ECE2111 laboratory 1: Signals and Systems in MATLAB

Prepared by: Dr. James Saunderson

Aims

At the end of this lab, students will be able to

1. manipulate and plot complex numbers in MATLAB
2. manipulate and plot discrete-time and continuous-time signals in MATLAB
3. load, play, and manipulate sound in MATLAB
4. implement simple linear systems as functions in MATLAB

Introduction

The aim of this lab is to build some basic skills in using MATLAB to model signals and systems. We will build on these skills in later labs, applying them to solve a variety of interesting problems.

The lab assumes you have some familiarity with MATLAB gained from ENG1060 Computing for Engineers. This lab will also help you diagnose your level of MATLAB skill early in semester, so you have time to improve (and seek help) as the semester proceeds and the labs become more involved and less guided. Please take this seriously and ask for help if necessary! Throughout the semester we want you to gain confidence in your ability to use MATLAB to help you solve engineering problems.

Scheduling

This lab runs over two weeks (weeks two and three). As a guide, we recommend you aim to complete (at least) sections 1, 2, and 3 in week 2 (you are welcome to do more). This will give you a good chance of finishing the lab by the end of week 3.

Prelab

There are three prelab questions throughout this lab document. Before the end of week 1 (for the precise due date see Moodle), read through the lab document, find the prelab questions, and answer them. **Submit your answers to the prelab questions via the Moodle quiz called 'prelab1' on the Moodle page. You are expected to do this individually.**

Results document

You are required to organize your code and outputs in what we will call a "results document". This must be created following the guidelines in the accompanying file "Formatting requirements for lab results document". Submit this by the end of week 3 (for the precise due date see Moodle) via the 'results document' assignment link on Moodle. **Your submission must reflect your own work!**

End-of-lab quiz

There is a timed, end-of-lab quiz that must be completed by the end of week 3 (for the precise due date see Moodle). **Please do not start this quiz until you are ready.** This quiz tests your understanding of the lab material. **It must be completed individually.**

1 Complex numbers in MATLAB

By default, the variables `i` and `j` are both defined to be `sqrt(-1)` in MATLAB. Since this has a lot of potential to cause bugs and confusion, the variables `1i` and `1j` are also defined to be `sqrt(-1)` in MATLAB. In this unit, we recommend (but do not require) you use `1j` to represent the complex number j in MATLAB.

1.1 Activities

Create a script file containing MATLAB commands to carry out each of the following. Remember that if you leave the semicolon off the end of a line of MATLAB code, the line is printed. Make sure you can explain the results you get.

1. Enter `sqrt(-1)` and display the result.
2. Enter `i+j` and `1i + 1j` and display the results.
3. Define $z_1 = 1+j$ by entering `z1 = 1+1j;`. Find the magnitude, phase, real, and imaginary parts of z_1 using `abs()`, `angle()`, `real()`, and `imag()`. Display the results of each.
4. Is the phase in radians or degrees? (No coding required.)
5. Find and display the magnitude of $z_1 + z_2$ where $z_2 = 2e^{j\pi/3}$.
6. Plot the complex number $z_2 = 2e^{j\pi/3}$ in the complex plane. You can do this using the commands `scatter()` and `real()` and `imag()`, for example.

Once you have finished this section:

- copy your code into your results document
- copy the MATLAB output from running your code into your results document
- copy the figure you produced into your results document.
- answer question 4 in your results document

2 Discrete-time signals in MATLAB

In MATLAB we can only explicitly represent finite-duration discrete-time signals. One way to do this is to use two row-vectors of the same length. One of these contains integers and represents time instants. The other contains real or complex numbers and represents the values of the signal at those time instants.[1]

For example, we might represent the discrete-time signal

$$x[n] = \begin{cases} 2 & \text{if } n = -3 \\ 1 & \text{if } n = -2 \\ -1 & \text{if } n = -1 \\ 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 4 & \text{if } n = 2 \\ 3 & \text{if } n = 3 \\ 7 & \text{if } n = 4 \end{cases}$$

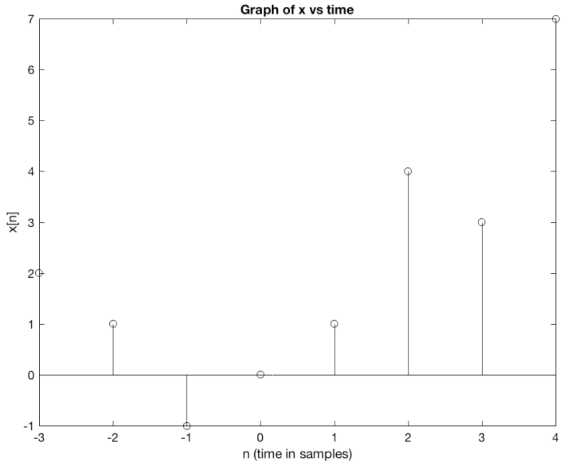
for $-3 \leq n \leq 4$ by the two vectors

```
nx = -3:1:4;
x = [2 1 -1 0 1 4 3 7];
```

(Note that we have used the shorthand `-3:1:4` instead of writing `[-3 -2 -1 0 1 2 3 4]`. The function `linspace()` is also useful for generating vectors of time indices.) We usually use the command `stem()` to plot discrete-time signals. For example the code

```
stem(nx,x);
title('Graph of x vs time');
xlabel('n (time in samples)');
ylabel('x[n]');
```

produces the plot



Sometimes we represent discrete-time signals in MATLAB using only one row-vector, for the values of the signal. For example, we could describe a discrete-time signal by

```
x = [1 5 -2 3 4 -3];
```

[1] This is not necessarily the most elegant way, but it is widely used. A nicer approach might be to define a structure `x` with two fields `x.values` and `x.times` (for example). In this case `x.values` would be a vector storing the signal values and `x.times` would be a vector of the same length storing the corresponding times. This is convenient because it allows us to pass and return a single variable to function that implement systems. It also makes it easy to pass around other information related to the signal.

In this case it is usually *assumed* that the corresponding vector of time indices is

```
nx = [0 1 2 3 4 5];
```

Note that `nx` starts at *zero*. Sometimes, however, we omit the time vector when we don't explicitly need it for the computation we are doing. The message is that if there is no vector of time indices, take care about how to interpret the signal!

2.1 Prelab question 1

Suppose a discrete-time signal y is represented in MATLAB by the vectors

```
y = [0.1 0.3 0 0 1 0 1];
ny = [-3 -2 -1 0 1 2 3];
```

What is the value of the signal y at time $n = 2$, i.e., what is $y[2]$?

2.2 Activities

The following function takes three integers `n0`, `n1`, and `n2` and returns a representation of the signal $x[n] = \delta[n - n_0]$ (where δ is the discrete-time unit impulse) for the time $n_1 \leq n \leq n_2$.

```
function [x, n] = dtimpulse(n0,n1,n2)
% dtimpulse:  returns discrete-time unit impulse function
% centered at n0 over time range n1:n2
% [x, n] = dtimpulse(n0,n1,n2)
% where n0 n1 and n2 are integers with n1 <= n0 <= n2,
% produces signal x[n] = delta[n-n0] for n1<=n<=n2.
n = n1:n2;
x = zeros(1,length(n));
x(n==n0) = 1;
```

The last line of this function may be a bit mysterious. To understand it, look up 'logical indexing' in the MATLAB documentation.

1. Write a MATLAB function `dtstep` that takes three integers `n0`, `n1`, and `n2` and returns a representation of the signal $x[n] = u[n - n_0]$ (where u is the discrete-time unit step function) for the time $n_1 \leq n \leq n_2$. Hint: you only need to change one line of `dtimpulse` above (apart from the header) to achieve this.

2. Use `dtimpulse` and `dtstep` to write a script that creates and plots the following discrete-time signals in separate figure windows:

   (a) $x_0[n] = e^{0.3n}u[n]$ for $-5 \leq n \leq 5$ (Hint: use `.*` to perform element-wise multiplication of row-vectors)

   (b) $x_1[n] = \delta[n + 2] - \delta[n - 4]$ for $-5 \leq n \leq 5$

   (c) $x_2[n] = n(u[n] - u[n - 10]) + 10e^{-0.3(n-10)}(u[n - 10] - u[n - 20])$ for $0 \leq n \leq 20$

   Make sure your plots are appropriately labelled.

Once you have finished this section:

- copy your code (your function `dtstep` and your script for item 2) into your results document
- copy the figures you produced into your results document.

3 Continuous-time signals in MATLAB

We cannot represent arbitrary continuous-time signals perfectly in MATLAB, because to do so we would have to store the signal value at infinitely many times. To get around this, we can discretise the time variable and store two row-vectors: one containing time-stamps (these are real numbers), and one containing the signal values (real or complex) at those time-stamps. For example, if we wanted to represent the signal

$$x(t) = e^{-t}\cos(4\pi t)$$

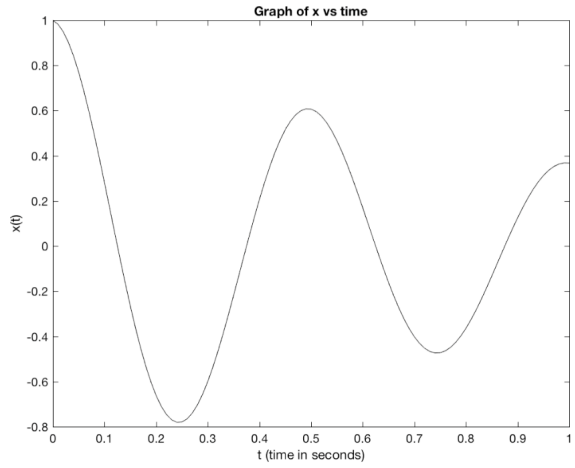
over the interval $0 \leq t \leq 1$, we could do this by forming the two vectors

```
tx = 0:0.01:1;
x = exp(-tx).*cos(4*pi*tx);
```

In this case both vectors have length 101. We usually plot such a signal using the `plot()` command. This does linear interpolation between the sample points, so if we choose enough sample points the plot looks smooth[2]. For example the code

```
plot(tx,x);
title('Graph of x vs time');
xlabel('t (time in seconds)');
ylabel('x(t)');
```

produces the plot



If we had another signal

$$y(t) = e^{-t}\cos(6\pi t)$$

we could represent it over the interval $0 \leq t \leq 1$ by

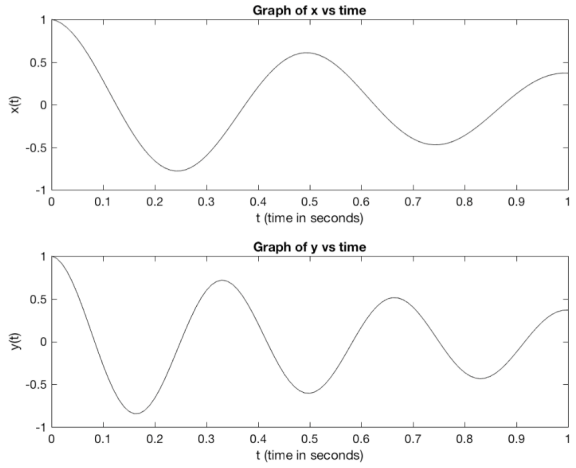
```
ty = 0:0.01:1;
y = exp(-ty).*cos(6*pi*ty);
```

[2]How do we decide how finely to sample the time variable? We will usually be working with fairly well-behaved continuous-time signals, and will usually sample time pretty finely, so that our representation of the signal is very accurate. Later in the unit (in topic 9) we will study more closely what the smallest number of samples is that we can use and still accurately represent a continuous-time signal with samples.

We could plot x and y in the same window, but on separate axes with one above the other, using the `subplot()` function, as shown below:

```
subplot(2,1,1);
plot(tx,x);
title('Graph of x vs time');
xlabel('t (time in seconds)');
ylabel('x(t)');
subplot(2,1,2);
plot(ty,y);
title('Graph of y vs time');
xlabel('t (time in seconds)');
ylabel('y(t)');
```

This produces the plot



3.1 Prelab question 2

Suppose a continuous-time signal y is represented in MATLAB by the vectors

```
ty = -2:0.01:2;
y = (ty).^2;
```

What is the length of the vector `y`?

3.2 Activities

Create a MATLAB script that carries out the following tasks.

1. Create the signal $x_o(t) = te^{-|t|}$ for $t \in [-10, 10]$ using 201 equally spaced samples starting with $t = -10$ and ending with $t = 10$ (hint: use `linspace()`).
2. Using the same vector `t` of time-stamps, create the signals $x_e(t) = |t|e^{-|t|}$ and $x(t) = 0.5*[x_e(t)+x_o(t)]$.
3. Plot all three signals in one window using `subplot()` (read `help subplot` to figure out how to do this).
4. What special property does the signal x_e have? What about the signal x_o?
5. Create a complex-valued continuous-time signal $x_1(t) = e^{j10\pi t}$ with 101 equally spaced time samples starting with time $t = 0$ and ending with time $t = 1$. (Hint: use `linspace()`.)

6. On the same set of axes plot the real and imaginary parts of x_1 over the interval $0 \leq t \leq 1$.

Once you have finished this section:

- copy your code into your results document
- copy the figures you produced into your results document
- answer question 4 in your results document

4 Sound in MATLAB

MATLAB makes it convenient to load, manipulate, and play sound signals. We will use this functionality throughout the labs.

In MATLAB, a sound signal is represented by

- a vector `x` taking values between -1 and 1 and
- a *sampling rate* `Fs` which has units samples/second.

The sound can be played using the command

```
sound(x,Fs);
```

If the sampling rate is omitted (so that the command is just `sound(x);`, it is assumed to be the default value of `Fs = 8192` samples/second. If the vector `x` has values larger than 1 or smaller than -1 it is clipped, so that values larger than one are interpreted (for playback) as 1 and values smaller than -1 are interpreted (for playback) as -1. This is very important to keep in mind when working with sound in MATLAB.

4.1 Prelab question 3

Let $x(t) = \cos(2\pi 440t)$ be a sinusoid with fundamental frequency 440Hz. Let x_s be the discrete-time signal obtained by sampling x with sampling frequency $f_s = 8192$Hz. How many samples are needed to represent 3 seconds of sound (assuming the first sample corresponds to time $t = 0$ and the last to time $t = 3$)?

4.2 Activities

Write a MATLAB script to carry out the following tasks. The command `pause()` may be useful so the script pauses while/after each time you run the `sound()` command.

1. Create a MATLAB vector `xs` that represents the values of the signal x_s from Prelab question 3 for three seconds. Use the command `sound()` to play the sound.

2. Again play the sound, but this time use the *incorrect* sampling frequency of 8000 samples/second. Is the sound higher or lower than before? Why do you think this is happening?

3. Now create a new signal $y_s = 2x_s$ by scaling x_s by a factor of two. Use the command `sound()` (with the correct sampling frequency) to play y_s. What do you hear? Why do you think you hear this? Now use the command `soundsc()` to play y_s. What do you hear? Use `help soundsc` to explain this.

4. There are a couple of predefined sound vectors in MATLAB. As one example, run the command

   ```
   load handel;
   ```

   to load a vector `y` and a sampling rate `Fs` into your workspace. How many samples correspond to the first three seconds of sound (including the sound at time $t = 0$ and time $t = 3$)?

5. Make a new vector `y3` consisting of the values of `y` corresponding to the first three seconds of sound. Play this sound (perhaps you recognise it?) using the command `sound()`.

6. Now we are going to plot the continuous-time signal corresponding to `y3`. We already have the signal values in `y3`, but we do not have a row-vector of time-stamps. Use the `:` operator to construct a vector `ty3` with the same length as `y3` with first entry `0` and last entry `3` and entries spaced by the sampling period. Plot the signal (as a continuous-time signal), being sure to add a title and axis labels to your plot.

Once you have finished this section:

- copy your code into your results document
- copy the figure you produced in item 6 into your results document
- answer the questions in item 2, item 3 and item 4 in your results document

5 Systems in MATLAB

If you reach this point in the first lab session (week 2), you will likely need to refer to the materials in topic 2 to proceed.

Recall that a system is a (mathematical) function that takes a signal as input and produces a signal as output. Because of this, it is very natural to model systems in MATLAB using functions (in the programming sense). For example, the following MATLAB function models a general gain system:

```
function [y, ty] = gainsys(K,x,tx)
% gainsys:  implements gain system
% [y, ty] = gainsys(K,x,tx)
% where x and tx are row vectors of the same length, and K is a scalar,
% produces signal y obtained by scaling x by K and having the same time indices.
if length(x) ~= length(tx)
  error('arguments 2 and 3 should have the same length');
end
if ~isscalar(K)
  error('argument 1 should be a scalar');
end
ty = tx;
y = K.*x;
```

Note that the function has a header consisting of comments describing how it is used. The function then checks that the inputs are valid, before implementing the rule for the function. This is good practice! Note, also, that the gain system we have implemented works just as well for our model of discrete-time signals as for our model of continuous-time signals. Be aware that some other systems may only make sense for discrete-time, or continuous-time signals.

In some cases, the output time vector may not be the same as the input time vector. If this is the case, the header for your function should make this clear, since it might cause confusion!

5.1 Activities

1. Write a MATLAB function `sumsys()` that implements the discrete-time system

$$y = \text{sum}(x_1, x_2) \quad \text{where} \quad y[n] = x_1[n] + x_2[n] \quad \text{for all } n.$$

Your function should take four row vectors `x1` and `nin1` and `x2` and `nin2`. It should return two row vectors `y` and `nout` as output. Make sure you check that the input to the function is valid, by checking that the every entry of the two time vectors agrees, and giving an error otherwise. Does your system also work for continuous-time signals?

2. Check your system `sumsys()` works, by applying it to the signals $\delta[n]$ for $n = 0, 1, 2, 3$ and $u[n]$ for $n = 0, 1, 2, 3$ to find the sum of the two signals for $n = 0, 1, 2, 3$.

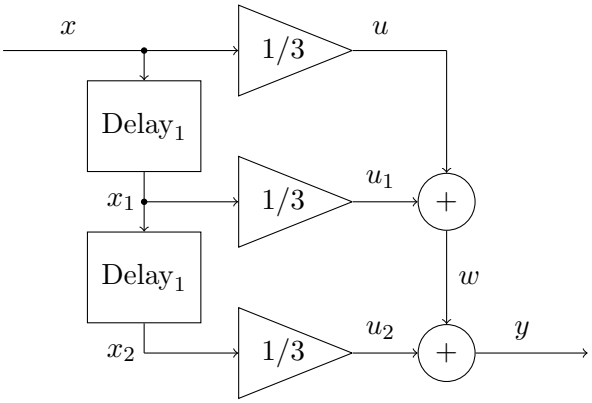
3. Write a MATLAB function `delaysys()` that implements the discrete-time system

$$y = \text{delay}_N(x) \quad \text{where} \quad y[n] = x[n - N] \quad \text{for all integers } n.$$

Your function should take as input an integer (positive or zero) `N` indicating the delay amount, and two row vectors `x` and `nx` for the input signal. Your function should return two row vectors `y` and `ny` corresponding to the values and time-stamps of the output signal. You should implement your function so that `ny` and `nx` are the same[3]. To do this you will need to assume that the input signal x is zero for times outside of the range defined by `nx`.

4. Check your system `delaysys()` works, by applying it to the signal $\delta[n]$ for $n = 0, 1, 2, 3$ with $N = 2$. (Use `dtimpulse()` to construct a representation of the discrete-time unit impulse for $n = 0, 1, 2, 3$.) What happens if $N = 4$?

5. In Section 2.3 of the course notes, the following block diagram is given that describes a three-point averaging system:



Write a MATLAB function `threepointaverage()` that implements this system by making use of the functions `gainsys()` and `delaysys()` and `sumsys()`. A good way to do this would be to label the wires in the diagram with intermediate signals, and use these intermediate signals in your implementation. (Note that this is not necessarily the easiest way to implement this function in MATLAB. We are doing it this way to practice thinking of a block diagram as describing a system as an interconnection of simpler subsystems.)

6. To test your `threepointaverage()` function, we are going to apply it to exchange-rate data[4]. From the ECE2111 Moodle site, download the file `AUDUSD.mat` to your working directory in MATLAB (usually `documents\MATLAB`) and load it into your MATLAB workspace by typing

```
load AUDUSD;
```

You will now see row-vectors `aud` and `taud` in your workspace. The vector `aud` gives the historical value of 1 Australian dollar in US dollars for each day between July 1, 2015 and June 30, 2017.

[3]Note that there are other sensible ways to implement a version of this system, particularly by just changing the time vector and leaving the signal values alone. The downside of doing this is that changing the time vector makes forming interconnections with other systems complicated.

[4]These data are from `http://www.rba.gov.au/statistics/historical-data.html`

(a) Pass the signal `aud` through your three-point averaging system. Call the output signal `audout`.

(b) Plot `aud` and `audout` in the same window, one above the other (use `subplot`). Use `plot()` rather than `stem()` so that the signals are displayed as if they were continuous-time signals (since this is the convention in finance). Give each of your plots a title and label your axes. For each plot set the vertical axis to range from 0.68 to 0.8 using the command `ylim`.

(c) In a few words, how would you describe the signal after applying the system, compared with the input to the system?

Once you have finished this section:

- copy your code (specifically, the function `sumsys()` and `delaysys()` and `threepointaverage()`) into your results document
- copy your figure from item 6(b) into your results document
- answer the questions in item 1 in your results document

Once you have completed the lab tasks and understand them, you are ready for the end-of-lab Moodle quiz. This quiz is closely based on the lab tasks. It must be completed **individually**.You may use MATLAB. Unlike the prelab, you only have **one attempt at each question**.

Assessment

This lab is marked out of 12. Your mark is based on the following:

- **Prelab:** Correct responses to the three prelab questions, submitted via the prelab Moodle quiz (3 marks, 1 per question)
- **Results document:** (4 marks, 3 marks for content and 1 mark for presentation)
  Marks per section: Each of the 5 sections is marked out of 0.6 (3 marks):
  - 0 marks if not attempted
  - 0.4 marks if a reasonable attempt is made, but has clear flaws
  - 0.6 marks if no errors or possibly very minor errors

  Presentation: (1 mark)
  - Results document adheres to the formatting requirements (1 mark)
  - Results document mostly adhere to formatting requirements, but not completely (0.5 mark)
  - Results document rarely adheres to formatting requirements (0 marks)
- **End-of-lab quiz:** Correct responses to the end-of-lab Moodle questions (5 marks, 1 per question).

References and Acknowledgements

Section 1 is adapted from Section 4.4 of [1]. Part of Section 3 is adapted from Section 4.3 of [1]. Part of Section 5 is adapted from Section 3 of [2].

[1] A. Axelrod, J. Bowen, and M. Gupta, *EE235 Lab1, Introduction to MATLAB and scripts*, Available at `http://faculty.washington.edu/tcchen/EE235/Labs/EE235-Lab1.pdf`

[2] A. Axelrod, J. Bowen, and M. Gupta, *EE235 Lab2, Functions in MATLAB and playing sounds*, Available at `http://faculty.washington.edu/tcchen/EE235/Labs/EE235-Lab2.pdf`