EEE3008 Lab 1: Convolution and Difference Equations

Lab Session:	Tue 15 Oct 2024, 12:00-14:00 (Week 4 - Check personal timetable)
Submission:	Before 16:00 Tue 22 Oct 2024
Feedback:	Within 2 weeks of submission deadline

Introduction

The first part of this lab introduces the concept of convolution. You will learn how convolution corresponds to filtering. The second part of this lab is concerned with solutions to difference equations, which can also be used to specify a digital filter, and stability of filters.

Note: Some parts of this lab build on the “Self-learning: Familiarization with Matlab” worksheet, you are recommended to work through that first.

Instructions: Report all the plots and answer all questions asked in each section. The marks awarded for each part are indicated in square brackets.

Experiments – Section A: Convolution

1. Performing convolution

a) Perform the convolution below using the standard command, ‘**conv**’ and plot the result using ‘**stem**’:

```
» x = [3, 11, 7, 0, -1, 4, 2];
» h = [2, 3, 0, -5, 2, 1];
» y = conv(x,h);
```

Now perform the following convolution, and again plot using ‘**stem**’:

```
» y = conv(h,x);
```

How do the results of the two convolutions compare? **[2 marks]**

b) If the operands **h** and **x** were 6 and 5 samples long respectively, how long is the result y?
Suggest a general rule for calculating the length of **y**, given the lengths of **h** and **x**. **[2 marks]**

c) The standard **conv** command gives us no way to locate any of the time signals. The following Matlab function solves that problem, using parameters **nx** and **nh** to indicate the discrete time indicators (**n**) for **x** and **h**.
Enter the script below in a file named **convolve.m** in your work folder.

```
function [y,ny] = convolve(x,h,nx,nh)
nymin = nx(1)+nh(1);
nymax = nx(length(x)) + nh(length(h));
ny = [nymin:nymax];
y = conv(x,h);
end;
```

Then try the following:

```
» x = [3, 11, 7, 0, -1, 4, 2];
» h = [2, 3, 0, -5, 2, 1];
» nx = [-3 : 3]; nh = [-1 : 4];
» [y, ny] = convolve(x, h, nx, nh);
```

Plot (using **stem**) **y** against **ny**.
Sketch and makes notes on the two different convolution commands. **[2 marks]**

2. Convolution examples

a) Now perform and display the following convolutions (where "x * h" means "x convolved with h"), using the **convolve** function from Part 1. The number underlined indicates the zero index. Plot the results to see what happens. **Note:** You will need to define **nh** and **nx** appropriately.

i) y = x * x, with x = [0, 0, $\underline{1}$, 1, 1].

ii) y = x * h, with x = [0, 0, $\underline{1}$, 1, 1] and h = [$\underline{1}$, 1].

iii) y = x * h, with x = [$\underline{0}$, 0, 1, 1] and h = [$\underline{0.5}$, 0.5]. **[3 marks]**

b) In the following, define xtilde to be 10 repetitions of x (e.g. using **repmat**), and create an appropriate nxtilde.

iv) y = xtilde * h, with x = [$\underline{1}$, 2, 3, 4, 3, 2] and h = [$\underline{0.5}$, 0.5].

v) y = xtilde * h, with x = [$\underline{0}$, 0, 1, 1] and h = [$\underline{0.5}$, 0.5].

Compare the results of the convolution (iii) with the repeated version (v), stating what you notice. Suggest a condition on x and/or h that is required for this special property to hold. **[2 marks]**

3. Convolving with a moving average filter

Create the following signal

```
» nx = [0:100];
» x = sin(2*pi*nx/50) + sin(20*pi*nx/50);
```

Create h as given below: this is a 10-point "moving average" filter.

```
» nh = [0:9]; h=0.1*ones(1,10);
```

View x using **stem**. Now convolve x with h and view the output, y, using **stem**. Try also using **plot**.

Compare the input signal x with the result of convolving with the moving average filter h. Give a qualitative description of what the filter is doing to the signal. **[2 marks]**

Optional Part 4. Viewing result of convolution in the frequency domain (No marks)

Now view x and y from Part 3 in the frequency domain as you did in the "Self-learning: Familiarization with Matlab" worksheet, i.e.

```
» freq = [0:1/512:1];
» fx=fft (x,1024);
» fxmag = abs(fx(1:513));
» plot(freq,fxmag);
```

Plot of x & y and fxmag & fymag. Consider what is happening in the time and frequency domains.

Optional Part 5. Multiplying the frequency domain representation of signals (No marks)

Using **fft()** as above, transform x and h into their frequency domain representations and multiply both vectors. (The Matlab command "**.***" multiplies vectors element-by-element)

```
» fx = fft(x,1024); fh = fft(h,1024);
» mpy = fx.*fh; mpymag = abs(mpy(1:513));
```

Now, plot mpymag against frequency (freq as defined in 4).

```
» plot(freq,mpymag);
```

Compare this plot with that obtained for the output, fy, in Part 4.

Experiments – Section B: Difference Equations

6. Impulse and step response of a filter

This part of the experiment involves writing a Matlab script to examine the impulse and step responses of filters, starting with the following difference equation:

$$y[n] - y[n-1] + 0.9y[n-2] = x[n]$$

and using the 'filter' command.

a) Create a new Matlab script and set up the a and b vectors as follows:

```
» b = [1];
» a = [1, -1, 0.9];
```

Explain how these are obtained from the difference equation. The following code can be used to create the impulse or step responses. Use **impulsesequence.m** and **stepsequence.m** from the "Self-learning: Familiarization with Matlab" worksheet.

Note: When using the previously created impulsesequence and stepsequence functions, ensure the line that plots the graphs is commented out. This can be done by putting a "**%**" symbol at the beginning of the appropriate line. Always ensure that the working directory contains all the functions and scripts required.

```
% The impulse response:
» x = impulsesequence (0, -20, 120);
» n = [-20:120];
» h = filter(b, a, x);
» subplot(2, 1, 1);
» stem(n, h);
» title('Impulse Response'); xlabel('n'); ylabel('h(n)');


% The step response:
» x = stepsequence (0, -20, 120);
» s = filter(b, a, x);
» subplot(2, 1, 2); stem(n, s);
» title('Step Response'); xlabel('n'); ylabel('s(n)');
```

Plot the graphs to check that you understand the results. **[2 marks]**

b) Now check that the system is stable, using these two alternative methods:

i) by calculating the absolute sum of the impulse response using **sum**:

```
» sum (abs(h))
```

ii) by looking at the *poles* of the system, defined by the vector **roots(a),** and checking whether or not they lie within the unit circle, as follows:

```
» abs(roots(a))
```

We will cover more about "poles" and "zeros" in Topic 4 ("Z-Transform"). In the meantime, for this lab, we can assume that a causal system with **b = [1]** is stable if and only if all the roots of **a** lie within the unit circle (i.e. that all elements of **abs(roots(a))** are less than one).

[2 marks]

7. Second order difference equations: responses and poles

In this section, we will explore four second order difference equations. Here, "second order" means that they use past outputs only as far back as y[n-2].

The difference equations are as follows, with all filters using **b=[1]**:

i) **a = [1, -1, -0.9]**

ii) **a = [1, 1, 0.9]**

iii) **a = [1, 1, 1.1]**

iv) **a = [1, -1.9, 1]**

For each filter, examine its impulse response and step response.

The script you wrote for section 6 above can be rewritten as a function to take in the different parameters. Alter the appropriate parts of the script and use the following function definition:

```
function diffequ(a0,a1,a2)
```

[You might need to use **stem(h(1:30))** to look at just the beginning of the resulting output sequence.]

Also, for each filter, find the poles (roots of **a**) and check whether they are inside or outside the unit circle. Consider what the root position implies for stability. Also consider the connection between how damped the impulse response is, and how close to the unit circle the stable poles are.

Finally, look at the z-plane locations of the poles for all the above using the following function call:

```
» zplane(b,a);
```

In your report, for each of the four difference equations (i)-(iv) above, do the following:

(a) write down the appropriate difference equation
(b) plot and the positions of the poles
(c) state whether or not the filter is stable, and why.

[8 marks]

Optional Part 8. Filtering a sound signal (No marks)

Try convolution on the sinusoid from the "Self-learning: Familiarization with Matlab" worksheet using the code below. For this to work you will need the same stored variables that were created in Section 8 of that worksheet.

Note: Use **stem(h)** to view the impulse sequence we are using. Try modifying this.

```
» h = [1  zeros(1,100)  0.5  zeros(1,100)  0.25  zeros(1,100)];
» y = conv(x,h);
» soundsc(y,fs);
```

[Total 25 Marks]