SYDE 252

Matlab® Assignment 1

(Due date: Nov. 8, Sunday, 5 pm)

Note:

- The underlined text are the tasks to be done in Matlab (either a script, or a function, or a part of the function).
- From Problem 1 to Problem 3, submission your results in the forms of Matlab .m files (either scripts or functions). Clearly name the files, i.e. Problem1_a.m
- For Problem 4, include the finished `AssignmentOne_SYDE252.m` file.
- You can also include a text document (Word or PDF) with explanations to your work, if you believe it is necessary, for example problem 2(b);
- Zip all files, and upload the zip file to the Dropbox in LEARN;
- Problem 4 requires a Windows computer that has a working soundcard, speaker and microphone, and it should have 32-bit Matlab with Data Acquisition Toolbox installed. If you have difficulty in find such a computer, please let me as soon as possible!

Problem 1: Numeric approximate of continuous-time convolution

To calculate continuous-time convolution with Matlab, we in fact are using numeric approximation. Specifically:

$$z(t) = x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau$$

is approximated by:

$$\widehat{z(t)} = \sum_{k=-\infty}^{+\infty} x(k\Delta)y(t-k\Delta)\Delta$$

where Δ is a small time interval. When $\Delta \to 0$, the summation becomes the integral. In Matlab, the time between two data points of a Matlab vector can be used to represent Δ.

(**a**) Given two Matlab vectors, **`x`** and **`y`**, each representing an equally sampled version of continuous time signal, write a script that calculates the convolution of x and y with you own code (do not use Matlab built-in function: `conv`)
For example (see Oppenheim Example 2.7, page 99 -101):

```
>> x = ones(1,101);
>> y = 0:0.01:2;
```

Represents the following two signals, sampled every 10 ms:

$$x(t) = \begin{cases} 1, & 0 < t < 1s \\ 0, & \text{otherwise} \end{cases}$$

$$y(t) = \begin{cases} t, & 0 < t < 2s \\ 0, & \text{otherwise} \end{cases}$$

You can try different signals to validate your script.
(**b**) Change your script into a function (something like `z = MyConv(x,y)`)

Problem 2: Numeric approximation of Fourier series for continuous-time periodic signals

Similarly, Fourier series for continuous-time signals can be numerically approximated. For example (see Oppenheim Example 3.2):

```
>> t = 0:0.1:3;
>> x = 1+cos(2*pi*t)/4 + cos(2*pi*t*2)/2 + cos(2*pi*t*3)/3;
>> figure, plot(t,x);
```

x is an equally sampled continuous time signal $x(t)$ between 0:3 seconds:

$$x(t) = 1 + \cos(2\pi t)/4 + \cos(2\pi t 2)/2 + \cos(2\pi t 3)/3$$

and **t** is its corresponding time index. The signal is sampled every 100 ms.
(**a**) Write a script that calculates the Fourier series coefficient, a_k, of a periodic signal $x(t)$. (no built-in Matlab Fourier functions should be used). Test various periodic signals with your script.
(**b**) With the example above, the coefficients, ideally, should be real. However, the obtained coefficients using Matlab will be imaginary, but with very small imaginary parts (magnitude likely smaller than 10^{-15}). This is due to the numeric precision of Matlab, which is acceptable. And it is sufficient to use the real part:

```
>> a = real(a);
```

where `a` is the complex coefficient.
However, very likely the real part of the coefficient will seem a bit odd. For example, you might get $a_1 = a_{-1} = 0.1319$ or $a_0 = 1.0069$, in which case the actual coefficient should be 0.125 and 1, respectively. This large error is not due to numeric precision. It is not due to information loss in the sampling process neither, as we would learn later in the course that 100 ms sampling interval (10 Hz) is more than enough to perfectly reconstruct a sinusoid signal with highest frequency of 3 Hz. Explain why the real part is not as accurate as it should be and make changes to correct this problem (Hint: the smaller the sampling interval, the smaller such error would be. Try `t = 0:0.01:3`). If you don't find this problem, i.e. you have $a_1 = a_{-1} = 0.1250$ and $a_0 = 1.000$, then congrats and Move on!
(**c**) Change the script into a function (e.g. `ak = MyFSAnalysis(x,t,k,w0)`);
(**d**) Write a script that reconstruct $x(t)$ with the obtained Fourier series coefficients, a_k. (no built-in Matlab Fourier functions should be used);
(**e**) Change the above script into a function;

Problem 3: Numeric approximation of Fourier transform of continuous-time signals

With the experiences above, write two functions that perform Fourier Transform and inverse Fourier Transform of continuous time signals, such as:

```
Xw = MyFT(Xt, t, w);
```

```
Xt = MyiFT(Wt, w, t);
```
Again, no built-in Matlab Fourier functions should be used.

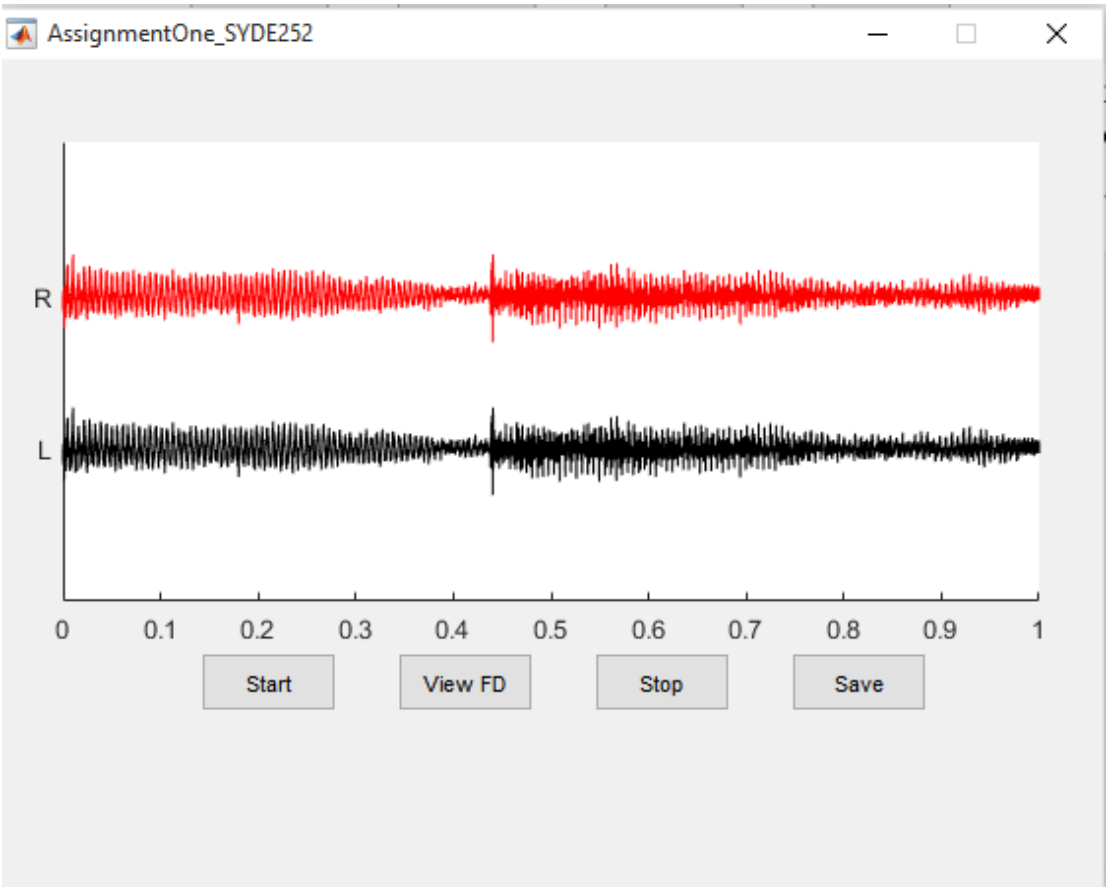
Problem 4: Fourier transform of continuous-time signals, or your favorite music

Get the Matlab program from LEARN:
`AssignmentOne_SYDE252.m`
`AssignmentOne_SYDE252.fig`

This is a small program that would run in 32-bit version of Matlab (It uses the Data Acquisition Toolbox's `winsound` object, which is only available with 32 version). You can install the 32-bit version on a 64-bit Windows system.

Assuming the computer has a working soundcard, you can start the program by typing in the command window of Matlab:
```
>> AssignmentOne_SYDE252;
```
The following window will appear:

Start playing your favorite music with the speaker of the computer, assuming the computer has a working mic, now click 'Start' button, you will see something like this:



The two lines are the time domain signals of the two channels of the audio signals acquired by the microphone of the computer (i.e. the music you are playing).

Click 'Stop' button, the acquisition will stop; click 'Save' button will save the acquired music in a file. 'View FD' button is intended to switch the data view between frequency-domain and time-domain. However, it is not fully functional: only time domain view is available.

Open `AssignmentOne_SYDE252.m`, scroll down to Line 228. Insert necessary code here so that the Frequency domain data view will be functional. You should use your Fourier Transform function you developed in Problem 3 of this assignment.

A few hints:
1. When you click 'Start' button, the function `updateAxes` (from Line 191 on) will be executed every 300 ms. The variable `acqData` holds the data to be plotted. Between Line 218 and 227, is where the time-domain plotting takes place.

2. In case you find the program is not responsive enough, two changes might help:
a) Increasing the timer period of the data acquisition object, i.e. reducing the refresh rate of the data view:
@Line 63: `handles.h_soundCard.TimerPeriod = 0.3;`
It is currently set at 300 ms (3 frames per second).

b) Reducing the time span of the data view:
@Line 65: `handles.dataTimeSpan = 1;`
It is currently set at 1 s.