Ekin Çelebi
150160152

HW5

With this project, we were required to show Fourier convolution of given audio and 3, 256-point length samples picked from audio. We simply try to derive a clever approach for time convolution and obtain similarity between signal and window.

Brief introduction:
As a result of Fourier transform functions can be written as sums of sinusoids, thus each sinusoidal term has magnitude and a phase, expressed as using complex numbers. Therefore the original function is represented as a series of magnitudes and phase coefficients.
With help of Fourier transform, function is converted to frequency domain from time domain

Why Fast Fourier transform?
Normal arithmetic operations to compute Fourier transform requires N*N computation, however this number can be reduced up to NlogN with FFT algorithms.

What is convolution?
Convolution formula is described below.

$$(f*g)[n]=\sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

A powerful approach to convolution is using Fourier transforms.
Convolving two functions is equal to multiplication of these functions in the frequency domain.
Time domain equivalent of convolution can be obtained by:

1) Taking Fourier transform of both window and signal
2) Multiplying two transformed functions
3) Taking the inverse Fourier transform of the result.

$$\mathcal{F}\{f*g\}=F\cdot G$$

Steps for the project:

1)Picking 256 point samples from audio, from now these samples are called as window

2) FFT of window is slide over the audio and, multiplied with FFT of each distinct 256-point parts of audio. This computation will result to convolution of Fourier domain.

3) Taking average of each multiplication vector.

4) Plotting

The points plotted are the result of magnitudes of the FFTs and the similarity of signals.

Implementation
For simplicity reasons, original sound data is divided into 256-point parts. To derive 10, 20, and 30 second windows, the closest point of these time indexes are picked among the beginning indexes of each 256-point groups. (These groups are named as blocks below.)

```
second 10 index starts:  441000
block number:  1723
second 10 index should now start at:  441088
################
second 20 index starts:  882000
block number:  3446
second 20 index should now start at:  882176
################
second 30 index starts:  1323000
block number:  5168
second 30 index should now start at:  1323008
```

DIF Radix2 FFT algorithm is derived from HW4, with some changes.
my_fft() is the final version of the function.

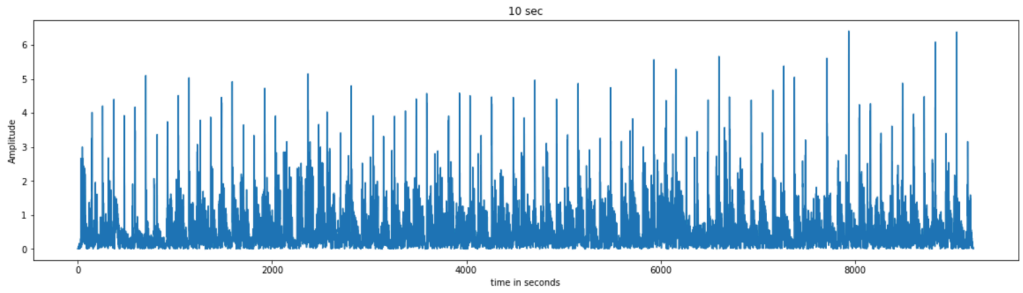
```
def radix2(inarray,N,outarray,twid):
    '''
    inarray: audio samples
    N:number of radix points
    outarray:where to append results
    twid: to be used to calculate twiddle

    recursively apply dif fft blocks
    resuls are in bit reverse order
    '''

    xrange = int(N/2)


    if N == 1:

        outarray.append(inarray[0])
        return


    ###twiddle function has changed
    def twiddle(x,point):
        return np.exp(-1j*(2*np.pi/point)*x)

    upper_half = np.zeros((xrange,),dtype=np.complex_)
    lower_half = np.zeros((xrange,),dtype=np.complex_)


    ###lower half implementation has also changed
    for i in range(0,xrange):
        upper_half[i] = inarray[i]+inarray[i+xrange]
        lower_half[i] = (inarray[i]-inarray[i+xrange])*twiddle(i*(twid/N),twid)


    radix2(upper_half,xrange,outarray,twid)
    radix2(lower_half,xrange,outarray,twid)
```

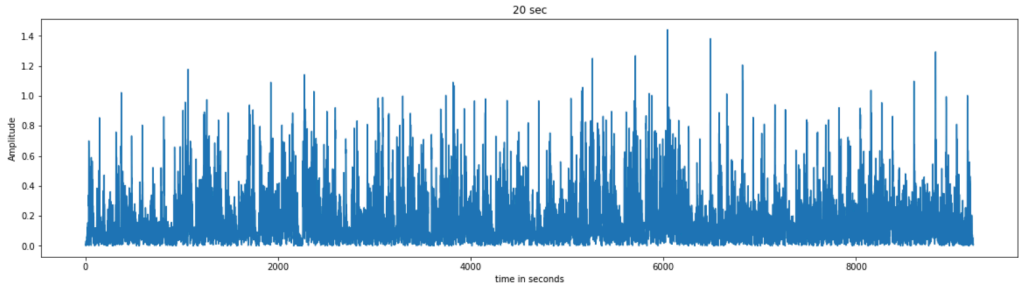
```
#### additional reverse function to find bit reverse of a given number
def reverse(num, bitlen):
    ####number: number to be reversed,bitlen: bit length
    result = 0
    for i in range(bitlen):
        if (num >> i) & 1: result |= 1 << (bitlen - 1 - i)
    return result


def reverse_bits(inarray,point,bitlen):
    '''
    input array
    number of points
    bit length
    '''
    outarray = np.zeros((point,),dtype=np.complex_)
    for i in range(point):
        outarray[i] = inarray[reverse(i,bitlen)]
    return outarray
```

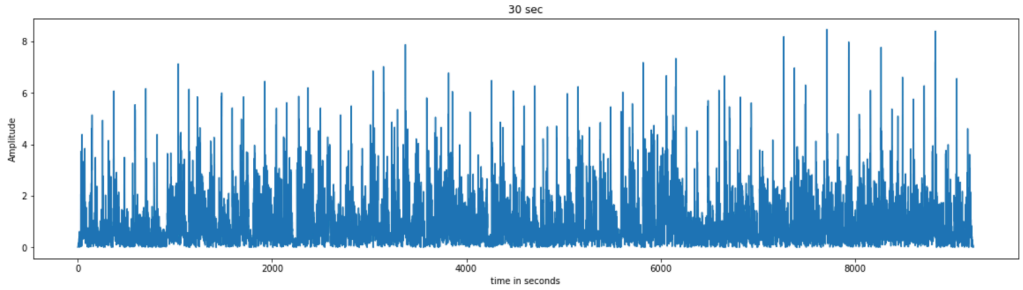
```
def my_fft(inarray):
    radix_result = []
    return_arr = np.zeros((256,),dtype=np.complex_)
    radix2(inarray,256,radix_result,256)
    return_arr = reverse_bits(radix_result,256,8)
    return return_arr
```

Part A:

Original sound data is divided into 256 point blocks, 256 point window slides over each block and, their FFTs are multiplied.
Average value of each block of FFT multiplications are calculated.
As a result, an array of complex values ,is obtained.
To plot the calculated array of complex numbers, abs() function is used(magnitude calculation).
Below is shown the respective function and the derived results.



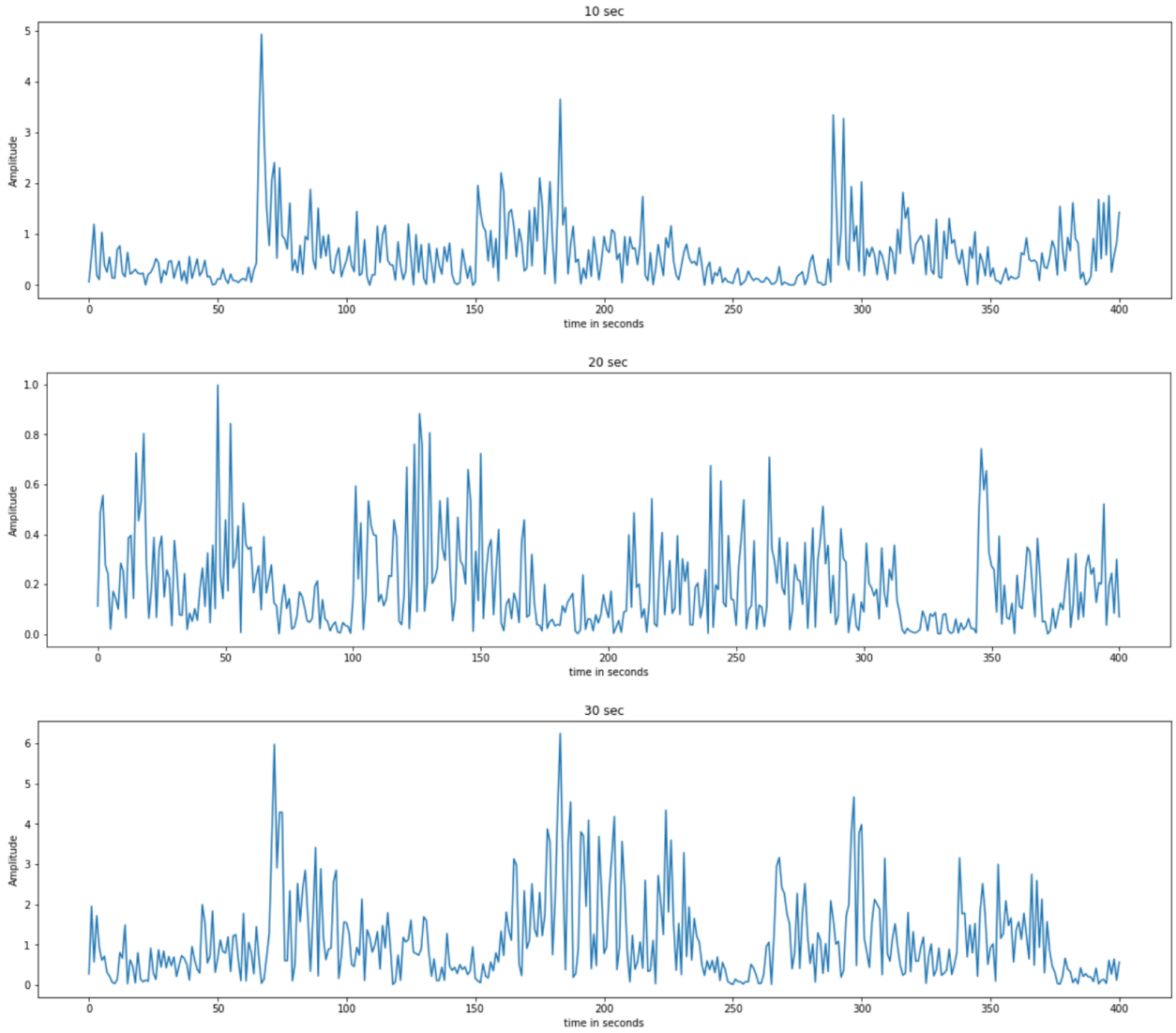




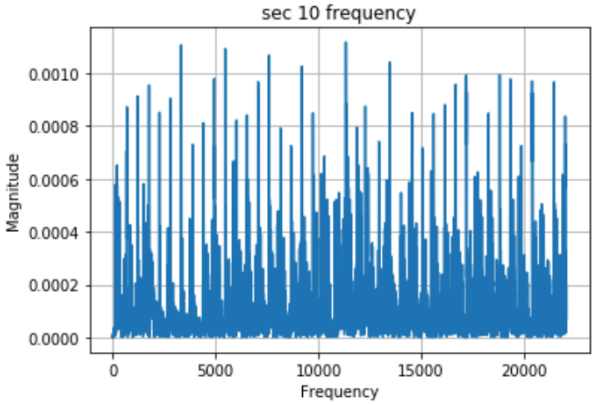

```
def convolve_256(x,h):

    '''
    FFT of window is slide over the audio and, multiplied with
    FFT of each distinct 256-point parts of audio.
    This computation will result to convolution of Fourier domain.
    '''

    lenX = len(x)
    lenH = len(h)

    size = math.floor(lenX/lenH)

    fft_h = my_fft(h)



    conv = np.zeros((size,),dtype=complex) ## holds average values of window multiplications
    temp = np.zeros((lenH,),dtype=complex) ## holds shifted windows of the signal



    for i in range(size):
        temp = x[(i*lenH):(i*lenH)+lenH]

        fft_x = my_fft(temp)
        tempVal = np.average(np.multiply(fft_x,fft_h)) #multiply two FFT's
        conv[i] = tempVal


    return conv
```

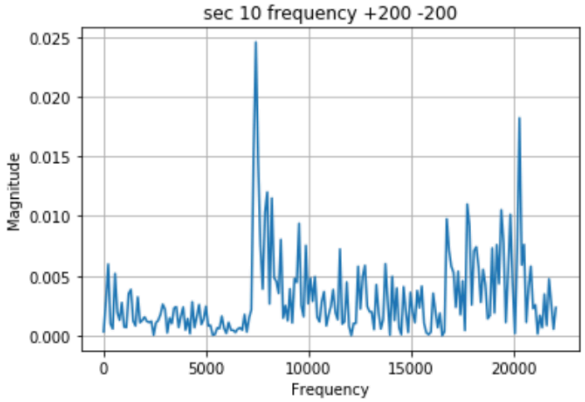
Part B:

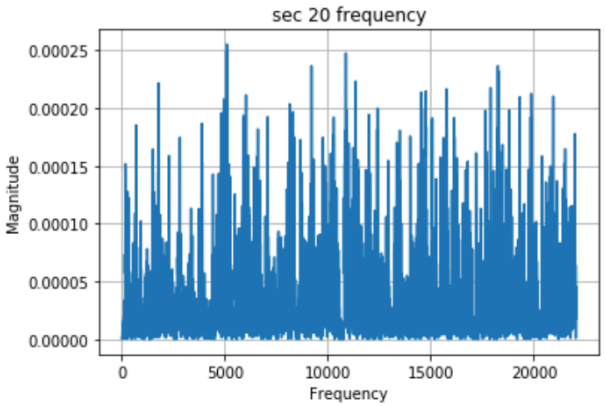
For this part, only the +-200 from the overlapping value need to be shown.

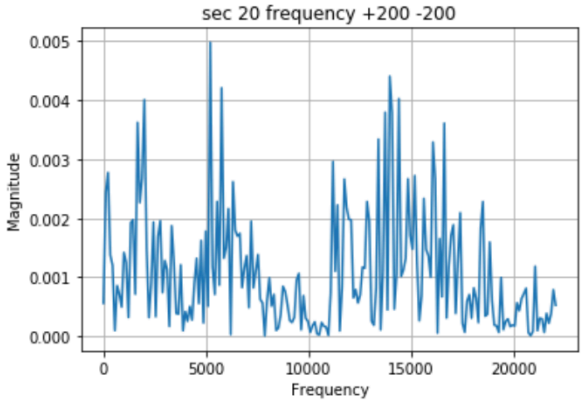


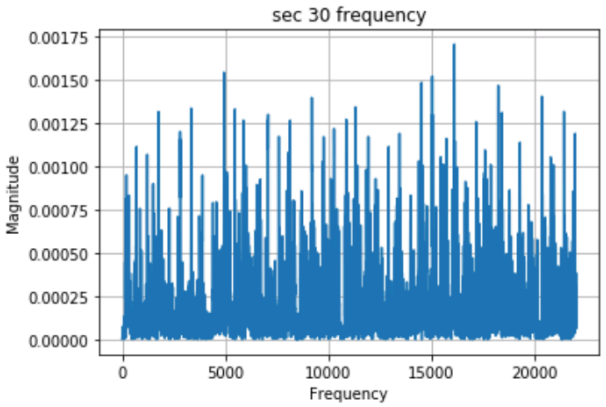
Additionally: I implemented frequency equavalent plot of the convolution problem. Using radix2_plot() method.

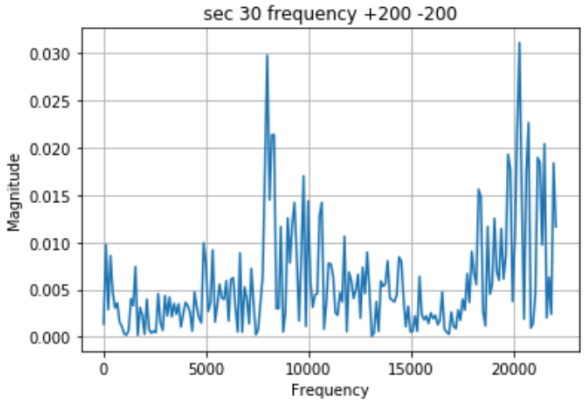













```
def radix2_plot(radix_out,sampling_rate,in_title):
    n = len(radix_out)

    yf = radix_out
    xf = np.linspace(0.0,int(sampling_rate/2),int(n/2))
    fig, ax = plt.subplots()
    ax.plot(xf, 2.0/n * np.abs(yf[:int(n//2)]))
    plt.grid()
    plt.title(in_title)
    plt.xlabel('Frequency')
    plt.ylabel('Magnitude')
    return plt.show()
```

Part C:

If our intend is to find similarity between the signal and a random sample from the the audio, cross correlation is another option to derive similarity between two signals. This method allows us to make comparison and results the match of two given signals. Using the correlation coefficient, similarity of the window and the signal is calculated, for correspondence the results become maximum.
Below is the formula of the cross correlation, that a similariy of this formula with convolution formula is obviously seen.

$$f \star g = \int_{-\infty}^{\infty} f(x+u)g(u)^* \, du :$$

References:

https://class.ece.uw.edu/235dl/EE235/Project/lesson17/lesson17.html
https://www.cs.unm.edu/~williams/cs530/theorems6.pdf
https://web.stanford.edu/class/cs279/lectures/lecture9.pdf