ITU Computer and Informatics Faculty
BLG 354E Signals and Systems for Computer Engineering -2020 Spring Homework-5

O. Kürşat Karayılan
150140011

I did this homework in Jupiter notebook but I will share both python file and notebook file with you.
Notebook:
https://drive.google.com/file/d/1XN4xYbWi9kwFieEHv4mbgKTHgsBVA_sA/view?usp=sharing

```
In [238]: import scipy.io.wavfile as wavfile
          import scipy
          from scipy import signal as sg
          import scipy.fftpack as fftpk
          import numpy as np
          from matplotlib import pyplot as plt
```

```
In [239]: s_rate, signalAfrica = wavfile.read('Africa.wav')
          s_rate, signalWinner = wavfile.read('WinnerTakesAll.wav')
```

```
In [240]: def ffthelper(signal):
              #abs since it contains complex numbers
              FFT = FFT = abs(scipy.fft(signal))
              #normalization
              FFT = FFT * (1/len(FFT))
              freqs = fftpk.fftfreq(len(FFT), (1.0/s_rate))
              return FFT, freqs
```

In first block I imported necessary libraries. Then on second block I read wav files. Ffthelper fuction takes signal as an input. It computes its fft with the help of scipy.fft function. Then I normalize it. After that it computes frequency array. Finally returns FFT and freqs arrays.

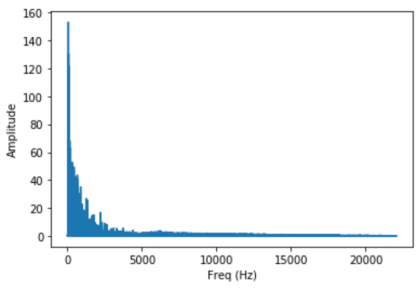
```
In [241]: fftAfrica, freqsAfrica = ffthelper(signalAfrica)
          fftWinner, freqsWinner = ffthelper(signalWinner)
```

```
In [242]: def plotter(fft, freqs):
              plt.plot(freqs[range(len(fft)//2)], fft[range(len(fft)//2)]) #we just need half of the spectrum
              plt.xlabel("Freq (Hz)")
              plt.ylabel("Amplitude")
              plt.show()
```

In the above image first, I computed FFT values. Then I created another function to plot these values.

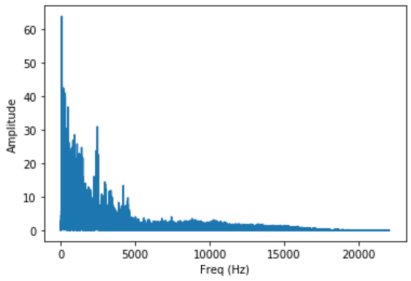
This is FFT of Africa wav file:

```
In [243]: plotter(fftAfrica, freqsAfrica) #fft spectrum of africa
```



And this is FFT of winner wav file:

```
In [244]: plotter(fftWinner, freqsWinner) #fft spectrum of winner
```



Then I created cut function. It takes signal and cut from specified second. It gives 256-point long parts from given signal.

```
In [245]: def cut(data, s_rate, start):
              #cut 256 point sample from the start(second)
              return data[(start*s_rate):((start*s_rate)+256)]
```

```
In [246]: sample1_africa = cut(signalAfrica, s_rate, 10)
          sample2_africa = cut(signalAfrica, s_rate, 20)
          sample3_africa = cut(signalAfrica, s_rate, 30)
          sample1_winner = cut(signalWinner, s_rate, 10)
          sample2_winner = cut(signalWinner, s_rate, 20)
          sample3_winner = cut(signalWinner, s_rate, 30)
```

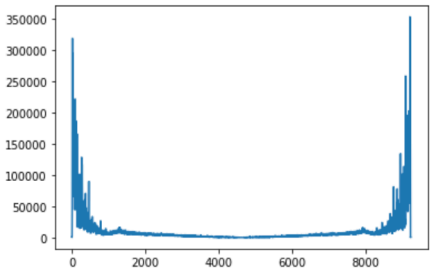
Then I cut wav files from 10,20,30th seconds.

In the below image, I take Fourier transform of these parts.

```
fftAfrica_sample1, freqsAfrica_sample1 = ffthelper(sample1_africa)
fftAfrica_sample2, freqsAfrica_sample2 = ffthelper(sample2_africa)
fftAfrica_sample3, freqsAfrica_sample3 = ffthelper(sample3_africa)
fftWinner_sample1, freqsWinner_sample1 = ffthelper(sample1_winner)
fftWinner_sample2, freqsWinner_sample2 = ffthelper(sample2_winner)
fftWinner_sample3, freqsWinner_sample3 = ffthelper(sample3_winner)
```

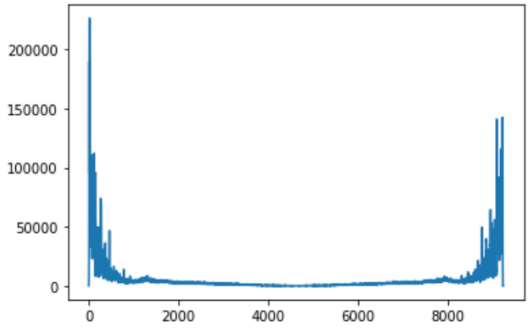
In below image I splitted original wav file into 256 long parts. Then I multiplied 256long part from 10th second with these all 256 parts.

```
In [425]: #fftAfrica_sample1
          x = [fftAfrica[i:i + 256] for i in range(0, len(fftAfrica), 256)]
          arr = []
          for j in range(len(x)):
              total = 0
              for i in range(len(x[j])):
                  total = total + x[j][i] * fftAfrica_sample1[i]
              arr.append(total)
          plt.plot(arr)
          plt.show()
```



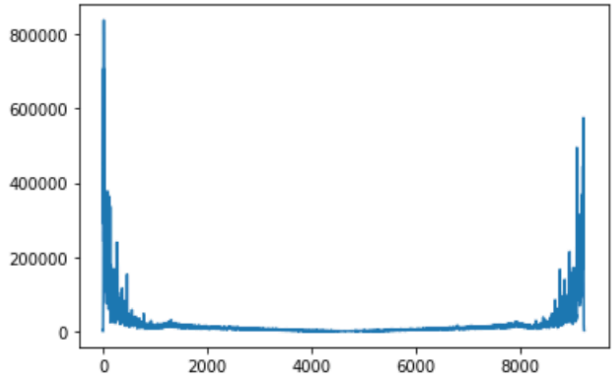
20-

```
In [430]: #fftAfrica_sample2
arr = []
for j in range(len(x)):
    total = 0
    for i in range(len(x[j])):
        total = total + x[j][i] * fftAfrica_sample2[i]
    arr.append(total)
plt.plot(arr)
plt.show()
```



30-

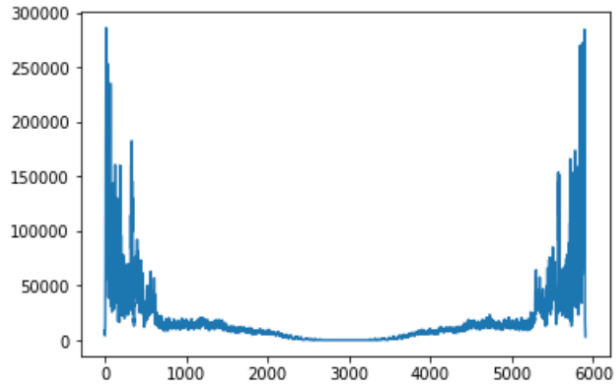
```
In [432]: #fftAfrica_sample3
arr = []
for j in range(len(x)):
    total = 0
    for i in range(len(x[j])):
        total = total + x[j][i] * fftAfrica_sample3[i]
    arr.append(total)
plt.plot(arr)
plt.show()
```



And all these operations for winner wav file.

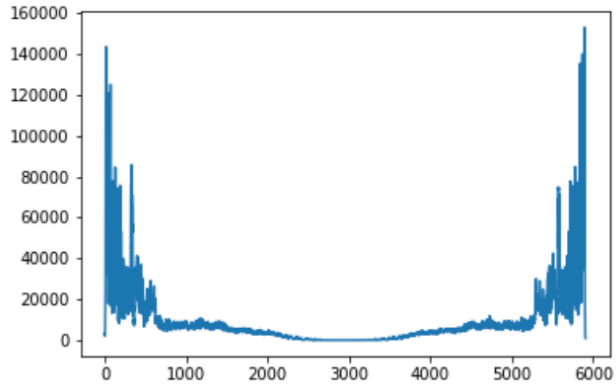
10-

In [415]:
```
#fftWinner_sample1
x = [fftWinner[i:i + 256] for i in range(0, len(fftWinner), 256)]
arr = []
for j in range(len(x)):
    total = 0
    for i in range(len(x[j])):
        total = total + x[j][i] * fftWinner_sample1[i]
    arr.append(total)
plt.plot(arr)
plt.show()
```



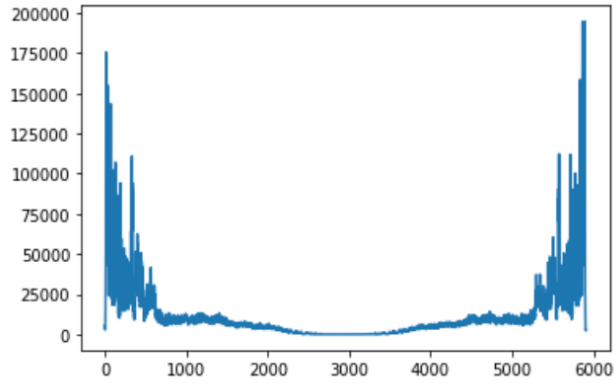
20-

In [416]:
```
#fftWinner_sample2
arr = []
for j in range(len(x)):
    total = 0
    for i in range(len(x[j])):
        total = total + x[j][i] * fftWinner_sample2[i]
    arr.append(total)
plt.plot(arr)
plt.show()
```



30-

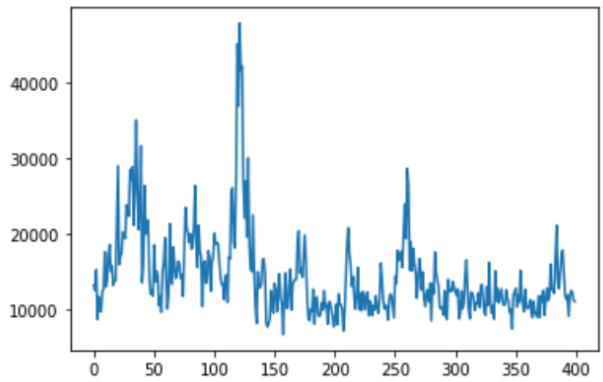
```
In [417]: #fftWinner_sample3
arr = []
for j in range(len(x)):
    total = 0
    for i in range(len(x[j])):
        total = total + x[j][i] * fftWinner_sample3[i]
    arr.append(total)
plt.plot(arr)
plt.show()
```



And below are the results in -200 +200 range but I could not get meaningful information. I must have done something wrong.

Africa-10

```
In [434]: plt.plot(arr[661:1061])
Out[434]: [<matplotlib.lines.Line2D at 0x1c212ef190>]
```

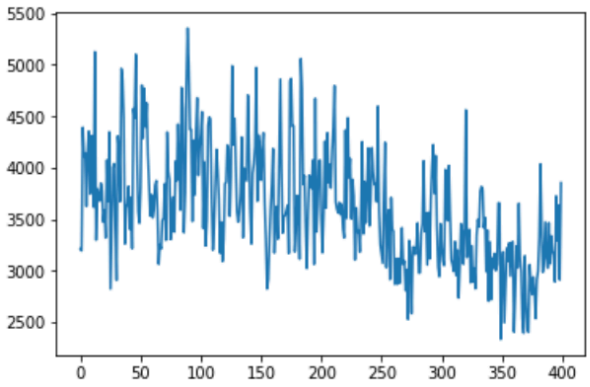


These values 661 and 1061 since 10^{th} second makes 441000^{th} point. When we divide this by 256 it gives us 1722. I divided this by 2 since the graph is symmetric. Result is 861th one.

Africa20-

```
In [431]: plt.plot(arr[1522:1922])
```

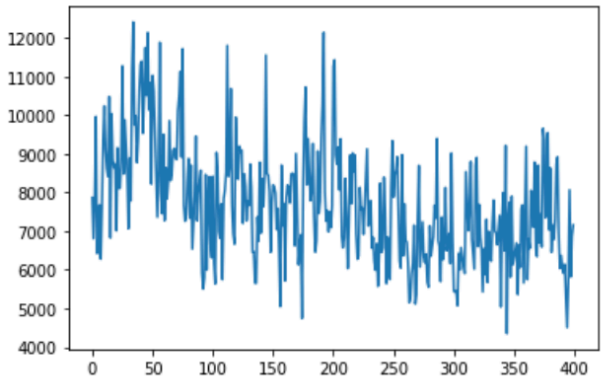
```
Out[431]: [<matplotlib.lines.Line2D at 0x1cdb8bd8d0>]
```



Africa30-

```
In [433]: plt.plot(arr[2383:2783])
```

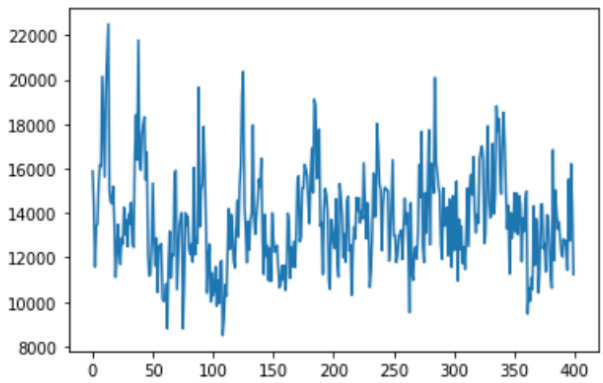
```
Out[433]: [<matplotlib.lines.Line2D at 0x1c2ac4dd50>]
```



Winner10-

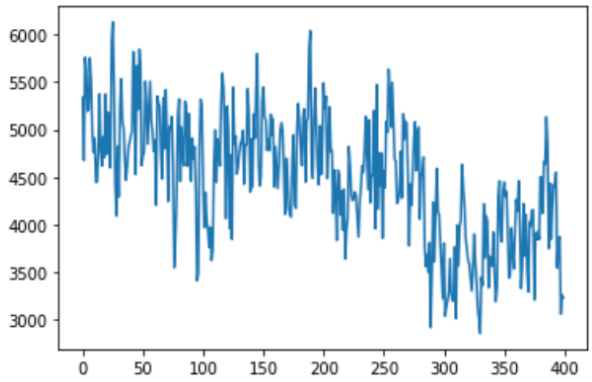
```
In [448]: plt.plot(arr[661:1061])
```

```
Out[448]: [<matplotlib.lines.Line2D at 0x1c2adad750>]
```



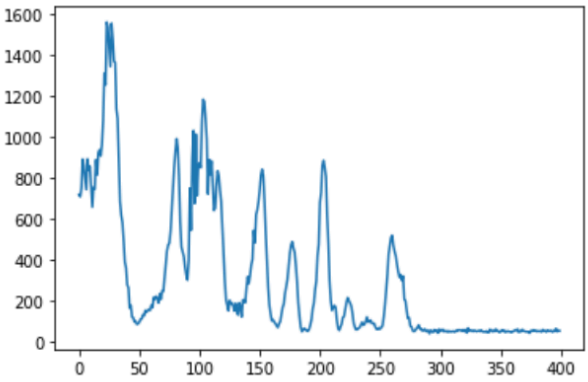
Winner20-

```
In [450]: plt.plot(arr[1522:1922])
Out[450]: [<matplotlib.lines.Line2D at 0x1c21bfb7d0>]
```



Winner30-

```
In [453]: plt.plot(arr[2383:2783])
Out[453]: [<matplotlib.lines.Line2D at 0x1c2d314fd0>]
```



Part c:

Since we convolved with 256 point data we can get similarity for 5.8ms. In order to get 100ms we can increase 256 point to around 3600 point sample. But this can take long time to compute. Instead we can divide this problem into sub parts. 256 point gives us 5.8ms. So in order to get 100ms similarity result we can do this operation 20 times. Dividing problems usually gives faster results.