Homework # 2

Due Date: Tuesday, September 17, 2019 by 11:59PM
Total Points: 100

The purpose of this assignment is to gain familiarity with performing signal processing (e.g. Short-time Fourier transform) in a Python Environment. The questions below will reinforce the concepts that we discussed during lecture. **For the purposes of this assignment, please work individually**. However, feel free to ask basic questions to the instructor and classmates, but complete the assignment on your own. Feel free to search for information (inputs and outputs) about key Python functions, using the internet. From Canvas, go to Files and download all content from the HW#2 folder.

You must submit a typed report that lists the questions, shows plots, discusses results, and answers all questions in a clear and concise manner. Your report should be submitted as a single pdf document through Canvas. Also be sure to submit all code in a separate zip (or tar) file through Canvas. Be sure to include a README that explains each major script/function. Answer all questions within the report. Feel free to create separate functions as needed. **Be sure to comment your code and to submit all files to Canvas**.

All assignments must be submitted on time to receive credit. No late work will be accepted, unless you have a prior arrangement with the instructor.

Question 1. [30 POINTS]

Short-Time Fourier Transform (STFT): Fourier View. A time-frequency domain signal can be computed from the analysis function of the Short-Time Fourier Transform.

1. Create your own function that when given a time-domain signal, computes its STFT. Be sure to implement this function on your own using only basic Python functions (e.g. exp()). This function should be written so that it can handle generic parameter values (e.g. window type, window length, hopsize (L), etc.). You can use Python's built-in windowing functions (e.g. *numpy.hamming*()).

2. Use your function to compute the STFT of the *speech2.wav* audio file. Compute the STFT using each of the parameter configurations shown below. For each configuration, plot the power spectrogram (in dB). **Be sure to correctly label the x- (e.g. in seconds) and y- (e.g. frequency) axis. Also include a title for each plot.** This will generate spectrograms similar to the ones shown in class.

   - N=256, window type = hamming, window length = 10ms, L = 5ms
   - N=256, window type = hamming, window length = 20ms, L = 10ms
   - N=512, window type = hamming, window length = 40ms, L = 20ms
   - N=1024, window type = hamming, window length = 80ms, L = 40ms
   - N=2048, window type = hamming, window length = 160ms, L = 80ms

3. Discuss the differences and similarities between the power spectrograms from the different configurations.

Question 2. [25 POINTS]

Overlap-Add (OLA) Synthesis: Overlap-Add synthesis can be used to convert a time-frequency domain signal back to its time-domain representation.

1. Create your own OLA function that when given a time-frequency domain signal, it outputs its time-domain version. The function should take as inputs the parameters that were used to generate the STFT (e.g. window type, window length, N, etc.), along with the STFT. Be sure to implement this function on your own using only basic Python functions (e.g. *exp*(), *numpy.hamming*(), etc.).

2. Using the STFTs that were generated in the prior problem, use your OLA function to generate time-domain signals for each parameter configuration. Plot these time-domain signals, along with the original time-domain signal. Discuss any differences or similarities that you notice. Be sure to label all axes.

Question 3. [10 POINTS]

Signal Power and Energy: This problem provides practical experience with computing the power and energy of a speech (e.g. speech.wav) and a noise (e.g. noise.wav) signal.

1. In a single figure, but within separate plots, plot the speech and noise signals as **functions of time (seconds)**. Be sure to correctly label all axes. See *matplotlib.pyplot* for help with plotting.

2. In a separate figure, plot both the instantaneous power of the speech and noise signals. Be sure to correctly label all axes and provide a legend, if necessary.

3. Compute the total energy of each of the signals, and display these values to the screen.

4. In class, it was shown that the sound level between to signals is computed as a log-ratio between the instantaneous power (or intensity) of the two signals. In practice, however, it is often desired to compute the log-ratio of the total energy of the two signals. For this problem, compute the log-ratio (e.g. $10\log_{10}()$) of the two signals. Use the energy of the speech in the numerator and that of the noise in the denominator.

   (a) Display this sound level to the screen.

   (b) From the ratio, what can be said about the two signals?

Question 4. [15 POINTS]

Parseval's Theorem: Parseval's Theorem states that energy can be calculated in the time or frequency domains, and that the calculated energies are equal.

$$\sum_{n=0}^{N-1} x^2[n] = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2 \tag{1}$$

For this problem, you will test this theorem to determine if it is true in practice.

1. Write your own code to compute the total energy of the speech signal, *speech2.wav*, in the time domain. Display the energy value to the screen.

2. Write a function that computes the DFT of a time-domain signal. Use this function to compute the DFT of the above speech signal. In a single figure, use the *subplot*() function to plot the time domain and magnitude response of the signal.

3. Compute the energy of the signal in the frequency domain. Compare the energy computed in the time domain, to the energy that is computed in the frequency domain. Does Parseval's Theorem hold in practice?

Question 5. [20 POINTS]

In class we discuss how an analysis window is needed when computing the short-time Fourier transform (STFT). We also mentioned how the selected analysis window and it's length impacts time and frequency resolution. This problem we give you practical experience to help better understand these concepts. Perform the following steps:

1. Use Python's *hamming* window function (e.g. *numpy.hamming*), and generate hamming windows with the following lengths: 25 ms, 50 ms, 100 ms, 200 ms, and 400 ms. Assume a sampling rate of 10 kHz is used.

2. Use Python's *fft* function to compute the discrete Fourier transform (DFT) of each window. Plot the power spectrum (dB) of each result. Be sure to appropriately label the x- and y-axis, and provide a title. Also, **Describe the differences and similarities of each power spectrum.**

3. Read in the speech data that is contained in the 'speech2.wav' file. Extract the first YY ms of this signal, where YY is 25, 50, 100, 200, and 400 (as above). Compute the DFT of each of these speech segments. Plot the power spectrum (dB) of each result. Be sure to appropriately label the x- and y-axis, and provide a title.

4. Multiply (pointwise) the YY ms power spectrum of the windowing function with the YY ms power spectrum of the speech segment. Do this for each value of YY and plot the power spectrum (dB) of each result. Be sure to appropriately label the x- and y-axis, and provide a title. **Describe the differences and similarities of each power spectrum and explain why they are different. Also discuss how and why the windowed-speech DFT compares to the non-windowed (original) speech DFT.**