



EAST WEST UNIVERSITY

PROJECT

Course Title: Design & Simulation

Course Code: ETE399

Submitted By:

Parveen Akther Diti

2016-2-55-024

Submitted to:

S. M. Raiyan Chowdhury

Lecturer, Dept of ECE

Date: 07.01.2021

Project Name: Audio file to Numpy Array.

a) Objective:

Audio files can be of various formats depending on their compression schemes (e.g. wav, mp3). In this project, we are going to deal with the audio files of .wav format (uncompressed raw audio file). Python has a variety of ways to read write and process audio files. The first step of audio file processing is to convert them into numerical arrays (i.e. numpy arrays) in python. Note that, audio signal can have more than one channels. If there are ' k ' channels and the number of samples in audio signal is ' N ', then the numpy array a 2-D array of size $N \times k$. Here given the code to play the audio, convert audio data into numpy array (we are using a audio file 'test.wav' kept in the same directory of the code) and normalize the array as well.

b) Problem statement:

Problem no: 01

Code:

```
from scipy.io.wavfile import read, write

from matplotlib import pyplot as plt

from playsound import playsound

import numpy as np

audio_sig = read('test.wav')

n_channel = len(audio_sig)

sampling_rate = audio_sig[0]

audio_array = np.array(audio_sig[1])

# Normalizing the Audio Array

audio_array_norm = np.zeros((len(audio_array[:,0]), n_channel)) # Initialize Norm Array

max_sigs = np.zeros(n_channel) # Max value of each channel

for i in range(n_channel):

    max_sigs[i] = max(audio_array[:,i])

    audio_array_norm[:,i] = audio_array[:,i]/max_sigs[i]

Duration = len(audio_array[:,0])/sampling_rate
```

```
Time = np.linspace(0, Duration, len(audio_array[:,0]))
```

```
for i in range(n_channel):
```

```
    plt.figure(i+1, dpi=300)
```

```
    plt.plot(Time, audio_array_norm[:,i])
```

```
    plt.xlabel("Time (sec)")
```

```
    plt.ylabel("Normalized Audio of Channel-" + str(i+1))
```

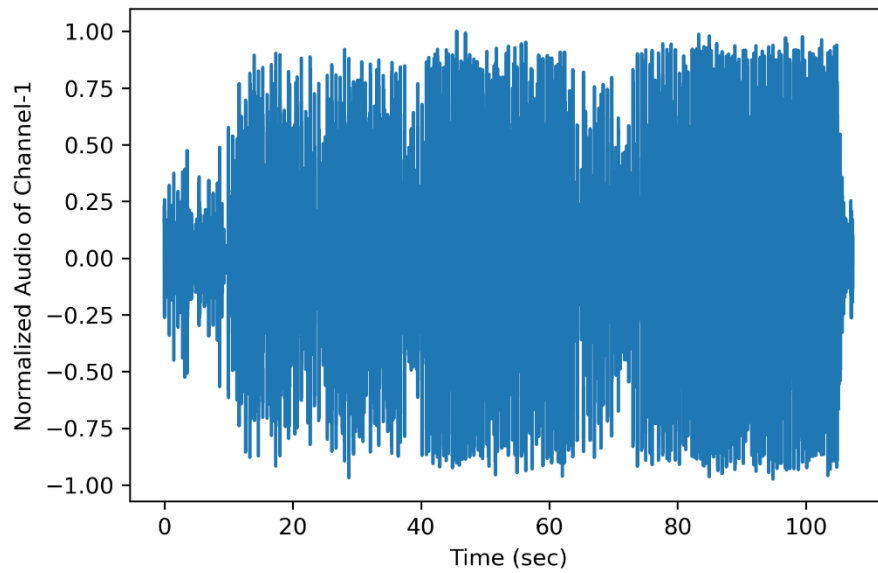


Fig:01

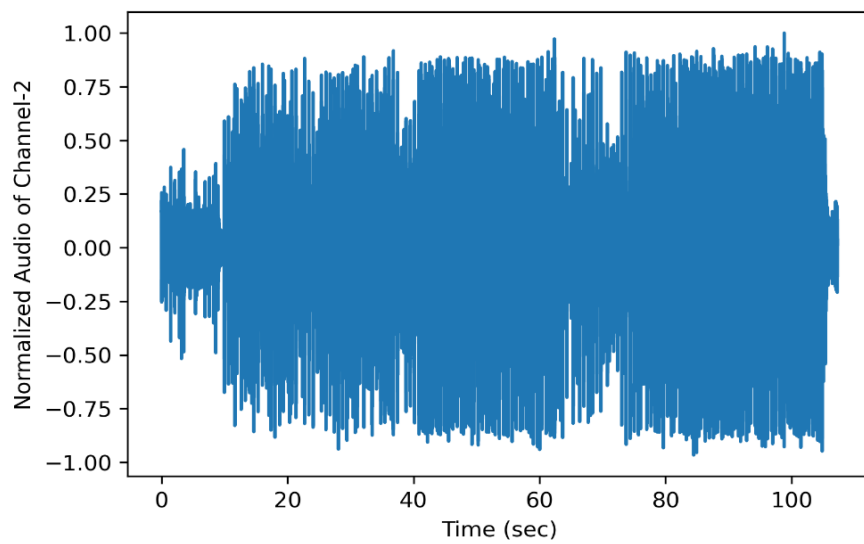


Fig::02

Discussion:

From first task we run the 'test.wav' file and played the audio by converting this audio file into Numpy array. Then we normalized the Audio signals for each channels. Then we found the pictures of plotted normalized audio signals for each channel.

Problem no:02

Code:

```
audio_array_norm = np.zeros((len(audio_array[:,0]), n_channel))

max_sigs = np.zeros(n_channel)

for i in range(n_channel):

    max_sigs[i] = max(audio_array[:,i])

    audio_array_norm[:,i] = audio_array[:,i]/max_sigs[i]

for i in range(n_channel):

    plt.figure(i+1)

    plt.plot(Time, audio_array_norm[:,i])

    plt.xlabel("Time (sec)")

    plt.ylabel("Normalized Audio of Channel -" + str(i+1))
```

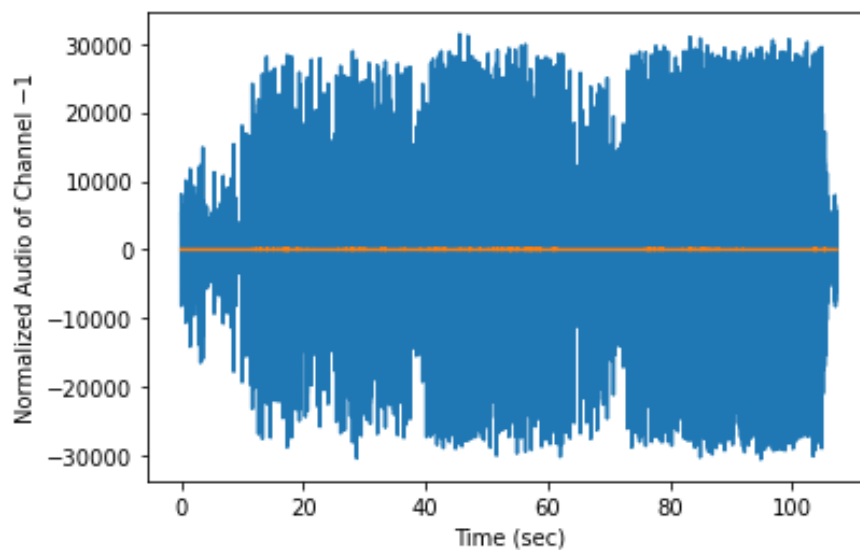


Fig: 01

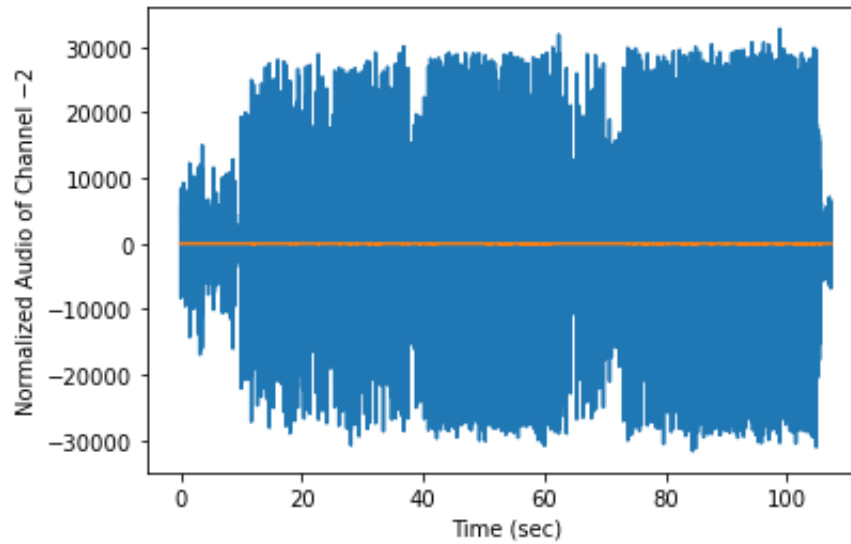


Fig: 02

Discussion: From this task we also normalized the audio signals and decided the energy & maximum intensity for this audio signals to each channels. We got the plot of audio signals those are normalized by Numpy array.

Problem no: 03

Code:

```
from numpy.fft import fft
h=[]
for i in range(n_channel):
    x=fft(audio_array_norm[:,i])
    h.append(x)
for i in range(n_channel):
    plt.figure(i+1)
    plt.plot(Time, h[i])
```

```

plt.xlabel("Time (sec)")

cs=0

for i in range(n_channel):

    cs=cs+h[i]

maxcs=max(cs)

norcs=cs/maxcs

plt.figure(1)

plt.plot(Time, cs)

plt.xlabel("Time (sec)")

```

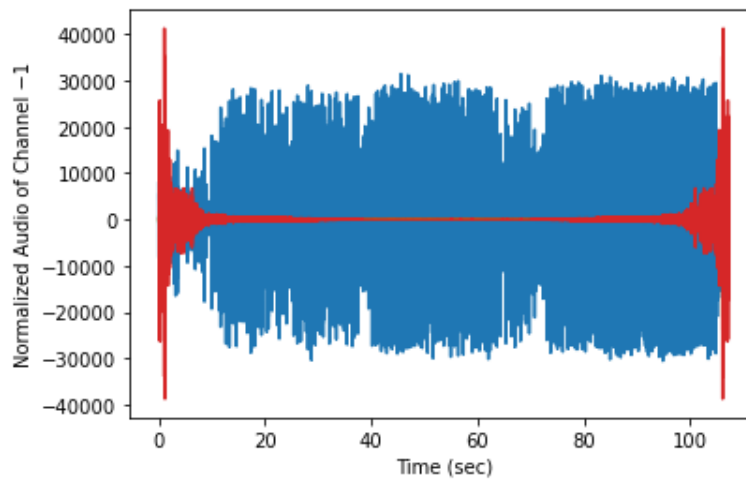


Fig: 01

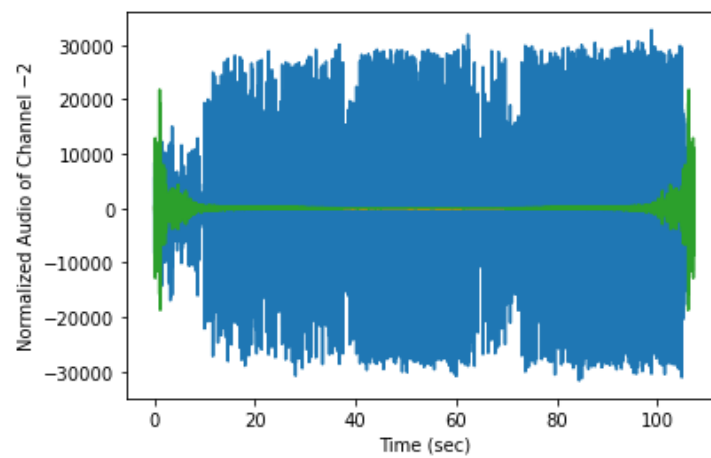


Fig: 02

Discussion: After getting the Numpy array Audio signals and there normalized signals we determined the audio signals by generating spectrums for both channels. And also normalized the spectrums of audio file od each channels.

Problem No: 04

Code:

```
import librosa

import librosa.display

fs = 10e3

T= float(1/fs)

del_time = 10*T

scale, sr = librosa.load('test.wav', duration=del_time)

Frame_size=2048

Hop_size=512

S_scale = librosa.stft(scale,n_fft=Frame_size,hop_length=Hop_size)

Y_scale= np.abs(S_scale)**2

def plot_spectrogram(Y,sr,hop_length,y_axis="linear"):

    plt.figure(figsize=(10,10))

    librosa.display.specshow(Y,sr=sr,hop_length=hop_length,x_axis="time",y_axis=y_axis)

Y_log_scale = librosa.power_to_db(Y_scale)

plot_spectrogram(Y_log_scale, sr, Hop_size)
```

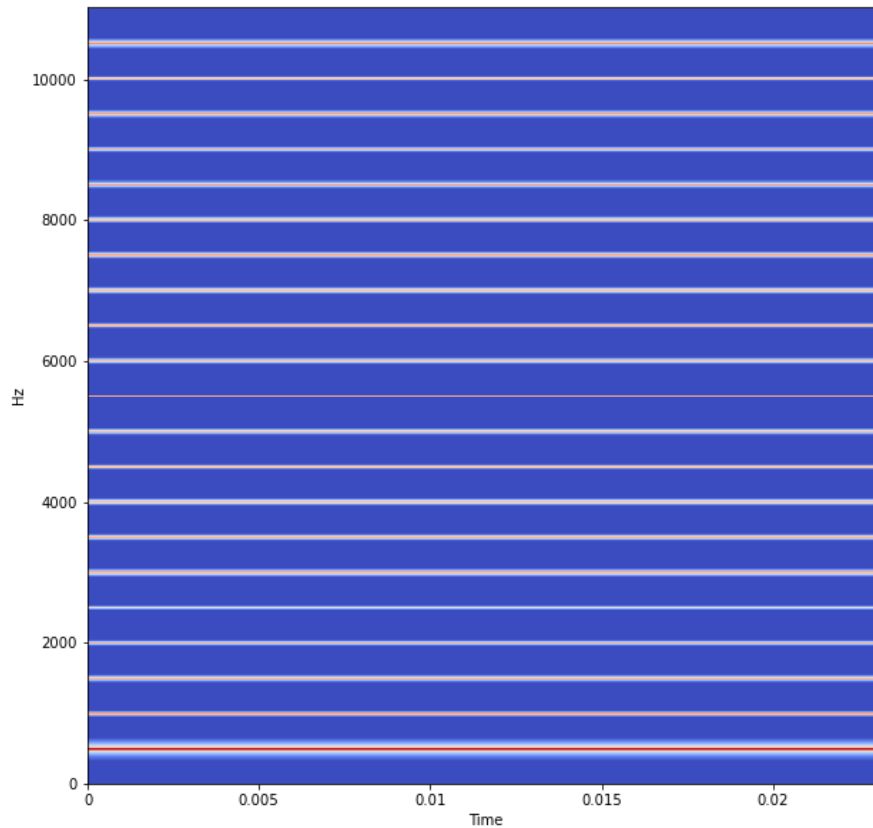


Fig: Spectrogram for each channel using FFT & STFFT

Discussion: By using STFFT we determined the Spectrogram of audio signals for each channels. We used some given value for sampling the time period. And we got a smooth and clear spectrogram plotted audio signals.

Problem No: 05

Code:

```
scale, sr = librosa.load('test.wav')

Frame_size=2048

Hop_size=512

S_scale = librosa.stft(scale,n_fft=Frame_size,hop_length=Hop_size)

Y_scale= np.abs(S_scale)**2

def plot_spectrogram(Y,sr,hop_length,y_axis="linear"):
```



```
plt.figure(figsize=(10,10))

librosa.display.specshow(Y,sr=sr,hop_length=hop_length,x_axis="time",y_axis=y_axis)

Y_log_scale = librosa.power_to_db(Y_scale)

plot_spectrogram(Y_log_scale, sr, Hop_size)
```

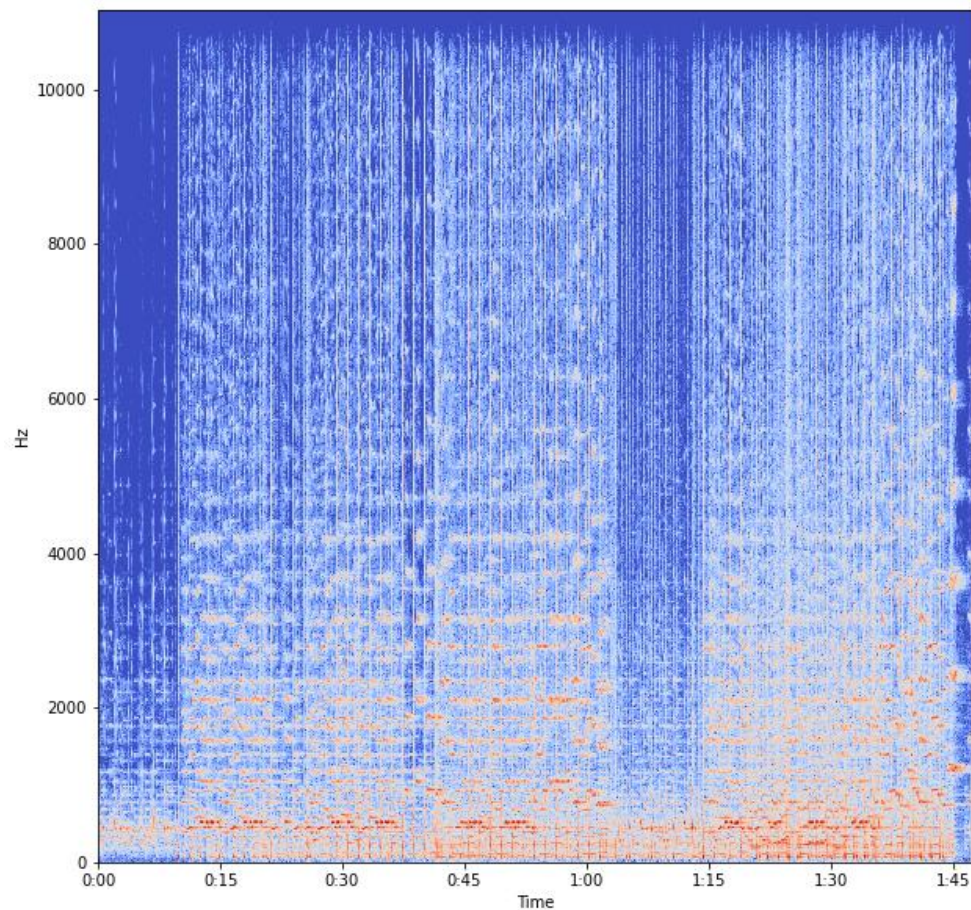


Fig: Spectrogram for both channel

Discussion: After generating spectrogram for each channels we also found out for both channels of the audio signals.

Problem no: 06

➤ **Parson's Code:** *DRUDDURD

Discussion: After generating spectrogram we found the pearson's code for this audio signals.

Problem no: 07

Code:

```
min_sigs = np.zeros(n_channel)

for i in range(n_channel):

    max_sigs[i] = max(audio_array[:,i])

    min_sigs[i]=min(audio_array[:,i])

def quantizer(x, qmin=-1, qmax=1, level=8):

    x_normalize = (x-qmin) * (level-1) / (qmax-qmin)

    x_normalize[x_normalize > level - 1] = level - 1

    x_normalize[x_normalize < 0] = 0

    x_normalize_quant = np.around(x_normalize)

    x_quant = (x_normalize_quant) * (qmax-qmin) / (level-1) + qmin

    return x_quant

x_quan = np.zeros((len(audio_array[:,0]), n_channel))

for i in range(n_channel):

    x_quan[:,i]=quantizer(audio_array[:,i],min_sigs[i],max_sigs[i],level=8)

for i in range(n_channel):

    plt.figure(i+1)

    plt.plot(Time,x_quan[:,i] )

    plt.ylabel("Quantized both normalized signals")
```

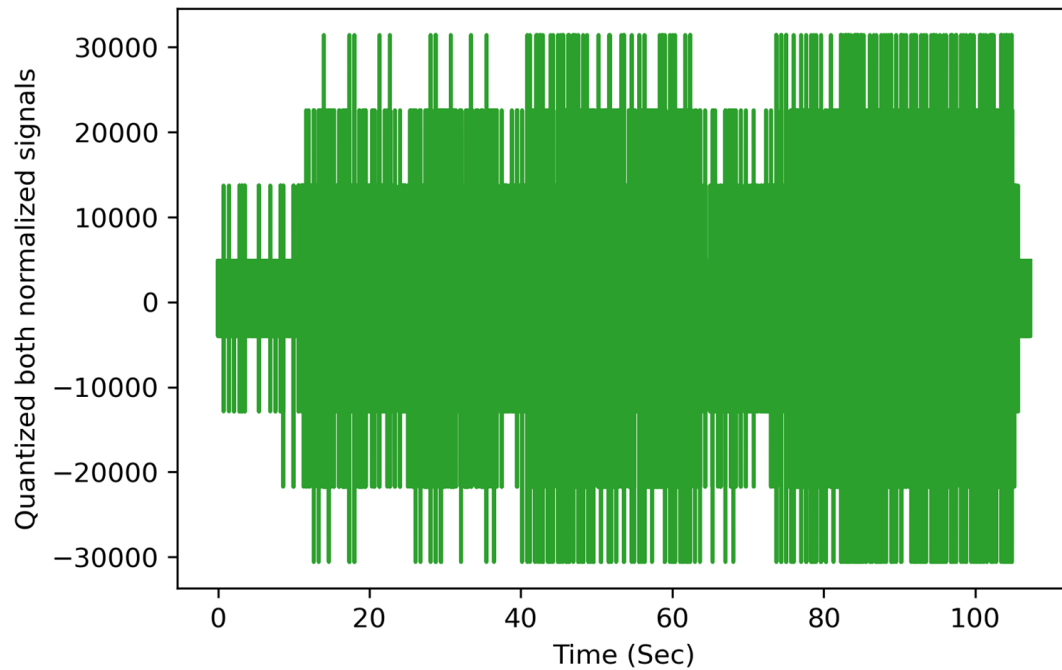


Fig: 3-bit Quantizer for audio signals

Discussion: Now we used the sampling signals for 3-bit quantization where we quantized the audio signals and got the normalized quantized audio signals.

Problem No:8

Code:

```
Lvl=8

Delta = np.zeros(n_channel)

Ps = np.zeros(n_channel)

Pq = np.zeros(n_channel)

SQNRdB_theory=np.zeros(n_channel)

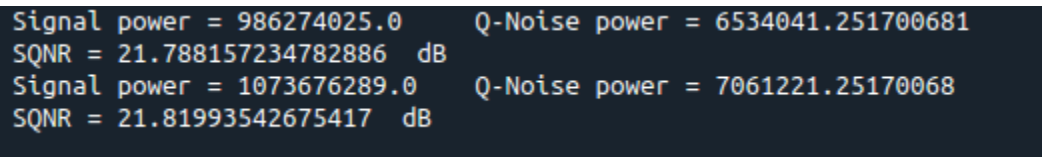
decode_signal= [];

for i in range(n_channel):
```

```

max_sigs[i] = max(audio_array[:,i])
min_sigs[i]=min(audio_array[:,i])
Delta[i]=(max_sigs[i]-min_sigs[i])/(Lvl-1)
Ps[i]=np.mean(max_sigs[i]**2)
Pq[i]=(Delta[i]**2)/12
SQNRdB_theory[i]=10*np.log10(Ps[i]/Pq[i])
decode_signal.append(Ps[i].decode("int16"))
print("Signal power =",Ps[i],"\t Q-Noise power =",Pq[i])
print("SQNR =",SQNRdB_theory[i]," dB")

```



```

Signal power = 986274025.0      Q-Noise power = 6534041.251700681
SQNR = 21.788157234782886  dB
Signal power = 1073676289.0    Q-Noise power = 7061221.25170068
SQNR = 21.81993542675417  dB

```

Fig:

bitstreams using hexadecimal symbols

The

Discussion: After quantization we generated PCM bitstreams on this quantized signals. We used hexadecimal symbols and got SQNR in db by PCM bitstreams. Then we implemented the decoded signals from the PCM bitstreams.

Problem No: 09

Code:

```

dec_audio_array = np.zeros((len(decode_signal[:,0]), n_channel), dtype='int16')
for i in range(n_channel):
    dec_audio_array[:,i] = audio_array_norm[:,i]*max_sigs[i]
write("Decoded_Audio.wav", sampling_rate, dec_audio_array)
playsound("Decoded_Audio.wav")

```

Discussion: Last we had to combined the quantized audio signals for both channel. Then we played the decoded audio signals which is different from given audio signals. Generated audio file is smooth than given audio file.

Conclusion:

In this project, we designed and simulated our project topic 'Audio file to Numpy array'. We tried figure out the audio file into Numpy array by using Python. All of these code are generated by python code and the simulation and plotted audio signals are got from python code. We modified the normal audio signals in numpy array and also used short time fft. Then we normalized and quantized the audio signals and found spectrum, spectrograms for each and every channels. After that we generated PCM bitstreams to get SNR value from quantized audio signals. Then we modified this quantized signals by decoded signals. And finally we got different result from quantized audio signals which totally unique from given Audio file. This was not so easy for me to doing this project as my laptop processor and ram is not good enough. Though many of troubles I completed and generated the results as much as I can.

Appendix:

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
@author: diti
```

```
"""
```

```
from scipy.io.wavfile import read, write
```

```
from matplotlib import pyplot as plt
```

```
from playsound import playsound
```

```
import numpy as np
```

```
audio_sig = read('test.wav')
```

```

playsound(audio_sig)

n_channel = len(audio_sig)

sampling_rate = audio_sig[0]

audio_array = np.array(audio_sig[1])

# Normalizing the Audio Array

audio_array_norm = np.zeros((len(audio_array[:,0]), n_channel)) # Initialize Norm Array

max_sigs = np.zeros(n_channel) # Max value of each channel

for i in range(n_channel):

    max_sigs[i] = max(audio_array[:,i])

    audio_array_norm[:,i] = audio_array[:,i]/max_sigs[i]

Duration = len(audio_array[:,0])/sampling_rate

Time = np.linspace(0, Duration, len(audio_array[:,0]))

for i in range(n_channel):

    plt.figure(i+1, dpi=300)

    plt.plot(Time, audio_array_norm[:,i])

    plt.xlabel("Time (Sec)")

    plt.ylabel("Normalized Audio of Channel-" + str(i+1))


    audio_array_norm = np.zeros((len(audio_array[:,0]), n_channel))

max_sigs = np.zeros(n_channel)

for i in range(n_channel):

    max_sigs[i] = max(audio_array[:,i])

    audio_array_norm[:,i] = audio_array[:,i]/max_sigs[i]

for i in range(n_channel):

    plt.figure(i+1)

```

```
plt.plot(Time, audio_array_norm[:,i])  
plt.ylabel("Normalized Audio of Channel -" + str(i+1))
```

```
from numpy.fft import fft  
h=[]  
for i in range(n_channel):  
    x=fft(audio_array_norm[:,i])  
    h.append(x)  
for i in range(n_channel):  
    plt.figure(i+1)  
    plt.plot(Time, h[i])
```

```
cs=0  
for i in range(n_channel):  
    cs=cs+h[i]  
maxcs=max(cs)  
norcs=cs/maxcs  
plt.figure(1)  
plt.plot(Time, cs)
```

```
import librosa  
import librosa.display  
fs = 10e3  
T= float(1/fs)
```



```

del_time = 10*T

scale, sr = librosa.load('test.wav', duration=del_time)

Frame_size=2048

Hop_size=512

S_scale = librosa.stft(scale,n_fft=Frame_size,hop_length=Hop_size)

Y_scale= np.abs(S_scale)**2

def plot_spectrogram(Y,sr,hop_length,y_axis="linear"):

    plt.figure(figsize=(10,10))

    librosa.display.specshow(Y,sr=sr,hop_length=hop_length,x_axis="time",y_axis=y_axis)

Y_log_scale = librosa.power_to_db(Y_scale)

plot_spectrogram(Y_log_scale, sr, Hop_size)

```

```

scale, sr = librosa.load('test.wav')

Frame_size=2048

Hop_size=512

S_scale = librosa.stft(scale,n_fft=Frame_size,hop_length=Hop_size)

Y_scale= np.abs(S_scale)**2

def plot_spectrogram(Y,sr,hop_length,y_axis="linear"):

    plt.figure(figsize=(10,10))

    librosa.display.specshow(Y,sr=sr,hop_length=hop_length,x_axis="time",y_axis=y_axis)

Y_log_scale = librosa.power_to_db(Y_scale)

plot_spectrogram(Y_log_scale, sr, Hop_size)

```

```

min_sigs = np.zeros(n_channel)

for i in range(n_channel):

```

```
max_sigs[i] = max(audio_array[:,i])
```

```
min_sigs[i]=min(audio_array[:,i])
```

```
def quantizer(x, qmin=-1, qmax=1, level=8):
```

```
    x_normalize = (x-qmin) * (level-1) / (qmax-qmin)
```

```
    x_normalize[x_normalize > level - 1] = level - 1
```

```
    x_normalize[x_normalize < 0] = 0
```

```
    x_normalize_quant = np.around(x_normalize)
```

```
    x_quant = (x_normalize_quant) * (qmax-qmin) / (level-1) + qmin
```

```
    return x_quant
```

```
x_quan = np.zeros((len(audio_array[:,0]), n_channel))
```

```
for i in range(n_channel):
```

```
    x_quan[:,i]=quantizer(audio_array[:,i],min_sigs[i],max_sigs[i],level=8)
```

```
for i in range(n_channel):
```

```
    plt.figure(i+1)
```

```
    plt.plot(Time,x_quan[:,i] )
```

```
    plt.ylabel("Quantized both normalized signals")
```

```
Lvl=8
```

```
Delta = np.zeros(n_channel)
```

```
Ps = np.zeros(n_channel)
```

```

Pq = np.zeros(n_channel)

SQNRdB_theory=np.zeros(n_channel)

decode_signal= [];

for i in range(n_channel):

    max_sigs[i] = max(audio_array[:,i])

    min_sigs[i]=min(audio_array[:,i])


    Delta[i]=(max_sigs[i]-min_sigs[i])/(Lvl-1)

    Ps[i]=np.mean(max_sigs[i]**2)

    Pq[i]=(Delta[i]**2)/12

    SQNRdB_theory[i]=10*np.log10(Ps[i]/Pq[i])

    decode_signal.append(Ps[i].decode("int16"))

    print("Signal power =",Ps[i],"t Q-Noise power =",Pq[i])

    print("SQNR =",SQNRdB_theory[i]," dB")


dec_audio_array = np.zeros((len(decode_signal[:,0]), n_channel), dtype='int16')

for i in range(n_channel):

    dec_audio_array[:,i] = audio_array_norm[:,i]*max_sigs[i]

write("Decoded_Audio.wav", sampling_rate, dec_audio_array)

playsound("Decoded_Audio.wav")

```