Computing the spectrogram

You can read a wav file in python using different packages. Following is an example using Scipy package.

```
from scipy.io import wavfile
from scipy import signal

samplerate, data = wavfile.read('filename')
data = signal.resample(data,int((len(data)*16000)/samplerate))
```

Here, we resample the signal to a standard sampling rate (16000 Hz) for convenience. Edit the "**load_data"** function to load the notes and the music.

Next, we can compute the complex short-time Fourier transform of the signal, and the magnitude spectrogram from it. "**stft**" function computes the complex spectrogram of a signal. The complete function is given for your convenience. Following is an example of how to use it-

```
import numpy as np

spectrum, sphase = stft(data,2048,256,0,np.hanning(2048)) # stft, phase
music = np.abs(spectrum) # magnitude
```

Here we use 2048 sample windows, which correspond to 64ms analysis windows. Adjacent frames are shifted by 256 samples, so we get 64 frames/second of signal. This will result in a 1025-dimensional (rows) spectrogram, with 64 frames(columns) per second of signal. Note that we are also storing the phase of the original signal. We will need it later for reconstructing the signal. We explain how to do this later.

You can compute the spectra for the notes as follows. The following script reads the directory and computes spectra for all notes in it.

```
import os
from scipy.io import wavfile
from scipy import signal
import numpy as np

notesfolder = '/content/drive/MyDrive/data/notes_15/'
listname = [f for f in os.listdir(notesfolder) if f.endswith(".wav")]
notes = []
for k in range(len(listname)):
   samplerate, data = wavfile.read(os.path.join(notesfolder, listname[k]))
   data = signal.resample(data[:,0],int((len(data)*16000)/samplerate))
   spectrum, sphase = stft(data,2048,256,0,np.hanning(2048)) # stft, phase

   # Find the central frame
```

```
        middle = int(np.ceil(spectrum.shape[1]/2))
        note = np.abs(spectrum[:, middle])

        # Clean up everything more than 40db below the peak
        note[np.argwhere(note < (note.max()/100))] = 0
        note = note/np.sqrt(np.sum(note**2)); #normalize the note to unit length

        notes.append(note)

    notes = np.array(notes).T
```

The “**notes**” matrix will have as many columns as there are notes (15 in our data). Each column represents a note. The notes will each be represented by a 1025-dimensional column vector.

Reconstructing a signal from a spectrogram

The recordings of the complete music can be read just as you read the notes. To convert it to a spectrogram, do the following. Let “**reconstructedmagnitude**” be the reconstructed magnitude spectrogram from which you want to compute a signal. In our homework we get many variants of this. To recover the signal, we will use the “**sphase**” we computed earlier from the original signal-

```
    reconstructedsignal =
    stft(reconstructedmagnitude*sphase,2048,256,0,np.hanning(2048))
```