SCHOOL *of* ENGINEERING and ELECTRONICS
MSc in Signal Processing and Communications

Digital Signal Processing Labs (PGEE11108)

Laboratory Session 1:

MATLAB Introduction & Revision

Introduction
MATLAB is a very powerful, high level language; It is also very easy to use. It comes with a wealth of libraries and toolboxes that you can use directly, so that you don't need to program low level functions. It enables us to display very easily results on graphs and images. To get started with it, you need to understand how to manipulate and represent data, how to find information about the available functions and how to create scripts and functions to generate programs. This lab will revise the basic notation and operations available in MATLAB before we go on to investigate DSP.

Most students will have used MATLAB before in their undergraduate studies (either in DSP or Signals & Systems modules) and therefore the first part of this lab should be treated as a quick revision exercise of the basic structure of MATLAB and how to use it.

For those who have not used MATLAB previously, or who have forgotten how to use it, the School of Engineering has an interactive Introduction to MATLAB course available online that you should work through in your own time. The home page for this course is:

https://www.see.ed.ac.uk/auth/teaching/courses/matlab/

The Introduction to MATLAB notes are available at:

https://www.see.ed.ac.uk/auth/teaching/courses/matlab/Intro-to-MATLAB.pdf

This is a self study course using online notes, screen casts and exercises.

Please make use of this self study course.

For subsequent labs you will be expected to be familiar with the MATLAB language and environment.

Section A 'Revision' is a rapid refresher of the MATLAB notation and syntax before moving on to the DSP. Introductory Exercises in Section B; these introduce some of the MATLAB features to be used during the DSP course.

Section A: MATLAB Revision

MATLAB is a high performance language and software for technical computing. It allows

- Maths and computation
- Algorithm development
- Modelling, simulation and prototyping
- Data analysis and visualisation
- Scientific and engineering graphics

It is an interactive system (all data always in memory) with the following key facts:

- The basic data element is an array
- No declaration (or prototyping of functions) required
- All data represented as doubles
- In script mode or command line mode, data is always accessible and in memory.

To begin you need to start MATLAB

Open a terminal window and type “matlab” at the command prompt.

1. Help sources

There are extensive help facilities in the MATLAB environment. Three of the most commonly used are represented below.

1.1 MATLAB help window – type “helpwin” at the command prompt. This will launch a new window where all MATLAB core functions are categorised. Alternatively select “Full product family help” from the *help menu*, or click on the *question mark* icon. To find a function, select the appropriate category then select the operator you need to learn more about its syntax and operation.

1.2 MATLAB demos – type in “demo” at the command prompt. This takes you to a window where simple demos show different capabilities of the software. This is a good starting point for new users.

1.3 Help - by typing in “help <function_name>” you can receive help on a particular function. This is useful if you know the name of the function but are not sure about the syntax or exact functionality of a MATLAB function.

1.4 The `lookfor` command

If you do not know what is the name of the MATLAB command you want to use the `lookfor` command is what you are looking for.

For instance, if you are looking for the cosine or related functions, type

```
>> lookfor cosine
```

MATLAB comes back with all Matlab functions related to cosine with a short description as:

```
ACOS Inverse cosine.
ACOSH Inverse hyperbolic cosine.
COS Cosine.
COSH Hyperbolic cosine.
TFFUNC time and frequency domain versions of a cosine modulated
Gaussian pulse.
DCT2 Compute 2-D discrete cosine transform.
DCTMTX Compute discrete cosine transform matrix.
DCTMTX2 Discrete cosine transform matrix.
IDCT2 Compute 2-D inverse discrete cosine transform.
CHIRP Swept-frequency cosine generator.
DCT Discrete cosine transform.
FIRRCOS Raised Cosine FIR Filter design.
IDCT Inverse discrete cosine transform.
DCT Discrete cosine transform.
IDCT Inverse discrete cosine transform.
dctold.m: %DCT Discrete cosine transform.
idctold.m: %IDCT Inverse discrete cosine transform.
BLKACOS This block defines an output angle that is the arccosine of
the input.
BLKCOS This block defines the output as the cosine of the input.
BLKCOSASIN This block defines the cosine of an angle whose sine is u.
```

```
BLKCOSATAN This block defines the cosine of an angle whose tangent is
u1/u2.
```

Now type `help cos` to get details on the cosine function.

```
>> help cos
```

2. Simple MATLAB operations and variables

2.1 General maths expressions can be entered directly at the command prompt. This uses MATLAB as a calculator. Type the following then hit enter:

a) addition

```
>> 5 + 2
```

b) raising to the power

```
>> 2^4
```

c) trigonometric expressions

```
>> sin(pi/2)+exp(1)
```

The result from each operation is stored in the system variable *`ans`*. If you type "`ans`" the result of the last operation will be given.

Built-in functions: MATLAB contains a large range of built-in functions to perform basic mathematical operations. Look up the following functions using help: `log10, atan, tanh`.

2.2 Variables can be specified to store the result of an operation. These can then be used for further calculations. The example below shows how variables can be assigned.

```
>> x = 5 + 2;
>> y = exp (0);
>> z = x + y;
```

Comment: the semicolon at the end of each instruction line is used to suppress the echo of the result of the operation. To see the contents of a variable type the variable name and press enter: e.g. the following would give the contents of the variable z:

```
>> z
```

3. Complex numbers

Complex numbers can be handled in MATLAB and there are several supporting functions which simplify the manipulation of complex numbers.

3.1 Complex numbers can be stored in variables just like integers as shown above, e.g

```
>> x = 3 + 4j
```

All the standard mathematical operations can be performed directly on complex numbers, i.e. addition, subtraction, multiplication and division.

Exercise: Take the complex numbers and set a = 3 + 5j and b = 2 + 11j. Then apply the four operations described above and confirm the answer.
[Answer: `a+b=5.0+16.0i, a-b=1.0-6.0i, a*b=-49.0+43.0i, a/b=0.4880- 0.1840i`]

3.2 Use help to find out how the functions *real* and *imag* can be used to display only the real or only the imaginary parts of a complex number.

3.3 *abs*, *angle* and *conj* are functions that can be used to calculate the magnitude, angle and conjugate respectively, of complex numbers.

Exercise: Use the numbers from 3.1 to try out the above functions and confirm with your manual calculations.

4. Vectors and Matrices

Vectors, matrices and their manipulation lie at the heart of MATLAB. There are several built in functions which provide fast and efficient matrix manipulations. Most tasks should be attempted using these features instead of low level element-wise arithmetic.

4.1. Vectors, which are just one row or one-column matrices, can be assigned in several ways. Below some common examples are shown.

```
>> x = [0 1 2 3 4 5];
```

A vector's values are assigned by directly entering the values

```
>> x = 0:5;
```

This gives the same result as previous example. The vector is assigned values 0 to 5 with an increment step of 1. The colon operator used above has several applications within matrix manipulation in MATLAB. Use the help facilities to familiarise yourself with the COLON operator.

```
>> x = 0:0.01:5;
```

Here another example is shown where x is assigned values ranging from 0 to 5 with an increment step of 0.01.

4.2 Matrices are similar to vectors but have several rows and columns. The assignment is also similar to that of the vectors as was shown earlier. Type the following

```
>> A = [0 0; 1 1; 2 2]
```

It is also possible to assign a matrix of zeros or ones of a predefined by using the *ones* and the *zeros* operators, e.g.

```
>> B = zeros(2,2)
```

creates a 2 x 2 matrix of zeros.

4.3 A vector x (or matrix) can be transposed using the following notation:

```
>> x'
```

Exercise: Create a vector **x** containing the number sequence of 0 to 20 in steps of 0.2. Then create another vector **y** containing all ones and of the same size as the previous vector **x.**

4.4 Accessing vector/matrix components. Individual elements of a vector or matrix can be accessed using the following syntax. Type:

```
>> x = 1:10;
```

```
>> x(10)

>> x(2:2:10)
```

5. Graph representation of results.

MATLAB provides powerful tools for graphical representation of calculated results. These can be in both two and three dimensions. For the purpose of the DSP labs, mainly 2-D graphs will be used.

5.1 Launch the MATLAB demos and run the **Matlab → Graphics → 2-D plots** demo. This should give a general overview of the plot functions available. It may be worth making a note of these for further reference.

Exercise: The code below generates one period of a the signal $x(t) = sin(\omega t)$ at 100Hz. Enter the following lines that will plot the graph as amplitude against time (i.e. `x` against `t`).

```
>> f = 100;              % frequency is 100Hz
>> t = 0:1/(20*f):1/f;  % a time vector to cover 1 period of
                        % the wave with 20 points calculated
>> x = sin(2*pi*f*t);   % the sinewave signal
>> plot(t,x);           %add a plot command here
```

Note: the %-sign is used for writing comments in MATLAB. You do not have to enter the comments for the code to run but comments are essential for explaining the code and later remembering what it represents!

5.2 In order to give more information about the presented information you would need to label the graph and its axis. Further, you may want to change the scaling on your axis instead of using the default auto scaling mode.

Using the help facility, find out how to use the following functions in order to make your graphs clearer.

`Title` – to generate a title on top of the graph to describe e.g. what the graph shows

`xlabel`, `ylabel` – to label the x and the y axis respectively

`axis` – to customise x and y scaling of the axis

`grid` – shows a grid on the graph

`legend` – shows a legend on the graph. Useful when having several plots on the same figure

5.3 You may need to plot several graphs on one grid, for example, when comparing simulated results against theoretical results. Below are the most common functions you may need. Look up the detailed syntax using the MATLAB help window (Two dimensional graphs) or the help command.

`Figure` – a function that starts additional plotting windows.

`Subplot` – a function which splits up the current plot window in an array of plotting areas.

`Hold` – by typing in "hold on" the current plot is kept static and a second graph can be plotted on the same axis.

`Close` – is used to close selected, or all plotting windows.

6. Image Processing

There are a range of built-in functions that allow the reading, writing and viewing of images in MATLAB. There are also a number of standard images that MATLAB has pre-loaded, such as the "peppers.png" image:

6.1 Try the following:

```
im1 = imread('peppers.png'); % Reads in a standard image
im2 = randn(384,512); % Creates a 384x512 noise image

whos % Note that images are stored in uint8
```

Now view the two images using

```
imshow(im1); % Display matrices as images
imshow(im2);
```

or

```
figure(1)
imshow(im1); % Display matrices as images
figure(2)
imshow(im2); % Same thing with 2 windows.
```

Try playing with the figure menus and especially the export button that allows saving images and figures. Type help figure for more information.

6.2 Color images are composed of 3 stacked matrices in the form of: `im1(:,:,1)`(red), `im1(:,:,2)` (green) and `im1(:,:,3)` (blue). Try:

```
clf; % Clear figure
imshow(im1(:,:,1)) % Displays Red component of the image
```

and

```
clf;
subplot(3,1,1)
imshow(im1(:,:,1)) % Displays Red component of image
title('Red Component');
subplot(3,1,2)
imshow(im1(:,:,2)) % Displays Green component of image
title('Green Component');
subplot(3,1,3)
imshow(im1(:,:,3)) % Displays Blue component of image
title('Blue Component');
```

BE CAREFUL, ARITHMETIC OPERATORS DO NOT WORK ON UINT8 TYPE. To manipulate and image we can convert it to a double type:

```
im1 = im2double(rgb2gray(im1)); % Converts image to grey level
                                % and to a double type.
```

6.3 Operators and images - once converted to a double type it is possible to perform mathematical operations on images. Try

```
ima = im1 + im2;
imshow(ima);
```

7. Programming constructs

Most of the standard programming constructs as seen in, e.g. *C* and *Pascal* are also available in MATLAB. You should be familiar with their functionality by now although the syntax may be slightly different. Below are some examples for you to try.

7.1 if…else

```
>> a = randn;                  % let a be a random number
>> b = randn;                  % let b be a random number
>> if (a > b)                  % if a is greater than b then
clc                            % clear the screen
disp('a is greater than b')    % display message
else                           % otherwise
clc                            % clear screen
disp('b is greater than a')    % display message
end                            % end of if…else statement
```

7.2 for loops

```
>> x = 0:0.01:10;              % a vector ranging from 0 to 10
>> sine = sin(x);              % sin values for all x
>> subplot(211)                % first plot in figure(out of 2)
>> plot(x,sine);               % plot the graph
>> for i = 1:length(sine)      % repeat for all values in
                               % vector sine
      if(sine(i) < 0)          % if element is less than zer
            sine(i) = 0        % set that element to zero
      end
end
>> subplot(212)                % second plot in figure one
>> plot(x,sine)                % plot truncated sine
```

7.3 while

```
>> a = 0:99;                   %100 element matrix

>> i = 1;                      %initialise i
>> while i < length(a)         %loop until i = 100;
      a(i) = i;                %asign value of i for each ith
                               %element of a
      i = i + 1;
      end
```

7.4 Switch and case:

```
switch (expression)
      case value1
            action1;
      case value2
            action2;
      otherwise
            default action;
      end
```

8. Scripts and functions – in .m files and general MATLAB commands

8.1 A script is a file containing a sequence of commands to be executed and to provide a desired output. A MATLAB script is recognised by its file extension .m. To execute a script, the scripts name is entered at the prompt. If you keep your files in a separate

directory which is not on the MATLAB paths you have to make sure that you are in that directory when trying to execute the script. To check your current working directory you can type

```
>> pwd
```

To change your working directory

```
>> cd <dirname>
```

To list the contents of a directory

```
>> ls
```

A script file can be written and edited using MATLAB's editor. Open the editor and enter the following commands. Save the file as **wavescript.m**:

```
% This script produces a sampled sine wave
%
N = 64;                          % number of samples
T = 1/128;                       % a sample every T s
n = 0:N-1;                       % discrete time vector

x = sin(2*pi*5*n*T);             %5Hz sin signal

stairs(n*T,x)                          %plot and label the graph
title('5Hz sine wave')
xlabel('n*T'), ylabel('x(n*T)')
```

Now, make sure you are located in the same directory in MATLAB as the script file you just wrote. If so, type

```
>>wavescript
```

and you should get a sampled sine wave in your figure window. Note the comment line on top of the script file.

8.2 Functions differ from scripts in that they can take in data in form of arguments and the return data in variables too. A function must start with a function definition constructed as follows:

```
function var = func_name(arg1,arg2...argn)
```

or if several return variables are required then:

```
function [var_1,..., var_n] = func_name(arg1,...,argn)
```

Below the script shown in 8.1 is modified to a function.

```
function x = wavefunc(N,T,f)
%This function produces a sampled sine wave
%as x = sin(2pifnT)
%
% usage:
%       x = wavefunc(N,T,f)
%
% N = number of samples
% T = a sample every T
% f = analog signal frequency
%
n = 0:N-1;                              %discrete time vector
x = sin(2*pi*f*n*T);                    %sin signal

stairs(n*T,x)                           %plot and label the
```

```
title('Hz sine wave')                %graph
xlabel('n*T'), ylabel('x(n*T)')
```

9. Debugging a program

   Debugging is easy in Matlab as it is an interpreted language. You can go into the editor and set up the options and breakpoints where you want. The editor will be invoked and show the line where the error has occurred. In this case the K symbol appears in the command line.

   As Matlab is interpreted you can change the values of variables and continue the execution using:

```
K>>dbstep              % continues by one line
K>>dbcont              % continues until next stop
```

   To see all the options of the debugger, use `help dbstop`.

For a more through introduction to MATLAB, go to the School of Engineering: Introduction to Matlab interactive course.

Section B: Introductory DSP MATLAB Exercise

Rotating Phasor Animation

The MATLAB code `rotating_phasor.m` can be found in the resources directory. It is meant to simulate a rotating phasor and plot its real and imaginary components as functions of time ($\cos \omega t$ and $\sin \omega t$).

B1. Run this programme, study and understand its operation

Currently the code only plots $j \sin \omega t$.

B2. Modify the programme code to plot $\cos \omega t$ in the **lower left** window with time running vertically down the y-axis to parallel that of the imaginary component.

Decaying and Growing Phasor Animation

B3. Modify your code to plot:

a) a phasor with exponentially decaying amplitude

b) a phasor with exponentially increasing amplitude (to make this work you should try to make the FINAL amplitude unity)