ISTANBUL TECHNICAL UNIVERSITY

SIGNALS AND SYSTEMS

Homework – V

Report

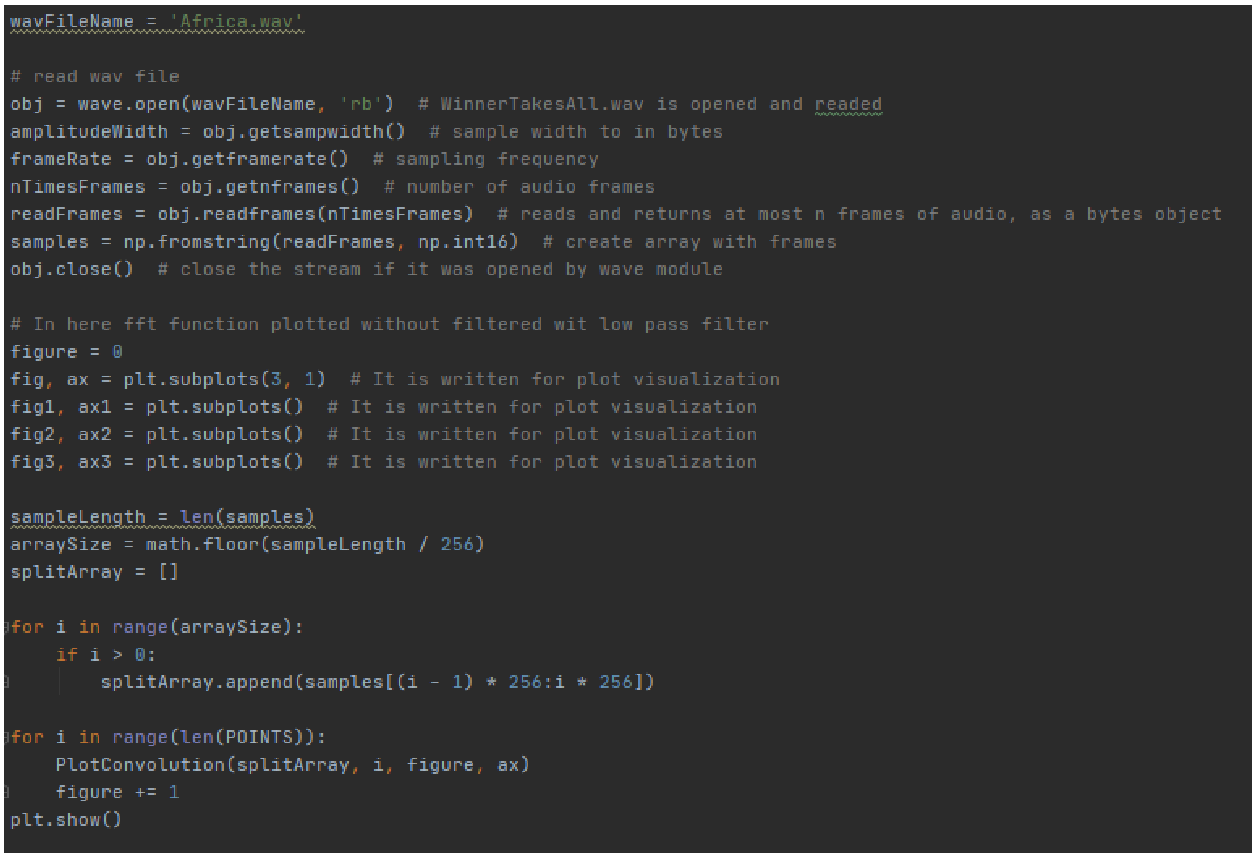
150160529

Nurettin Kağan Çocalak

Problem 1

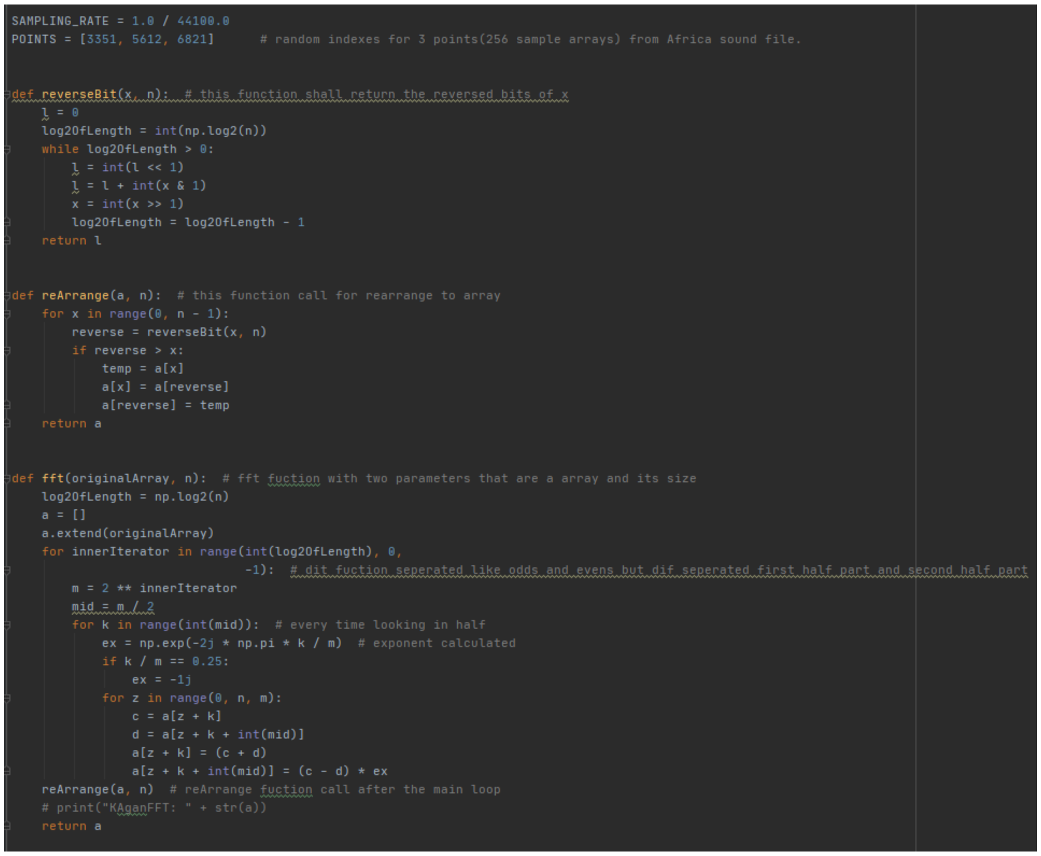
In this homework, when I was coding, I used Jupyter Notebook.

a) In this section, three graphs that shows variation of the Fourier convolution between the 256 point length audio record samples at three spots and the overall audio record by shifting the 256-point length window. Firstly, took the Africa.wav file samples and I divide to 256-size arrays (Figure-1). Then, index of this split arrays so selected window sent to fft function (Figure-2) and result is assigned to fftOfWindow variable. This variable's real part is taken only this time. After that, thanks to for loop, I took all samples of this window and sent them to fft function. Finally I took all one by one return samples and multiply with my specific window. Then sum of them appended to results array (Figure-3).



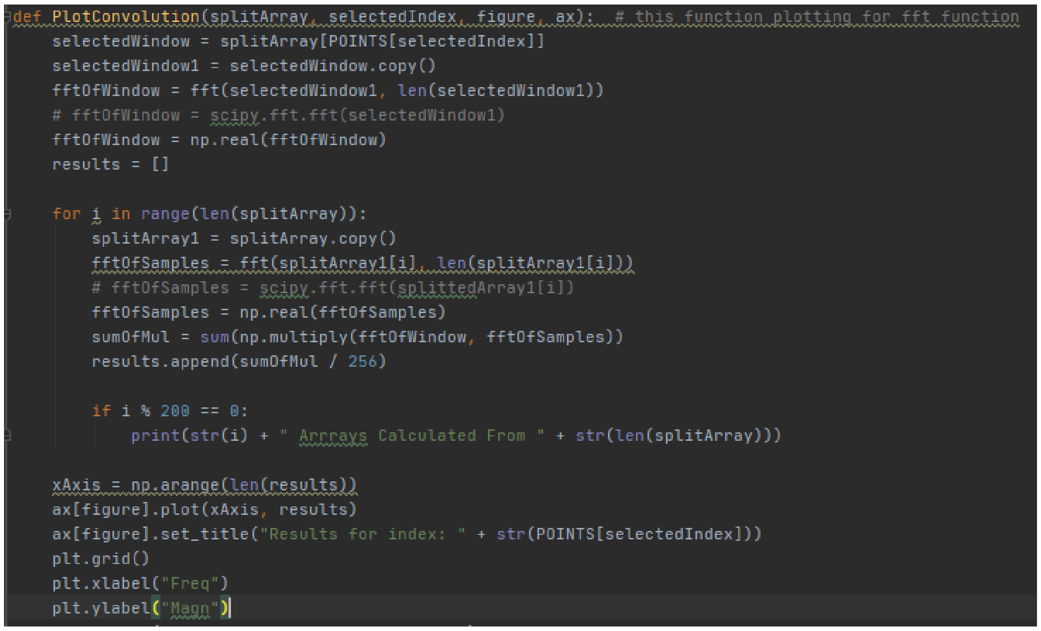
```python
wavFileName = 'Africa.wav'

# read wav file
obj = wave.open(wavFileName, 'rb')  # WinnerTakesAll.wav is opened and readed
amplitudeWidth = obj.getsampwidth()  # sample width to in bytes
frameRate = obj.getframerate()  # sampling frequency
nTimesFrames = obj.getnframes()  # number of audio frames
readFrames = obj.readframes(nTimesFrames)  # reads and returns at most n frames of audio, as a bytes object
samples = np.fromstring(readFrames, np.int16)  # create array with frames
obj.close()  # close the stream if it was opened by wave module

# In here fft function plotted without filtered wit low pass filter
figure = 0
fig, ax = plt.subplots(3, 1)  # It is written for plot visualization
fig1, ax1 = plt.subplots()  # It is written for plot visualization
fig2, ax2 = plt.subplots()  # It is written for plot visualization
fig3, ax3 = plt.subplots()  # It is written for plot visualization

sampleLength = len(samples)
arraySize = math.floor(sampleLength / 256)
splitArray = []

for i in range(arraySize):
    if i > 0:
        splitArray.append(samples[(i - 1) * 256:i * 256])

for i in range(len(POINTS)):
    PlotConvolution(splitArray, i, figure, ax)
    figure += 1
plt.show()
```

Figure-1



```python
SAMPLING_RATE = 1.0 / 44100.0
POINTS = [3351, 5612, 6821]      # random indexes for 3 points(256 sample arrays) from Africa sound file.


def reverseBit(x, n):  # this function shall return the reversed bits of x
    l = 0
    log2OfLength = int(np.log2(n))
    while log2OfLength > 0:
        l = int(l << 1)
        l = l + int(x & 1)
        x = int(x >> 1)
        log2OfLength = log2OfLength - 1
    return l


def reArrange(a, n):  # this function call for rearrange to array
    for x in range(0, n - 1):
        reverse = reverseBit(x, n)
        if reverse > x:
            temp = a[x]
            a[x] = a[reverse]
            a[reverse] = temp
    return a


def fft(originalArray, n):  # fft fuction with two parameters that are a array and its size
    log2OfLength = np.log2(n)
    a = []
    a.extend(originalArray)
    for innerIterator in range(int(log2OfLength), 0,
                               -1):  # dit fuction seperated like odds and evens but dif seperated first half part and second half part
        m = 2 ** innerIterator
        mid = m / 2
        for k in range(int(mid)):  # every time looking in half
            ex = np.exp(-2j * np.pi * k / m)  # exponent calculated
            if k / m == 0.25:
                ex = -1j
            for z in range(0, n, m):
                c = a[z + k]
                d = a[z + k + int(mid)]
                a[z + k] = (c + d)
                a[z + k + int(mid)] = (c - d) * ex
    reArrange(a, n)  # reArrange fuction call after the main loop
    # print("KAganFFT: " + str(a))
    return a
```

Figure-2



```python
def PlotConvolution(splitArray, selectedIndex, figure, ax):  # this function plotting for fft function
    selectedWindow = splitArray[POINTS[selectedIndex]]
    selectedWindow1 = selectedWindow.copy()
    fftOfWindow = fft(selectedWindow1, len(selectedWindow1))
    # fftOfWindow = scipy.fft.fft(selectedWindow1)
    fftOfWindow = np.real(fftOfWindow)
    results = []

    for i in range(len(splitArray)):
        splitArray1 = splitArray.copy()
        fftOfSamples = fft(splitArray1[i], len(splitArray1[i]))
        # fftOfSamples = scipy.fft.fft(splittedArray1[i])
        fftOfSamples = np.real(fftOfSamples)
        sumOfMul = sum(np.multiply(fftOfWindow, fftOfSamples))
        results.append(sumOfMul / 256)

        if i % 200 == 0:
            print(str(i) + " Arrrays Calculated From " + str(len(splitArray)))

    xAxis = np.arange(len(results))
    ax[figure].plot(xAxis, results)
    ax[figure].set_title("Results for index: " + str(POINTS[selectedIndex]))
    plt.grid()
    plt.xlabel("Freq")
    plt.ylabel("Magn")
```

Figure-3

You can see on this graph in Figure-4.

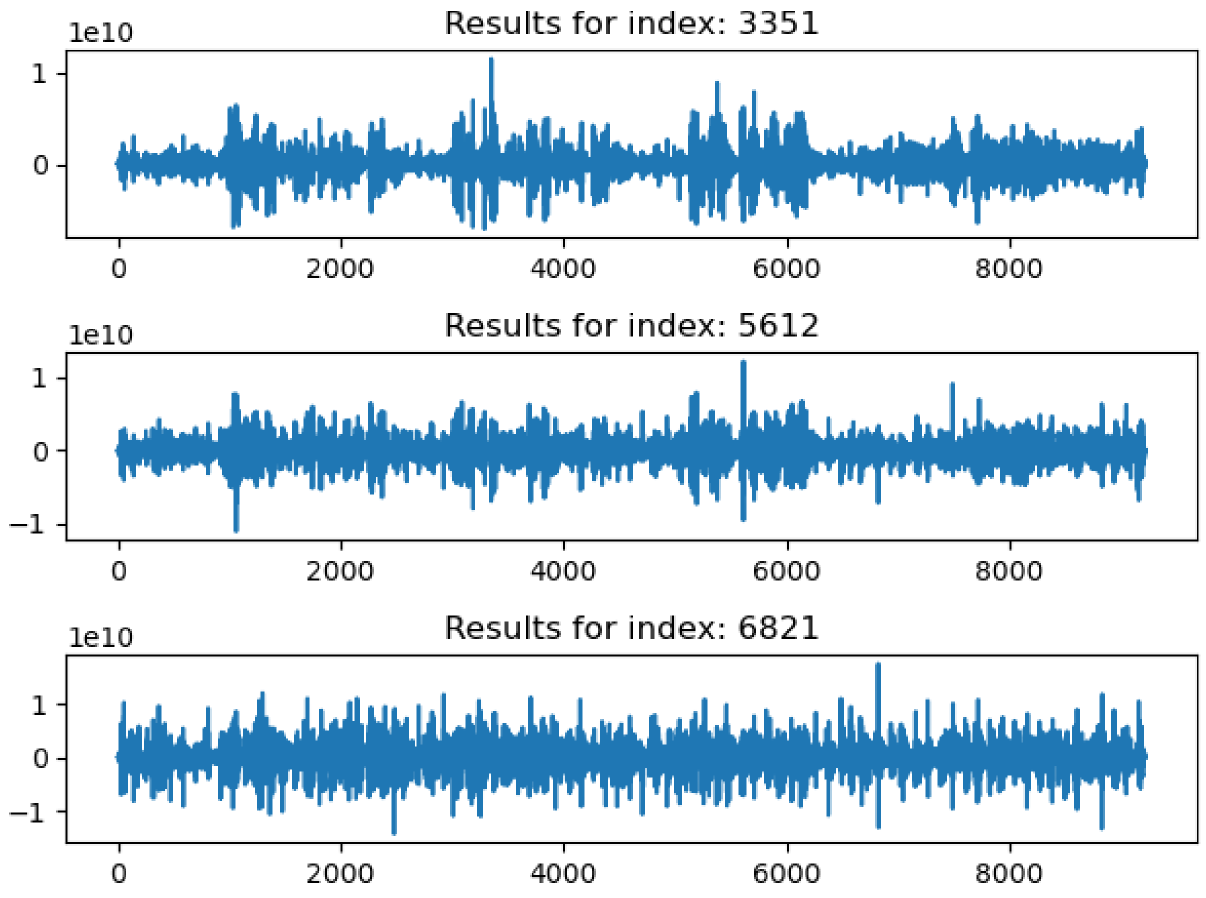
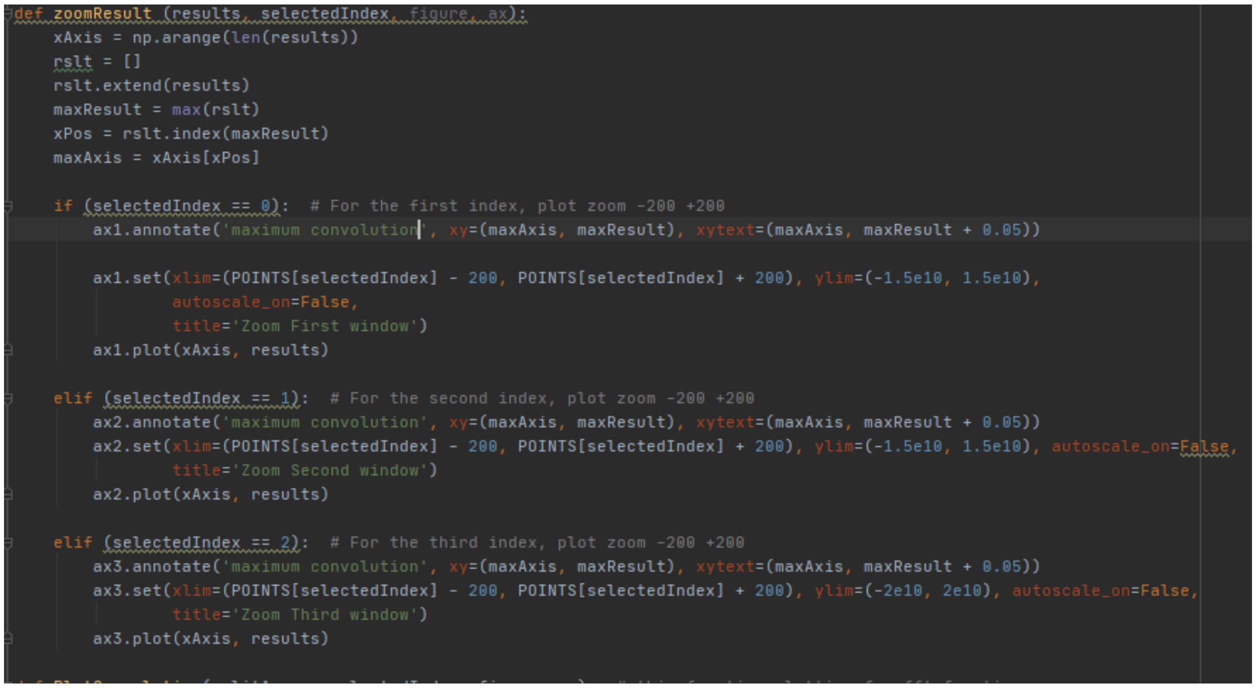


Figure-4

b) In this section, additional graphs by zooming in ±200 convolutional values range around the window location where 256-point sample is convolved with itself (for each of three graphs) drew. And maximum convolution value point is shown. For showing this graphs, I create a function that is zoomResult(Figure-5). And it called from PlotConvolution function. Graphs are below (Figure-6, Figure-7).



```python
def zoomResult (results, selectedIndex, figure, ax):
    xAxis = np.arange(len(results))
    rslt = []
    rslt.extend(results)
    maxResult = max(rslt)
    xPos = rslt.index(maxResult)
    maxAxis = xAxis[xPos]

    if (selectedIndex == 0):  # For the first index, plot zoom -200 +200
        ax1.annotate('maximum convolution', xy=(maxAxis, maxResult), xytext=(maxAxis, maxResult + 0.05))

        ax1.set(xlim=(POINTS[selectedIndex] - 200, POINTS[selectedIndex] + 200), ylim=(-1.5e10, 1.5e10),
                autoscale_on=False,
                title='Zoom First window')
        ax1.plot(xAxis, results)

    elif (selectedIndex == 1):  # For the second index, plot zoom -200 +200
        ax2.annotate('maximum convolution', xy=(maxAxis, maxResult), xytext=(maxAxis, maxResult + 0.05))
        ax2.set(xlim=(POINTS[selectedIndex] - 200, POINTS[selectedIndex] + 200), ylim=(-1.5e10, 1.5e10), autoscale_on=False,
                title='Zoom Second window')
        ax2.plot(xAxis, results)

    elif (selectedIndex == 2):  # For the third index, plot zoom -200 +200
        ax3.annotate('maximum convolution', xy=(maxAxis, maxResult), xytext=(maxAxis, maxResult + 0.05))
        ax3.set(xlim=(POINTS[selectedIndex] - 200, POINTS[selectedIndex] + 200), ylim=(-2e10, 2e10), autoscale_on=False,
                title='Zoom Third window')
        ax3.plot(xAxis, results)
```

Figure-5

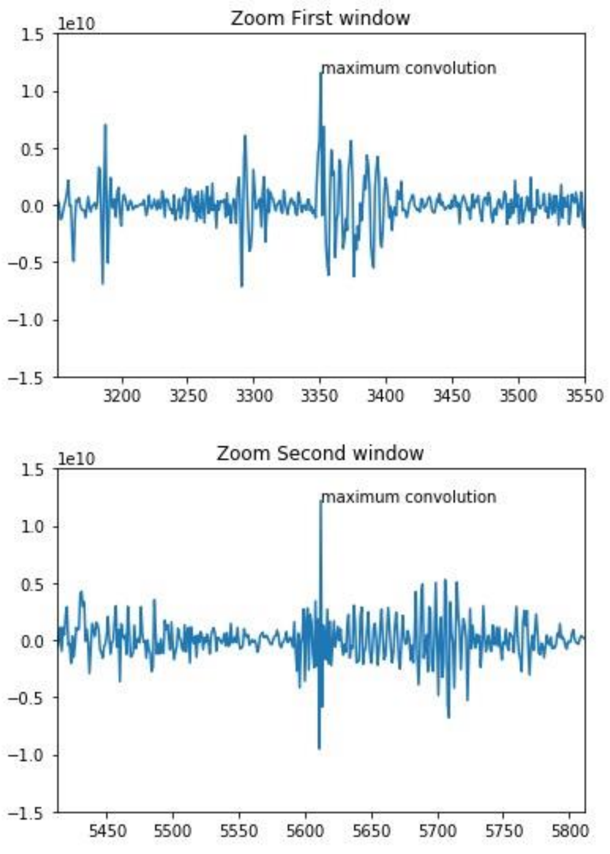


Figure-6

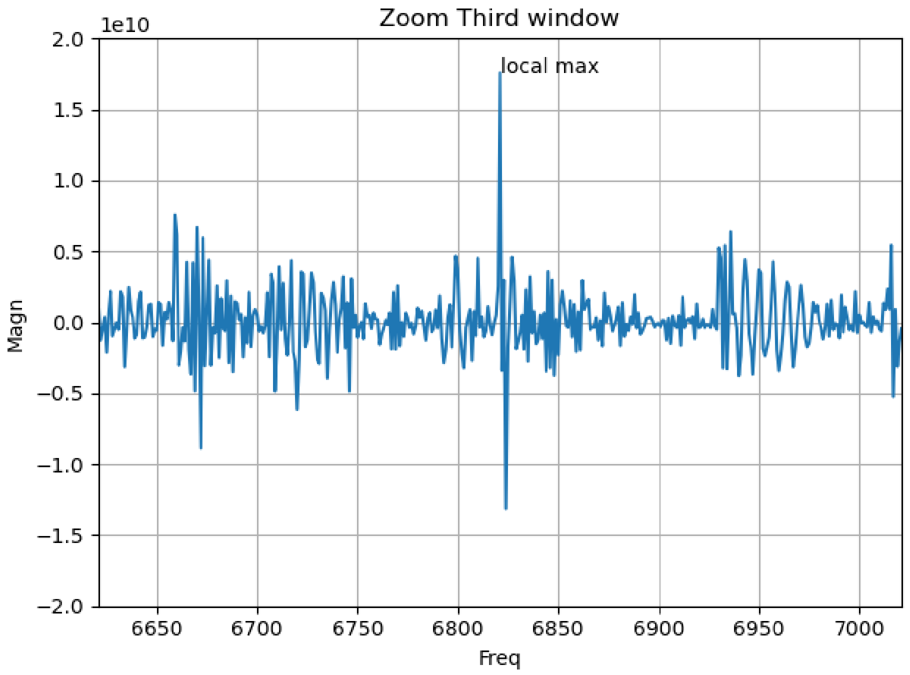


Figure-7

c) I think, maybe some pass filters can be used for this purpose.