory Browser*
- *Workspace Browser*

TASK 2 Documentation on MATLAB is readily available via the *Help Browser*, the Internet, or the `help` command. At the *Command Window* type:

```
help help
```

Press *Enter* to execute the command. This command will display text in the *Command Window* explaining what `help` does and how to use it. The most common use of the `help` is to find out what a certain command or function does in MATLAB.

2. The Basics

TASK 3 Try some simple operations. For scalar addition, subtraction, multiplication, and division type in the *Command Window:*

```
6+2
9-5
46*0.5
8/5
```

Notice how MATLAB always saves the result in a variable called `ans`. Good programming practice in MATLAB is to never use `ans` as a variable name. If a value is assigned to a variable `ans` is not used. Try:

```
a = 2
b = 5;
```

QUESTION What does the semicolon do?

MATLAB also has native support for complex numbers, both `i` and `j` have the value of the square root of –1. So be careful when assigning values to either of these variables, as that will override the default value. To enter a complex number just type:

```
4.9+j*5.6
a+j*b
```

3. Matrix Manipulation

MATLAB's biggest advantage is its relative ease of matrix manipulations. In other programming languages matrices are dealt with by using loops.

TASK 4 Create an array (also referred to as a vector); type in the *Command Window*:

```
a = [16 4 67 48 9 90]
```

As shorthand for sequence of numbers you can type:

```
a = 1:10
```

This will give you a sequence of numbers from 1 to 10 with unit interval. To change the interval type:

```
a = 1:2:10
a = 1.45:0.1:2.3
a = 0:pi/4:pi
```

TASK 5 Now to enter matrices, enter Dürer's matrix. The semicolon marks the end of the row.

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

This matrix is a famous magic square where all numbers in a row, column, or diagonal sum to 34. NB. MATLAB has a function `magic` for generating magic squares. Use the `sum` function by typing:

```
sum(A)
```

This is a row vector of the sums of the columns of A. To transpose A to get the sum of rows type:

```
A′
sum(A′)′
```

A column vector containing the row sums is displayed. Sum the diagonal elements with:

```
sum(diag(A))
```

Using the `help` command, look up how the `fliplr` function works to sum the anti-diagonal elements of A. Now, type the following:

```
A(1,4)
```

Subscripts This is specific indexing, such that it returns the value of row 1 column 4 of the matrix A. But it is essential that you master the colon, since this allows MATLAB to use its own internal routines which run much faster than loops using specific indexing and gives rise to much more readable code.

```
A(1:n,k)
```

displays the first n elements of the k-th column of A. So:

```
sum(A(1:4,4))
```

computes the sum of the 4th column. The colon by itself refers to all the elements in a row or column.

```
sum(A(:,end))
```

computes the sum of the elements of the last column of A. To reorder the columns in A type:

```
A(:,[4 3 2 1])
```

```
or
```

```
A(:,4:-1:1)
```

Both code statements say that for each of the rows of A reorder the elements in the order 4 3 2 1.

4. Array and Matrix Operators

In other programming languages matrix maths operations are commonly performed using loops. But in MATLAB operations are written intuitively. The operators as you would expect are shown in Table 1

Table 1 Array and Matrix Arithmetic Operators

Operator	**Matrix**	**Array** (element-by-element)
Addition	`+`	`+`
Subtraction	`-`	`-`
Multiplication	`*`	`.*`
Right Division	`/`	`./`
Left Division	`\`	`.\`
Power	`^`	`.^`
Transpose	`'`	`.'`
Brackets (are used to specify the order of evaluation)	`( )`	`( )`

TASK 6 Type the following:

```
x = (0:0.1:1)*pi
y = linspace(0,pi,11)
```

This generates two identical vectors. Try some scalar operations like:

```
z = x-2
w = z-x
```

Vectors in MATLAB are either single row or single column matrices. Create the vector c and find the outer and inner products by typing:

```
c = 1:5
c'*c
c*c'
```

Now, verify that:

```
W = rand(4)
b = rand(4,1)
x = W\b
```

solves the set of matrix equations Wx = b.

TASK 7 For array manipulations MATLAB also offers element-by-element operators to carry out common tasks shown in Table 1. Again with vector c.

```
c = 1:5
c'*c
c'.*c
```

Don't panic! The error is a common error you may come across when using array operators. Instead use the following code:

```
c.*c
```

Notice the difference between matrix and array operators?

5. Functions

Functions perform some processing on a number of inputs, in order to provide a number of outputs. So far, we've seen a few examples of functions, namely `sum`, `diag`, `linspace` and `rand`. Matlab has lots of built-in functions, which allow you to do all sorts of things.

TASK 8 For example the `rand` function can take two inputs and returns one output. It outputs a matrix of random numbers, where the first input specifies how many rows the matrix should have and the second input specifies how many columns. You can try this out using the command

```
rand(3,4)
```

which should output a 3×4 matrix of random numbers.

The `rand` function can also take only one input. In this case, it will output a square matrix of random numbers, having dimensions that are specified by the input. You can see this for yourself by inputting the command

```
rand(3)
```

Also, the `rand` function can take no inputs at all! In this case, it will output a single random number. Try this by inputting the command

```
rand
```

TASK 9 The `eig` function is an example of a function that can return two outputs. It outputs the eigenvalues and eigenvectors of the input matrix. You can try this out for yourself using the commands

```
X=rand(3)
[V,D]=eig(X)
```

You should find that X×V = V×D. Note that a function's outputs can be stored in a variable by using the equals operator. When there is more than one output, the variables should be listed inside square brackets, like in the example above.

If you're ever wondering what one of Matlab's built-in functions does, you can find out using the help command. Try this for the `rand` function by typing

```
help rand
```

TASK 10 Use Matlab's `help` command to find out about each of the other functions we've used so far.

6. Using Files

TASK 11 Outside of MATLAB, use a text editor (such as *Notepad*) to create a file called fred.dat containing a data array looking like:

```
1 2 3 4
5 6 7 8
8 7 6 5
4 3 2 1
```

Save the file in the MATLAB work directory and then enter in the MATLAB *Command Window:*

```
load fred.dat
```

Look at the variable `fred`. There are many commands and functions for saving and retrieving data such as `fread, fprintf, save,` etc.

TASK 12 So far, you have been typing commands into the *Command Window*. This is not a good way to realise larger programs. Scripts and functions can be stored in files with a `.m` extension. You can create these using the MATLAB editor, which is started from File-> New-> M-File or by clicking the *New M-File* button found on the top left-hand corner. Code written as a script, when executed is run as if typing each line into the *Command Window* yourself.

In the *Editor Window* type the following previous example code:

```
c = 1:5   %Generates a vector of numbers 1 to 5
c'*c      %Inner product by matrix operators
c'.*c     %This throws an error due to dimension mismatch
c.*c      %Multiply element-by-element
```

Note that the % sign in MATLAB is for inserting comments (these are ignored by the interpreter). It is good programming practice to write comments so that others and you are aware of what the code does. Save the file with the name `script1.m` in the MATLAB work directory. To call the script, click the run button on the *Editor Window* or type the script's name in the *Command Window,* in this case `script1`.

TASK 13 From now on type code into M-files as this makes life easier, unless otherwise instructed. If Matlab does not have a built-in function to do what you want, you can write your own function! To create a function in MATLAB, you write an M-file having a first line like:

```
function [y, z] = funcname(x, i)
```

The word function is a MATLAB keyword that declares a function. The variables in square brackets [ ] are the variables returned by the function. Brackets can be omitted if there is only one return variable. The variables in round brackets ( ) are input argument variables passed to the function and `funcname` is the function's name used to call it, which should be the same as that of the M-file. So, a function called `function1` should be saved in the M-file `function1.m`.

Differences between scripts and functions:

1. Scripts are lines of code that you would type into the *Command Window,* so variables are created in your workspace.
2. Functions have local variables (unless otherwise declared global by `global`), so they are destroyed once the function is returned.

Create a function called `logicarr` that uses **logical arrays**. These are useful for writing efficient codes. You can gain understanding of what they are and how they work by studying the following sequence of commands and the plots generated. Their meanings are later described in Table 2.

```
function z = logicarr(x,y)

figure(1);
plot(x,y);
z = (y >= 0).*y;
z = z + 0.5*(y<0);
z = (x <= 3*pi/2).*z;
figure(2);
plot(x,z);
```

At the *Command Window* create the arrays:

```
x = linspace(0,2*pi,30);
y = sin(x);
```

And call the function:

```
logicarr(x,y);
```

By not assigning the function to a variable the returned array is assigned to `ans`.

7. Plotting Graphs

A major benefit of the MATLAB environment is its useful graph plotting capabilities.

TASK 14 Creating a plot. Use the colon operator to make a vector having values in the range (0, 2π) and plot the sine of this vector.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y);
```

Remember the semicolons stop results printing to screen which makes the program run faster. Now label the axes and add a title.

```
xlabel('x = 0:2\pi');
ylabel('y = sin(x)');
title('Plot of the sine function','FontSize',12);
```

Find out how to print the figure.

TASK 15 Multiple data sets on one graph. Create another two functions dependent on x and put them in the `plot` function:

```
y2 = sin(x-0.25);
y3 = sin(x-0.5);
plot(x,y,x,y2,x,y3);
```

To make it easier to identify the different lines use a legend:

```
legend('sin(x)','sin(x-0.25)','sin(x-0.5)');
```

TASK 16 Specify line colours and styles

The `plot` function allows for many customisations such as line colours, line styles, etc. Use the `help` command to understand what the following example does.

```
x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
plot(x1,sin(x1),'r:',x2,sin(x2),'b+');
```

TASK 17 Imaginary and complex data

When plot is used on complex data the imaginary part is ignored, except when plot is given a single complex argument. For this case the real part is plotted versus the imaginary part. So `plot(Z)` is equivalent to `plot(real(Z),imag(Z))`.

```
t = 0:pi/10:2*pi;
plot(exp(j*t),'-o');
axis equal
```

TASK 18 Adding multiple graphs to a plot

By using the following command you can add more graphs to a plot as shown in the proceeding example.

```
hold on
```

This example code overlaps a contour plot with a psuedo colour plot:

```
[x,y,z] = peaks;
pcolor(x,y,z)
shading interp
hold on
contour(x, y, z, 20,'k')
hold off
```

TASK 19 Figure windows

To have more than one figure use the `figure` function every time you wish to have another *Figure Window* before plotting it. To create another figure but control its figure number, use:

```
x1 = 0:pi/100:2*pi
x2 = 0:pi/10:2*pi
figure(1)
plot(x1,sin(x1),'r:');
figure(2)
plot(x2, sin(x2),'b+');
```

TASK 20 Multiple plots in one figure

Use the `subplot` function for this:

```
t=0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1);
mesh(X);
subplot(2,2,2);
mesh(Y);
subplot(2,2,3);
mesh(Z);
subplot(2,2,4);
mesh(X,Y,Z);
```

grid lines can be set using:

```
grid on
grid off
```

8. Flow Control

TASK 21 User input and formatted output into your programs. To prompt a user for an input:

```
z = input('enter a value for z');
```

The user can input (for instance) [1,2,3; 4,5,6] which is assigned to the variable `z`. For formatted output look up how the `disp` will generate output more readable than the default and `fprintf` will produce user-specified formatted output.

TASK 22 The `if` statement in MATLAB follows the format of:

```
if condition 1
```

```
     statement 1
elseif condition 2
     statement 2
else
     statement 3
end
```

The conditions are expressions made from using logical and relational operators as described in Table 2 and Table 3. Try this example code:

```
temperature = input('enter a temperature ')
if temperature >= 90
  disp('It's getting hot');
elseif temperature < 90 & temperature > 50
  disp('This is just right');
else
  disp('I think I will get my coat');
end
```

Table 2 Relational Operators

Operation	**Operator**
Equal to	==
Not equal to	~=
Less than	<
Greater than	>
Less than or equal	<=
Greater than or equal	>=

Table 3 Logical Operators

Operator	**Description**
`&&`	Returns logical 1 (true) if both inputs evaluate to true, and logical 0 (false) otherwise.
`||`	Returns logical 1 (true) if either input, or both, evaluate to true, and logical 0 (false) otherwise.

To test for equality between scalars `if A == B` works, but for matrices tests where the two matrices are equal, resulting in a matrix of zeros or ones. The proper way to test for equality between two variables is `if isequal(A,B)`. Functions that are helpful for reducing the results of matrix comparisons to scalar conditions for use with `if` are: `isequal and isempty.`

TASK 23 The switch statement follows the format below.

```
switch switch_expr
 case case_expr1
    statement 1
  case case_expr2
    statement 2
  otherwise
    statement 3
end
```

The temperature comparator can also be described by:

```
switch (1)
  case temperature >= 90
    disp('It''s getting hot');
  case temperature > 50
    disp('This is just right');
  otherwise
    disp('I think I''ll get my coat');
end
```

Only the first matching case is executed. There must always be an `end` to match the `switch`.

TASK 24 The `for` loop. This example repeats a statement a predetermined number of times:

```
function factx = fact(x)
%this  function  accepts  a  scalar  input  and  return  its
%factorial
a=1;
for k=1:x
  a=a*k;
end
factx=a;
```

Note the matching `end` at the end. **N.B.** Do <u>not</u> use `for` on all the elements of an array and do <u>not</u> use loops when you can use the colon notation. MATLAB is an interpreted language, not compiled like C and consequently long `for` loops and complex logic is generally slow to execute. Check that this function operates correctly when you call it from the Command Window.

TASK 25 The `while` loop repeats a group of statements an indefinite number of times under the control of a logical condition. Here is a complete program that finds the roots of a polynomial by interval bisection.

```
a=0; fa=-Inf;
```

```
b=3;fb=Inf;
while b-a > eps*b
  x=(a+b)/2;
  fx=x^3-2*x-5;
  if(sign(fx) == sign(fa))
    a=x; fa=fx;
  else
    b=x;fb=fx;
  end
end
x
```

Two statements that you should also be aware sometimes useful for writing loops are the `continue` and `break` statements. The `continue` statement passes on to the next iteration of a `for` or `while` loop. The `break` statement is an early exit from a `for` or `while` loop.

9. Further Reading

There are many books on MATLAB and how to use MATLAB certain fields. The University's library has quite a few. There is also a vast amount of documentation online on MATLAB functions.

A first place to look online would be:
http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html

or a popular starting point if the `help` command is not adequate is www.google.com.