lab3

February 12, 2023

Lab 3 Submission for jorgejc2 and ericji3

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read, write
from numpy.fft import fft, ifft
```

[12]:
```
# FRAME_SIZE = 1024
# ZP_FACTOR = 2
# FFT_SIZE = FRAME_SIZE * ZP_FACTOR

FRAME_SIZE = 256
ZP_FACTOR = 2
FFT_SIZE = FRAME_SIZE * ZP_FACTOR
```

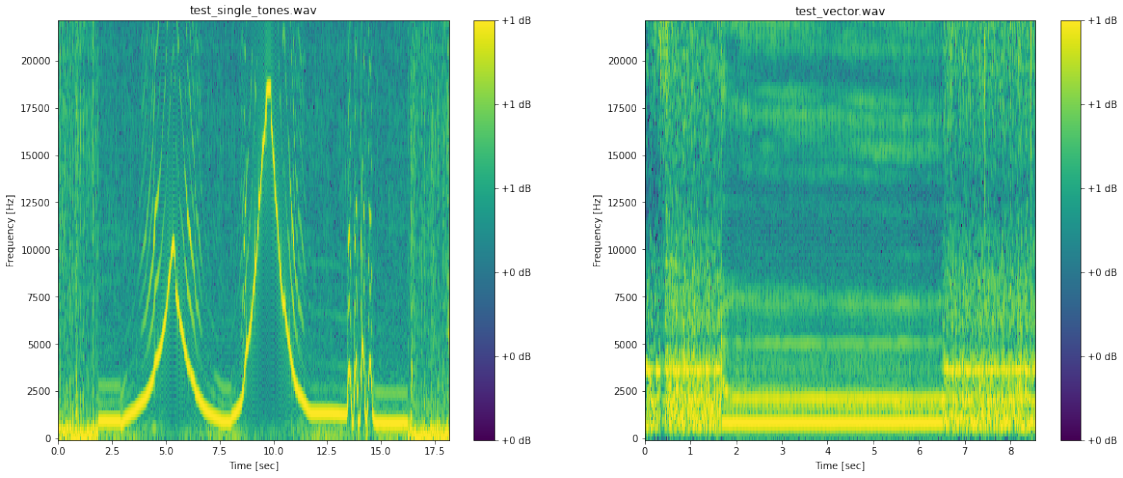
[13]:
```
################### YOUR CODE HERE ######################
def hammingWindow(N):
    """
    Description: Creates a Hamming window of length N
    """
    window = np.zeros(N)

    for n in range(N):
        window[n] = 0.54 - 0.46 * np.cos((2*np.pi*n)/(N - 1))

    return window

def ece420ProcessFrame(frame, Fs):
    # print("Len frame: {}, FRAME_SIZE: {}".format(len(frame), FRAME_SIZE))
    curFft = np.zeros(FFT_SIZE)

    # fill in curFft with samples
    for i in range(len(frame)):
        curFft[i] = frame[i]

    # apply hamming window (wasted computations if window is applied on zeros)
    curFft[:len(frame)] = curFft[:len(frame)] * hammingWindow(len(frame))

    curFft = (np.log10(abs(np.fft.fft(curFft))**2))
```

```
    curFft = curFft / np.max(curFft)


    return curFft[:FRAME_SIZE]
```

[14]:
```
################## GIVEN CODE BELOW ######################

Fs_1, data_1 = read('test_single_tones.wav')
Fs_2, data_2 = read('test_vector.wav')

numFrames_1 = int(len(data_1) / FRAME_SIZE)
bmp_1 = np.zeros((numFrames_1, FRAME_SIZE))
total_t1 = (1/Fs_2) * len(data_1) # total time of file in seconds

numFrames_2 = int(len(data_2) / FRAME_SIZE)
bmp_2 = np.zeros((numFrames_2, FRAME_SIZE))
total_t2 = (1/Fs_2) * len(data_2) # total time of file in seconds


for i in range(numFrames_1):
    frame = data_1[i * FRAME_SIZE : (i + 1) * FRAME_SIZE]
    curFft = ece420ProcessFrame(frame, Fs_1)
    bmp_1[i, :] = curFft

bmp_1 = bmp_1.T

for i in range(numFrames_2):
    frame = data_2[i * FRAME_SIZE : (i + 1) * FRAME_SIZE]
    curFft = ece420ProcessFrame(frame, Fs_2)
    bmp_2[i, :] = curFft

bmp_2 = bmp_2.T

len_t1 = bmp_1.shape[1] # number of column is number of time frames
len_f1 = bmp_1.shape[0]//2 # number of rows is number of real frequency bins
t1 = np.linspace(0, total_t1, len_t1)
f1 = np.linspace(0, Fs_1/2, len_f1)

len_t2 = bmp_2.shape[1] # number of column is number of time frames
len_f2 = bmp_2.shape[0]//2 # number of rows is number of real frequency bins
t2 = np.linspace(0, total_t2, len_t2)
f2 = np.linspace(0, Fs_2/2, len_f2)

plt.figure(figsize=(20,8))
plt.subplot(121)
plt.pcolormesh(t1, f1, bmp_1[:len_f1,:], vmin=0, vmax=1)
```

```
plt.title("test_single_tones.wav")
plt.axis('tight')
plt.xlabel("Time [sec]")
plt.ylabel("Frequency [Hz]")
plt.colorbar(format="%+2.f dB")

plt.subplot(122)
plt.pcolormesh(t2, f2, bmp_2[:len_f2,:], vmin=0, vmax=1)
plt.title("test_vector.wav")
plt.axis('tight')
plt.xlabel("Time [sec]")
plt.ylabel("Frequency [Hz]")
plt.colorbar(format="%+2.f dB")
plt.show()
```

```
C:\Users\flaco\AppData\Local\Temp\ipykernel_888\1816986671.py:24:
RuntimeWarning: divide by zero encountered in log10
  curFft = (np.log10(abs(np.fft.fft(curFft))**2))
C:\Users\flaco\AppData\Local\Temp\ipykernel_888\1816986671.py:26:
RuntimeWarning: invalid value encountered in divide
  curFft = curFft / np.max(curFft)
```



Why would you use a STFT over a single-snapshot FFT? The stft creates spectograms for content whose frequencies vary throughout time (music being a perfect example). For a sound file, the FFT can mark all the frequencies, and at a good amount of frequency resolution since the sound file may be large, but the stft will give some time precision indicating at which time periods some frequency content appears. The standard FFT has no time precision.

What allows us to ignore the second half of our FFT output? Recall the Conjugate Symmetry property of the Fourier Transform? We can ignore the second half of our FFT because for real valued signals because of the Conjugate Symmetry property. This property results

in the second half of the FFT results being symmetric and equal in magnitude as the first half of the results. This makes sense since when we have a sampling rate of Fs, by Nyquist, we are looking for frequencies from 0 to Fs/2. We cannot look for frequencies from -Fs/2 to 0 since we are dealing with real valued signals. If we were instead taking IQ samples, this would no longer be the case because we would then have complex samples that can no longer be symmetric. That would be the scenario when working with Software Defined Radios and radio frequencies, not audio processing.

Given a buffer size N and a sampling rate Fs, how much time do you have to complete your processing before the next buffer comes in? You have $\frac{N}{Fs}$ seconds to complete your process.