UCLA
Dept. of Electrical and Computer Engineering
ECE 114, Fall 2019

Computer Assignment 1: Introduction to Frequency Analysis

Due: October 7, 2019

ECE 114 Computer Assignment Report Format: When completing computer assignment reports, we recommend that you start each with an abstract, consisting of one or two paragraphs, which provides a high level summary of the assignment. Proceed by answering the questions listed in the assignment and include any code that you write. Please work in groups of 2 students and submit 1 report per group.

Introduction: The goal of this assignment is to serve as an introduction to frequency analysis using MATLAB. Specifically, this assignment deals with implementing and applying the Discrete Fourier Transform (DFT) to speech signals.
Download the computer assignment 1 supplementary files on the CCLE website (in the folder **ca1_files** under **Course Materials** section). Run the first program by typing "one" at the MATLAB prompt. Two figures should be displayed. The first figure will include the entire sentence "she had your dark suit in greasy wash water all year" in the top panel, and the word "wash" in the bottom panel. The second figure will include the word segment "wa" in the top panel, and the phoneme /a/ in the bottom panel, both from the word "wash."
Note that variables have been created in the current workspace containing the time waveforms of the speech signals corresponding to the entire sentence ("data"), the word "wash", the word segment "wa", the phoneme /a/, and the phoneme /ʃ/ (as in "sh"). You can listen to the speech waveforms using the following command:

```
soundsc(xxx,8000);
```

where "xxx" is the name of the variable and 8000 is the sampling frequency in Hz.
You can plot the speech waveforms using the following command:

```
figure; plot(xxx);
```

Plot the phoneme /a/ and the phoneme /ʃ/ in separate figures. Include each in your write-up.

Frequency Analysis:
a) Implementing a Discrete Fourier Transform (DFT):
The N-point DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \text{ for } 0 \leq k \leq N-1$$

The output of an N-point DFT is a complex sequence. When analyzing speech, however, we are usually interested only in the magnitude of the spectral representation.
The file `dft.m` contains a shell for determining the magnitude of the Discrete Fourier Transform of an input sequence. Most of the routine is written. However, a couple of lines are left unfinished (marked by "???"'s). Please finish these lines according to the definition of the DFT (equation above).

The script `two.m` performs spectral analysis on the phoneme /a/ using the `dft.m` function previously written. To run this script, type "two" at the MATLAB prompt. The script will display two figures. The first figure will include the magnitude spectrum of the input speech signal on the frequency axis $[0, 2\pi)$ on the upper panel. The lower will show the magnitude spectrum on the frequency axis $[0, \pi)$. The second figure will include two versions of the log-scale magnitude spectrum on the $[0, \pi)$ frequency scale: the top panel computed by the `dft.m` function, the bottom panel computed by the FFT. If the `dft.m` function was written correctly, the two versions should match.

Questions:

1. Include separate figures for the phoneme /a/ and the phoneme /ʃ/.
2. Include the code written for the `dft.m` function.
3. From the magnitude spectrum plot on the frequency axis $[0, 2\pi)$, what do you observe? Give reasons about your observations.
4. The function `dft.m` determines the N-point magnitude DFT. How large is N in the case of the input /a/ signal? What is the resulting spacing between adjacent frequency bins?
5. "Brute force" computation of the DFT requires order of N^2 complex multiplies, where N is the length of the input signal. The computational complexity of the radix-2 FFT is order $N \log_2(N)$. (In both the DFT and FFT, various reductions can be realized by exploiting symmetries when possible). How much more efficient is the radix-2 FFT than a N=256 point DFT? First express this as a ratio of multiplications required for each algorithm. Then notice that the file two.m prints out the computation time for both our DFT implementation and the FFT. Give the ratio of computation times. This number will vary based on computer hardware, but you should notice that the FFT is much faster.

b) The Effects of Oversampling:

To represent an arbitrary N-point time sequence requires N points in frequency. However, we can improve the spacing between adjacent frequency bins by concatenating zeros to the end of the input sequence, and thus taking the transform of a longer sequence. This can be seen by running the script `three.m`. Two figures will be displayed. The first will include the time waveform of the /a/ phoneme, along with the log-scale magnitude spectrum. The second figure includes the log-scale magnitude spectrum after $\times 2$ zero-padding, and the log-scale magnitude spectrum after $\times 8$ zero-padding.

Questions:

1. Briefly describe the changes in the log-scale magnitude spectra with increased oversampling.
2. If we add an infinite number of zeros to the end of the input sequence, in the limit, what happens to the spacing between adjacent frequency bins?

c) Linear vs Circular Convolution:

In Problem Set 1, we ask you to compute the convolution y[n]=x[n] * x[n] where x[n]=[$\underline{4}$,-1,2]. Let's check our answer her using MATLAB. Use the conv() command to compute the linear convolution of x[n] * x[n]. The inputs to the conv() command are the two signals to be convolved entered as row vectors. It is up to the programmer to determine the position of the origin before and after the convolution. Keep in mind that if that if a signal x[n] is nonzero from x[a] to x[b] and a signal h[n] is nonzero from h[c] to h[d] then the signal y[n] = x[n] * h[n] is nonzero from y[a+c] to y[b+d]. Use the command:

```
conv(x,x)
```

to calculate the linear convolution. Record the output.

Now let's also compute the circular convolution using the cconv() command for comparison. Use the command:

```
cconv(x,x,3)
```

and record the output. The cconv() command is called with a command of the form cconv(x,h,N) where x and h are the signals to be circularly convolved entered as row vectors and N is the size of the circular convolution. Here, we choose a circular convolution the same size as our DFT size (the length of x[n]). The circular convolution here produces a time aliased version of the linear convolution. What is the minimum size N of the circular convolution that is needed to produce the same result as the linear convolution above?
It is often advantageous to calculate the convolution in the frequency domain rather than in the time domain. Let's compute the convolution by taking the Inverse Fourier Transform of the products of the Fourier transform of the two signals with the command:

```
y=ifft(fft(x).*fft(x))
```

Does this produce the result of the circular convolution or the linear convolution? Why?