University of Sheffield
Department of Electronic and Electrical Engineering
SECOND YEAR LABORATORY CLASSES

Matlab Laboratory

AIMS

By the end of the laboratory students should be able

i. to create user defined m-file scripts and functions to perform mathematical calculations
ii. to import data from file and perform curve fits
iii. to synthesise music
iv. to obtain the magnitude/phase spectrum of a given signal
v. to compute system response using Laplace Transform
vi. to use Fourier transform techniques to perform simple filtering tasks

ASSESSMENT METHOD

Your work will be assessed in the laboratory and it is your responsibility to ensure that you present your work to an instructor or demonstrator. You **must** show your instructor/demonstrator that your program works correctly and provides the required outputs.

1. INTRODUCTION

Matlab is a general purpose programming language and is widely used in scientific and engineering communities. In addition to providing the usual programming language building blocks (loops, conditional statements, functions, data types, etc.) it provides an extensive suite of prewritten functions suitable for processing and visualising data, system modelling and much more. The purpose of this laboratory is to introduce you to Matlab and provide you with experience of data processing and system modelling.

To ease your introduction to Matlab the laboratory has been divided into four sections. The first section introduces basic programming terminology, data input and plotting functions. In the second section you write your own user defined functions that you extend to generate music. The third section shows how Matlab may be used to model simple 1st-order circuits. In the final section you use Fourier transform techniques to perform filtering on signals and images.

The exercises in this laboratory will be based on prototype program code provided in this document. To ease readability all program code will be shown in the `Courier` font.

Further information regarding Matlab can be found in [1, 2].

2. MATLAB FUNDAMENTALS

The name Matlab stands for Matrix Laboratory and the matrix forms the basic data type that Matlab's functions operate on. Matlab can operate on a matrix using linear algebra (add, subtract, dot and cross product, invert, transpose, etc.) or they can use matrices as containers for data such as text strings. In programming terms a 1×1 matrix is a standard scalar-type variable and can contain integer, floating point or character data. A n×1 matrix (a column vector) looks like an n-dimensional array just like you would find in the C programming language. A 1×n matrix is a row vector which is just the transpose of a column vector. A m×n matrix looks like an m×n dimensional array. Having introduced matrices and vectors we can now look at some basic commands for defining and manipulating these.

Open Matlab and find the 'Command Window' – you enter commands directly after the '>>' prompt and the flashing cursor.

To generate a row vector starting from 0 and ending at 10 type the following:
```
y=0:1:10
```

To suppress the output printing to the screen include a semicolon at the end of your command:
```
y=0:1:10;
```

To find out the length of the vector y:
```
length(y)
```

You can scale the vector easily:
```
z=2*y
```

You can also perform addition and multiplication between 2 vectors:
```
x=0:pi/10:pi
z=x.*y
z1=x+y
```

You can type `help` to obtain more information on a given command. For example `help sin` would provide information about the sin function.

Here is a summary of some of the commands

Arithmetic	`+, -, *, /, ^`
Relational logic	`>, <, >=, =<, ==`
Logical operation	`or, and, not`
Elementary functions	`sqrt, sin, cos, tan, exp`
Other functions	`mod, floor, round, max, min, find, abs, log`

Use `help` whenever you need to check the syntax for a chosen command.

To generate a sequence of *x* zeros (or ones) use `zeros(1,x)` (or `ones (1,x)`). For example:
```
zeros (1,3)
ones (1,3)
```

Now try to generate and plot a sinusoidal curve. First specify a time sequence with a time increment *dt* and a duration *dur* (you have choose appropriate values for *dt* and *dur*)
```
t=0:dt:dur
```

Generate a sinusoidal curve with an angular frequency of 5 rad/s using
```
y=sin(5*t)
```

To plot the curve use

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

Label the x-axis and y-axis

```
xlabel('time (s)'), ylabel('y=sin(5 \times t)')
```

The character sequence `\times` is translated to the multiplication symbol in the y-axis label. Many other mathematical, Greek and general purpose symbols are supported – refer to the Matlab documentation for further information.

To include a title on the figure type

```
title('Sinusoid with frequency = 5 rad/s')
```

2.1 Complex numbers and the Mandelbrot set

Matlab has a number of built-in functions to manipulate complex numbers. To define a complex number *C* we can just simply write

```
C = 1 + 2i
```

We can then add another complex number to this using

```
D = 2 + 3i + C
```

Subtraction, multiplication and division can also be perform using the usual operators `-`, `*` & `/`.

Matlab also has functions for extracting the real or imaginary parts (`real` & `imag`), obtaining the modulus (`abs`) and argument (`angle`) from complex numbers.

We will now use complex numbers to generate the Mandelbrot set. The Mandelbrot set was popularised in the 1970s and it is a set of points that lie inside a fractal boundary that exhibits intricate detail and the closer one examines this boundary, the more detail it displays. What is surprising is that this detail arises from a very simple equation.

The Mandelbrot set is created from the complex mapping $Z_{n+1} = Z_n^2 + c$ where c is a point on the complex plane, n is the iteration number and $Z_0 = 0$. As this mapping is repeatedly applied two distinct things will happen: i) as $n \rightarrow \infty$ Z_{n+1} will diverge and shoot off to infinity indicating that point c is outside the Mandelbrot set, or ii) Z_{n+1} will remain within the boundary defined as the Mandelbrot set. In practice only a limited number of iterations are applied and if the result remains less than some limit ε, that is $|Z_{n+1}| < \varepsilon$, the point is said to lie within the Mandelbrot set. To plot the Mandelbrot set one must evaluate the mapping for each point c in the complex plane that is under consideration. If $|Z_{n+1}| < \varepsilon$ then a black pixel is plotted indicating that the point is part of the Mandelbrot set.

You have to write a program to plot the Mandelbrot set based on the algorithm given below. To begin, open the program code editor using `edit mandelbrot`. You can now type instructions in the editor just as you would in the Command Window.

1. Using `for` loops generate a point in the complex plane, $c = x + iy$. To view the whole Mandelbrot set both x and y should lie in the range ±3 and you only need to consider 100 points per line.
2. Iterate each point 16 times.
3. If $|Z_{16}| < 2$ then plot the point x, y on the screen using `scatter(x,y)`.

When you have finished typing, save your program and run it from the Command Window by typing `mandelbrot`.

Demonstrate and explain your work to an instructor/demonstrator.

This program can be improved in a number of ways:

- You can zoom in to specific regions by modifying the limits on the *x* and *y* co-ordinates.
- You can make a colour plot by setting the pixel colour according to the iteration number *n* that exceed the limit $|Z_{n+1}| < \varepsilon$.
- Improve processing speed by storing the result in a pre-allocated matrix (array) and then displaying the final result.
- Improve the display by using other plotting facilities including `surf, mesh, shading flat & view`.

2.2 Importing and exporting data and performing curve fits to data

Matlab has a number of functions for importing and exporting data. The commands `load` and `save` retrieve and store variables within the workspace. File operations can be carried out using `fopen`, `fread`, `fgetl`, `fwrite` and `fprintf` commands, among others. Commands for reading comma separated variable (CSV) files include `csvwrite` and `csvread`. A really useful feature of Matlab is the *Import Data* wizard that can be found on the *HOME* tab.

1. Go to the EEE260 Teaching Resources page and download the data file titled "transil.dat" remembering where you saved it.
2. Using the Import Data wizard open "transil.dat" and import the data into the workspace.
3. Plot columns 2 & 3 with column 2 on the horizontal axis.
4. Label the horizontal and vertical axis as "H (kA/m)" and "B (T)" respectively.
5. Set the title as "B-H curve for TRANSIL steel".

You should now have a plot of the B-H curve for TRANSIL steel. You will now fit a curve to this data using the `p=polyfit(X,Y,N)` command which fits an N^{th} order polynomial to the data (X,Y) where p is a row vector containing the coefficients of the polynomial in the format $p=[p(1)\,X^N + p(2)\,X^{N-1} + \ldots + p(N)\,X + p(N+1)]$. The polynomial can be evaluated at a particular value of x using the command `y=polyval(p,x)`.

6. Use `polyfit` to provide coefficients to a 19th order polynomial for the B-H data.
7. Using `polyval`, plot the 'curve fit' (in a different colour) alongside the B-H curve you plotted previously. You will need to use the `hold` command to stop the previous plot from being overwritten.

Demonstrate and explain your work to an instructor/demonstrator.

The `polyfit` command can also be used to fit curves to logarithmic data. The file "diode.csv" contains the experimentally measured V-I response of a 1N4148 diode. Use `polyfit` to obtain a curve fit the Shockley equation $I_d = I_s \exp(\frac{V_d}{V_t})$ where I_d and V_d are the diode's current and voltage respectively, I_s is the saturation current and V_t is the thermal voltage. Note the Shockley equation is only valid for positive voltages. Plot the measured data and your curve fit on the same figure, label all axes, provide a title and include a legend for the two curves.

Demonstrate and explain your work to an instructor/demonstrator.

3 SYNTHESISE MUSIC

3.1 Exercise 1

In this exercise you will construct your own user defined function and save it as an m-file. This function will be used to generate a sinusoid $x(t) = A\sin(2\pi ft + ph)$ with an amplitude A, a frequency *f* in Hz and a phase *ph* in rads. The program code shown in Listing 1 provides a prototype function for you to complete using knowledge you gained from the previous section.

Open the program code editor using `edit generate_sine` and then enter the code from Listing 1. Complete the prototype function so that it generates a sinusoid waveform with a defined sample frequency *fs* and *N* points per cycle. Your function should also plot the waveform (you must include axis labels and titles in the plot). Note the `%` symbol is used to comment out a line of code similar to the double slash (`//`) syntax used in C. The areas for you to complete are marked with a double underscore (___), for example, `fs = __`, in the listing below.

```
function generate_sine(A,f,ph)
%generate_sine function will generate a sinusoid with an amplitude A, a
frequency f in rad/s %and a phase ph in rad

fs = __           ; % select a sampling frequency fs >> f whenever possible
N= __      ; % specify the number of points/samples to generate
t= 0:1/fs:__         ;       % specify the time sequence
x=__        ;      % define the signal x(t)

%plot the signal
```

Listing 1

Now run it using different values of amplitude, frequency and phase (*A*, *f* & *ph*).
`generate_sine(A,f,ph)`

We can inspect the signal in the frequency domain by using the `fft` function (FFT is the Fast Fourier Transform). Add the code below to your function.

```
fr = (fs/N)*[(0:N/2) (–(N/2-1):-1)]; % specify frequency from –(N/2-1)*(fs/N) to
                                     % N/2*(fs/N)
X=fft(x)/N;                          % obtain the magnitude spectrum
```

Now plot both the time domain and frequency domain signals using the command subplot

```
subplot(2,1,1), plot(

subplot(2,1,2), stem(     %use stem to plot discrete signal
```

Demonstrate and explain your work to an instructor/demonstrator.

3.2 Exercise 2

In this exercise you will write a function to generate a tone from a piano keyboard, for example A_4. The layout of the keyboard is shown in Figure 1. The key A_4 has a frequency of 440Hz. There are 12 keys in one octave and the ratio of the frequencies between successive notes is $2^{1/12}$. For example the frequency of B_4 (2 keys above A_4) is $440\times2^{2/12}$ = 493.9Hz.

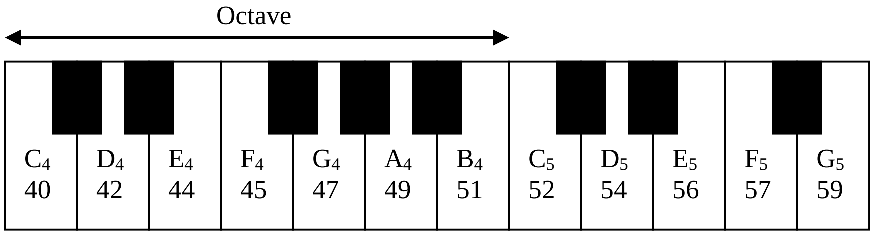

Figure 1: Typical piano keyboard

Matlab can play sound waveforms using the `soundsc(signal,sample frequency)` function which plays a signal at a specified sample frequency. For this exercise you have to write a function to generate a sine waveform. The frequency of sine waveform should equate to the required piano key and be of sufficient length to be played for a 0.5 second duration using the `soundsc` function. Using a sample frequency of 11025Hz and key A_4 as a reference modify the prototype function in Listing 2 to provide the required functionality.(Hint: Use the key reference A_4=0, B_4=2, C_5=4, etc.)

```
function tone=note(keynum,dur)
% generate a sinewave of frequency of keynum with a duration dur

fs=11025;
keydiff= __ % difference between keynum and A4
f= __ % frequency to the note
t= __ % time
tone=sin(2*pi*f*t);
```

Listing 2

Try to play different tones.

```
soundsc(note(keynum,dur),fs)
```

Demonstrate and explain your work to an instructor/demonstrator.

3.3 Exercise 3

Now you will create a function to play the key sequence C_4, D_4, E_4, F_4, A_4, B_4. Modify the prototype program code in Listing 3 so that the waveforms associated with each key are generated and stored in a single vector to be played by `soundsc`.

```
function play_tones
% play the sequence defined

keys = [__                             % fill in the key number here
durs = [__                             % specify the durations
fs = 11025;                         % sampling frequency
xx = zeros(1,sum(durs)*fs + 1);   % allocate a vector large enough for this
                                  % sequence
n1=1;
for k=1:length(keys)            % for loop counting through the key sequence
keynum = keys(k);
keydur = __                     % duration of play for a single key/tone
tone = __                       % the tone itself
n2 = n1 + length(tone) – 1;     % store this tone and make room for the next
xx(n1:n2) = xx(n1:n2) + tone;
n1=n2;
end

soundsc(xx,fs)
```

Listing 3

Demonstrate and explain your work to an instructor/demonstrator.

You might consider how the file manipulation commands (`fopen, fscanf & fclose`) may be used to read music from a file.

4 DYNAMIC SYSTEMS ANALYSIS

Complicated and tedious calculus operations in time domain (e.g: differentiation and integration) can be transformed into much simpler algebraic operations in frequency domain/s-domain (addition and multiplication) by using the Fourier/Laplace Transform. For instance the time domain differential equation for the 1st-order low-pass filter circuit shown in figure 2 can be transformed into s-domain equation (a transfer function) via Laplace Transform and Matlab has a plethora of built-in functions to simulate transfer functions.

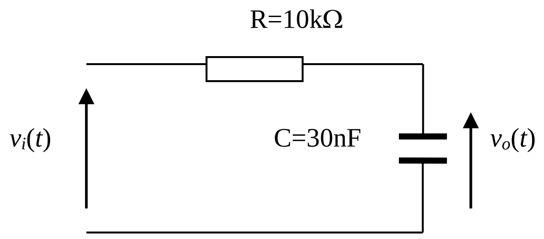

Figure 2: First order RC low-pass filter

Using calculus to analyse the circuit provides:

$$v_i(t) = i(t)R + v_o(t) \tag{1}$$

$$v_i(t) = CR\frac{dv_o(t)}{dt} + v_o(t), \tag{2}$$

since $i(t) = C\frac{dv_o(t)}{dt}$.

Instead of solving this differential equation, we can take the Laplace Transform of the equation to give

$$V_i(s) = RCsV_o(s) + V_o(s), \tag{3}$$

assuming zero initial condition for the capacitance. The transfer function is therefore

$$H(s) = \frac{V_o(s)}{V_i(s)} = \frac{1}{1+RCs}. \tag{4}$$

Exercise

In this exercise we will simulate the RC circuit's frequency response using the Matlab built-in function `freqs`. A prototype function is given in Listing 4 which generates the frequency response of a generic first order system. Modify this listing to plot the frequency response of the circuit in figure 2.

```
function firstorder_sys(n0,n1,d0,d1)
%first-order system in the form:
% H(s) = (n0.s + n1)/(d0.s + d1), where n0,n1,d0 and d1 are constants
% specify values for the constants and then generates frequency response

%specify the constants
N=400;
num=[n0,n1]; den=[d0,d1];

f=logspace(1,5,N);             %generate values from 101 to 105 with %N points
H=freqs(num,den,f);            %evaluate freq. response
mag = abs(H);                     %magnitude response
ph = angle(H);                 %phase response

%plot the response
```

Listing 4

Based on your modified firstorder_sys function:

i) Obtain the –3dB frequency.
ii) Using a sampling frequency of 10 kHz, generate a square wave, *x*(*t*), with a frequency of 100 Hz, using the function `square`. Compute and plot the magnitude spectrum of the square wave, $X(\omega)$ where ω is the frequency in rad/s.
iii) The response of the RC circuit to the square wave generated in part (ii) is given by $Y(\omega) = X(\omega)\times H(j\omega)$. Obtain and plot $Y(\omega)$.

[*Hint: you will have to obtain the magnitude of H(jω) at the frequencies of the magnitude spectrum generated in part (ii)]*

iv) Use the discrete inverse Fourier Transform function, `ifft,` to obtain the time domain output signal *y(t)*.

v) Use the Matlab functions `impulse` and `step` to generate the impulse response and step response respectively.

vi) Now use the Matlab command `lsim` to generate the output *y(t)*.

Demonstrate and explain your work to an instructor/demonstrator.

5 FFT BASED FILTERS

In this section you will see how to use the FFT implement 1-D signal and 2-D image filters.

5.1 1-D signal filter

Go to the EEE260 Teaching Resources page and download the wave file titled "Gigue.wav" remembering where you saved it. Load wave file into Matlab using `[y, Fs] = audioread('Gigue.wav');`.

Now play the wave file using `sound(y, Fs)` - use a music player if necessary. You should have noticed that an unwanted continuous tone can be heard alongside the music. Using the FFT techniques that you developed in the previous sections, determine the frequency of the unwanted tone and then remove it, restoring the piece of music to its former glory.

Demonstrate and explain your work to an instructor/demonstrator.

You may want to experiment by creating filters to amplify or attenuate groups of frequencies to mimic the behaviour of a graphic equaliser.

5.2 2-D image filter

Images can also be manipulated using FFT techniques. Go to the EEE260 Teaching Resources web page and download the GIF image file titled "image1.gif" remembering where you saved it. The Matlab script shown in Listing 5 will filter the image using a brick-wall low-pass filter. The corner frequency of the low-pass filter is controlled via parameter *r*. Type the script into the editor and then run it. Experiment with different corner frequency parameters.

```
im1=imread('image1.gif');       %load the image
imagesc(im1); colormap(gray);   %display the image
f1=fft2(im1,256,256);           %perform 2D fft
f2=fftshift(f1);                %centralise DC and LF components
figure
imagesc(log(1+abs(f2)))         %display fft

%create a mask to lowpass filter
mask=zeros(256,256);
r = 50                          %filter radius
for x=-127:128
    for y=-127:128
        if sqrt((x^2+y^2))<r
            mask(x+127,y+127)=1;
        end
    end
end
figure
imagesc(mask)                   %display the mask

%lowpass filter image
```

```
f3=f2.*mask;
figure
imagesc(log(1+abs(f3)))         %display filtered image fft

f4=ifftshift(f3);               %perform ifft and display
f5=ifft2(f4,256,256);

figure
imagesc(real(f5));colormap(gray)
```

Listing 5

Demonstrate and explain your work to an instructor/demonstrator.

Now modify your program so that it high-pass filters the image. Experiment with different corner frequency parameters.

Demonstrate and explain your work to an instructor/demonstrator.

Finally, investigate what happens when you combine the high-frequency content of one image ("image1.gif") with the low-frequency content of a second image ("image2.gif" which is also available on the EEE260 Teaching Resources web page.

Show your instructor/demonstrator that your program works correctly and discuss your findings.

REFERENCES

[1] Dr C. Bingham, 'Introducing Matlab and Simulink', The University Of Sheffield.
[2] Dr M. P. Foster, 'A concise introduction to Matlab', The University Of Sheffield.