SGN-14007 Introduction to Audio Processing

Exercise 2
Week 45

In this exercise, you will familiarize yourself with the implementation of spectrogram and its use for different types of audio signals (speech, music, synthetic sinusoids). Each problem is worth 1 point. There is a bonus problem worth 0.5 points, which is optional. **The submission should consist of a report of your observations and the python code**. Alternatively, both can be combined in a Jupyter notebook as well. Refer to the general notes (given in the end) before starting the exercise.

Problem 1: Implement a function to compute power spectrogram (use audio 1) (1 point)

Recall the loop/function you implemented in the last exercise where DFT was calculated for successive 100 ms length frames of an audio signal. Now add the following functionalities to the loop/function:

a) Multiply each signal frame with a windowing function (use Hamming). For any such frame observe the difference in DFT when you introduce windowing as compared to no widowing (can be thought of as rectangular windowing, you already did the latter in Ex1). Use FFT length equal to the window size while computing DFT.

b) Until now there was no overlap between consecutive signal frames. Now introduce frame overlap in the analysis, i.e., the next analysis frame starts from the middle of the current analysis frame. This amounts to 50 % overlap.

c) Now collect the power spectrum (i.e. square of the magnitude DFT) into a matrix such that each column of the matrix contains the power spectrum of the n^{th} frame, n =0, 1, 2, ….
Plot this matrix as an image (Hint *matplotlib.pyplot.imshow*). Now take the logarithm of the matrix and plot. What differences do you observe?

Problem 2: Analyse the effect of window size on the spectrum of different types of audio signals. (1 point)

a) Calculate spectrogram with a library function(*e.g.,librosa.stft,* use FFT length equal to the window size) and observe if there are any differences from your implementation. Report the same.

b) Now run your implementation in Problem 1 with different window sizes for different types of signals, i.e., music (*audio 1*), speech (*audio 2*), and synthetic sinusoids (*from exercise 1)*. For example, use 16ms, 32 ms, 64 ms, and 128 ms.

c) Report what do you observe when you increase the window size. What analysis window length you think is best for each of the signal types in question 1? (Hint: refer Lecture 2).

Bonus problem: Implement overlap-add reconstruction. (0.5 point)

For simplicity use a generated sinusoidal signal (for example any of the sinusoid in Exercise 1), it will allow you to see any discontinuities in the reconstructed signal
a) Change the window function to a symmetrical one, Do the analysis as was done in Problem 1, collect all frames into a matrix.
b) Now for synthesis apply inverse DFT (IDFT) in a loop with the same window size, overlap and window type as used in analysis. Overlap and add frames to obtain the original signal back. Plot and compare your sinusoidal signal with the reconstructed signal for first few frames. Report your observations.

General notes:

1) Only submissions in python will be graded from this exercise. It is advisable to use python framework for audio signal processing not only due to its open-source nature and ready availability of useful libraries but also because it has become a popular choice for prototyping audio focussed machine learning methods in recent years.

2) If you are using TUNI systems to do the exercise, it is advisable to have your own python installation so that you can install any additional libraries e.g., *librosa, sounddevice*. Download Anaconda (*https://www.anaconda.com/distribution/*) to your own directory and install. Any additional libraries can then be installed using conda install, e.g. for *librosa* and *sounddevice*,

conda install -c conda-forge librosa
conda install -c conda-forge python-sounddevice
(-c flag here refers to the channel the library is being downloaded from)

Please ask the teaching assistant for help if you need assistance with this.

3) There may be implementation differences between libraries when it comes to reading a .wav file. For example, *librosa* supports floating-point values and rescales the input audio to [-1, 1] while *scipy* does not do that. To avoid confusion it is advisable to use *librosa* as the main library for audio manipulation.

4) If you are using *librosa* to read audio, ensure that you put sampling rate to *None* to avoid resampling the audio to 22050 Hz which is the default behaviour in *librosa,* i.e.,

librosa .load('audio.wav', sr=None)