Audio Processing

Exercise 1

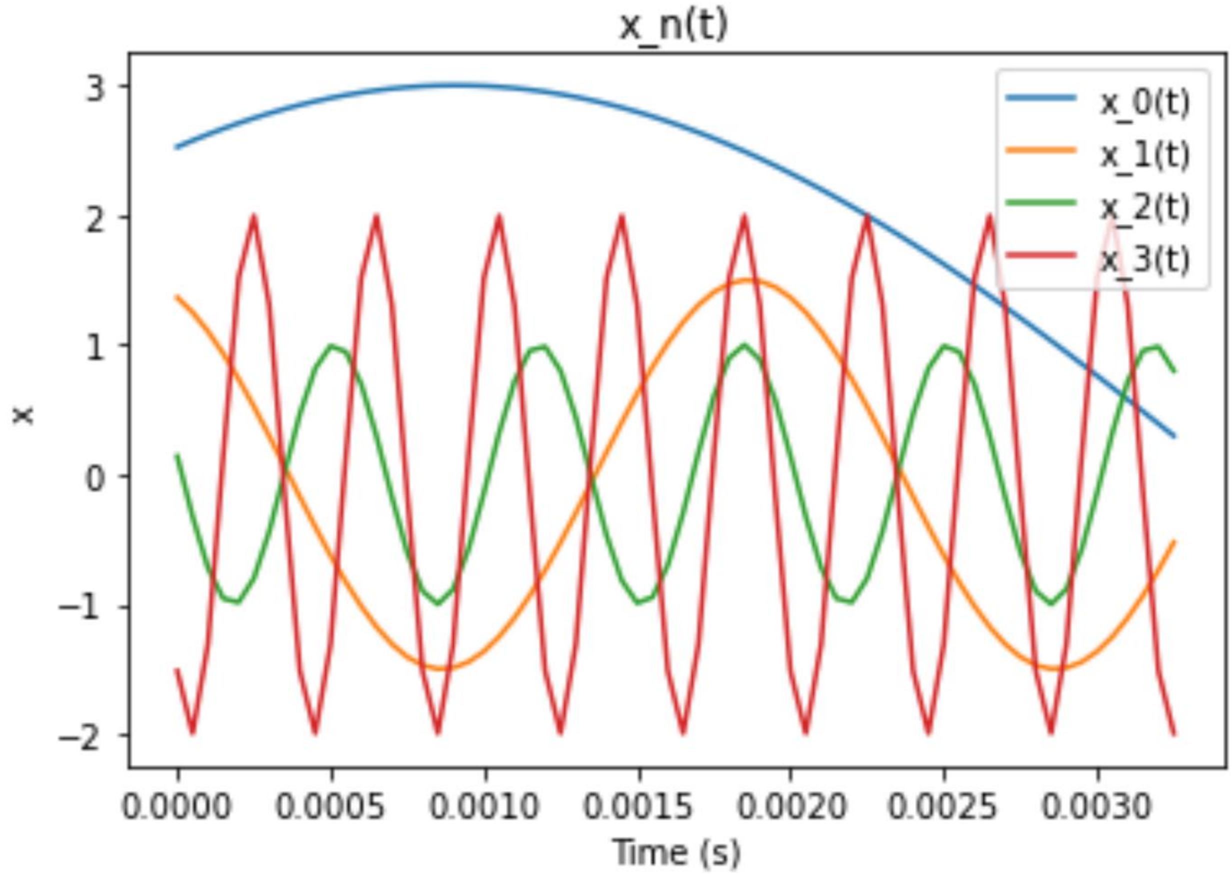
Anh Huy Bui

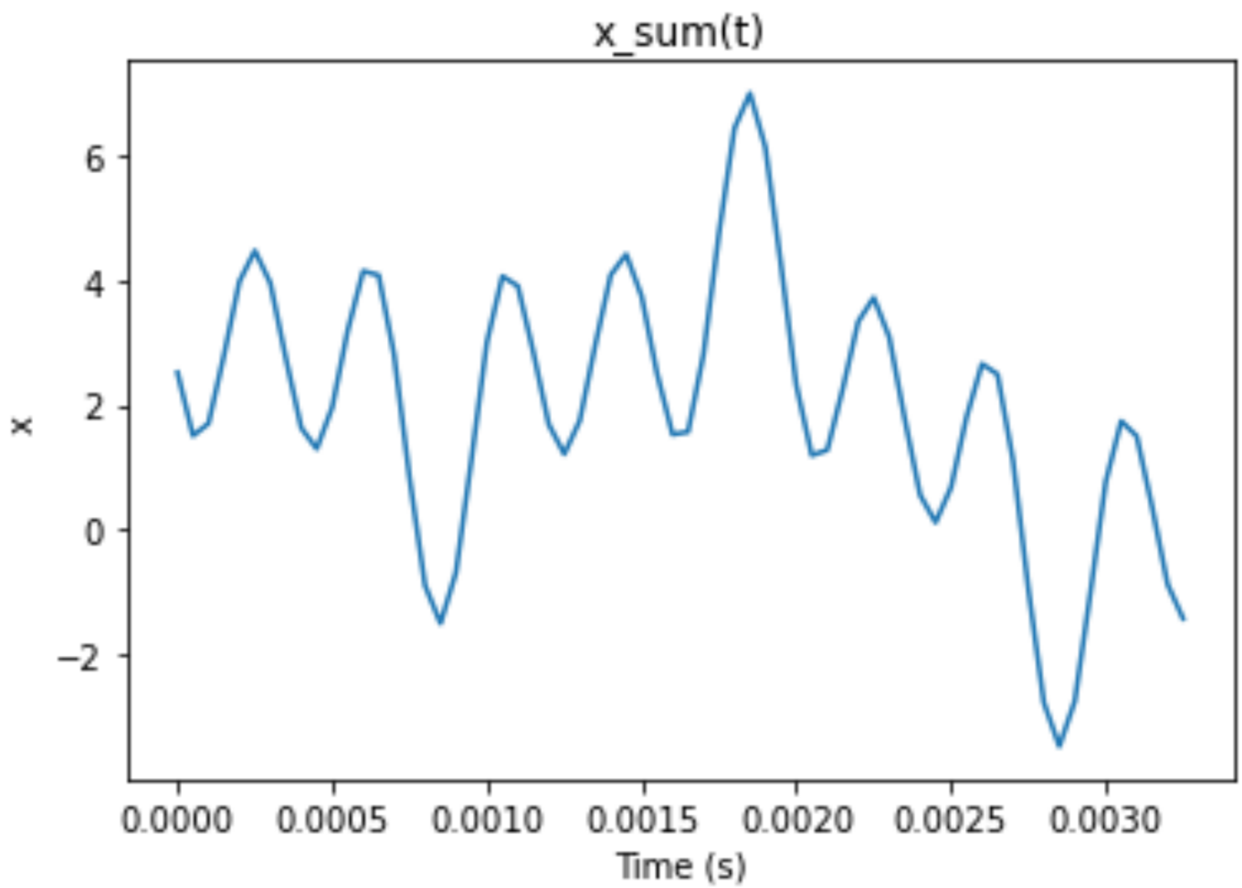
293257

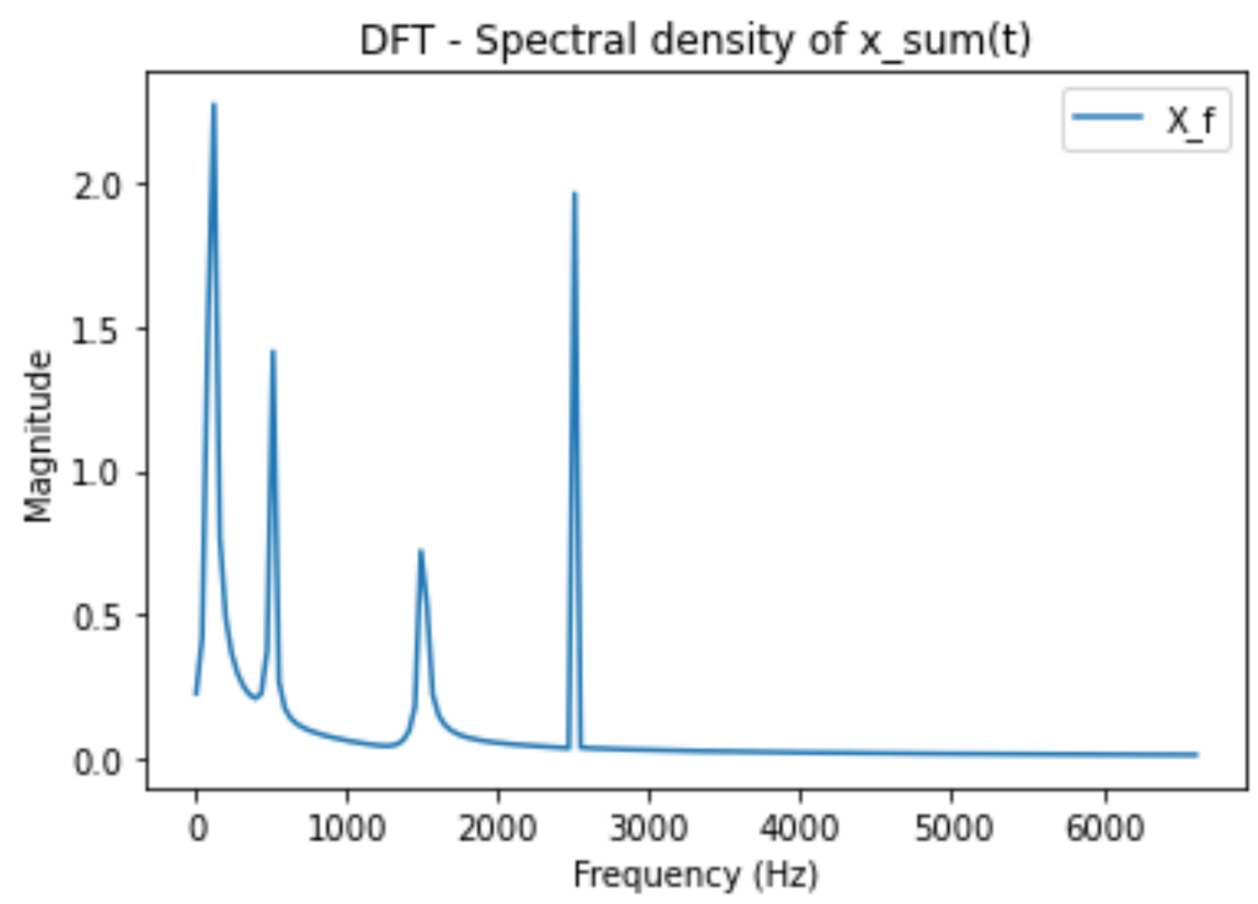
Problem 1:

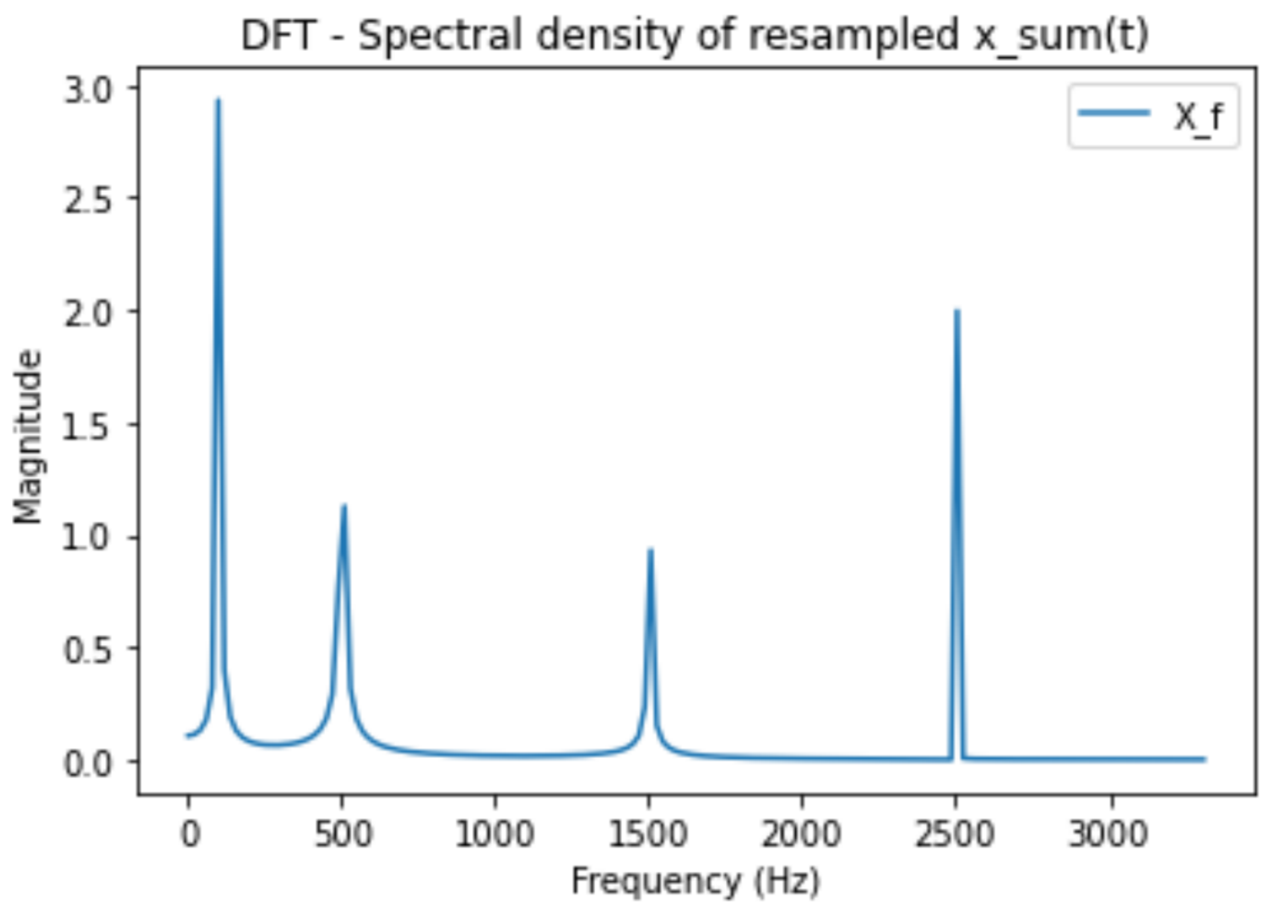
```python
#!/usr/bin/python
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
from scipy.io import wavfile
from scipy.fftpack import fft


def main():
    # - Define sampling parameters
    # - Highest frequency in this exercise is 2500 so
    # Fs must be at least 5000 (Nyquist Theorem), I chose 20000
    # to preserve waveform
    Fs = 20000
    SimulationTime = 3
    NumOfSample = int(SimulationTime*Fs)

    # Define 4 sine signal with given frequency
    # and different chosen amplitudes and phases
    t = np.linspace(0, SimulationTime,NumOfSample)
    x = [ 3*np.sin(2 * np.pi * 100 * t +1),
          1.5*np.sin(2 * np.pi * 500 * t +2),
          np.sin(2 * np.pi * 1500 * t +3),
          2*np.sin(2 * np.pi * 2500 * t +4)]

    # Define plot range to have a better observation of waveform
    plot_range = int(Fs/300)
    plt.figure(1)
    for i in range(0,len(x)):
        plt.plot(t[0:plot_range],x[i][0:plot_range], label = 'x_' + str(i) + '(t)')
    plt.title('x_n(t)')
    plt.xlabel('Time (s)')
    plt.ylabel('x')
    plt.legend()


    # Calculate sum of 4 signals
    x_sum = 0
    for i in range(0,len(x)):
        x_sum += x[i]

    # Plot sum signal
    plt.figure(2)
    plt.plot(t[0:plot_range],x_sum[0:plot_range])
    plt.title('x_sum(t)')
    plt.xlabel('Time (s)')
    plt.ylabel('x')

    # Write sum signal to file
    wavfile.write('Sine.wav', Fs, x_sum)


    # Define DFT-point
    N_point = 512
```

```
    # Fast Fourier Transform summed signal to get Spectral Density of x_sum
    # fftshift to plot both negative and positive part
    ff = np.linspace(0, int(Fs), N_point)
    xf = fft(x_sum,N_point)

    # Plot DFT in suitable range for better observation

    plt.figure(3)
    plt.plot(ff[0: int(N_point/3)], 2/N_point * np.abs(xf)[0: int(N_point/3)],
             label ='X_f')
    plt.title('DFT - Spectral density of x_sum(t)')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Magnitude')
    plt.legend()


    ################################
    # Resample x with 2 times less samples
    resampled_x_sum = signal.resample(x_sum,int(NumOfSample/2))

    # Fast Fourier Transform resampled signal to get Spectral Density
    # fftshift to plot both negative and positive part
    # Because number of samples is reduced by 2 times, so is Fs
    ff = np.linspace(0, int(Fs/2), N_point)
    xf = fft(resampled_x_sum,N_point)

    # Plot DFT in suitable range for better observation

    plt.figure(4)
    plt.plot(ff[0: int(N_point/3)], 2/N_point * np.abs(xf)[0: int(N_point/3)],
             label ='X_f')
    plt.title('DFT - Spectral density of resampled x_sum(t)')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Magnitude')
    plt.legend()



    plt.show()




if __name__== "__main__":
    main()
```


x_n(t)
x_0(t)
x_1(t)
x_2(t)
x_3(t)
x
Time (s)


x_sum(t)
x
Time (s)


DFT - Spectral density of x_sum(t)
X_f
Magnitude
Frequency (Hz)
0.0
0.5
1.0
1.5
2.0
0
1000
2000
3000
4000
5000
6000


DFT - Spectral density of resampled x_sum(t)
X_f
Magnitude
Frequency (Hz)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
500
1000
1500
2000
2500
3000

Comment:

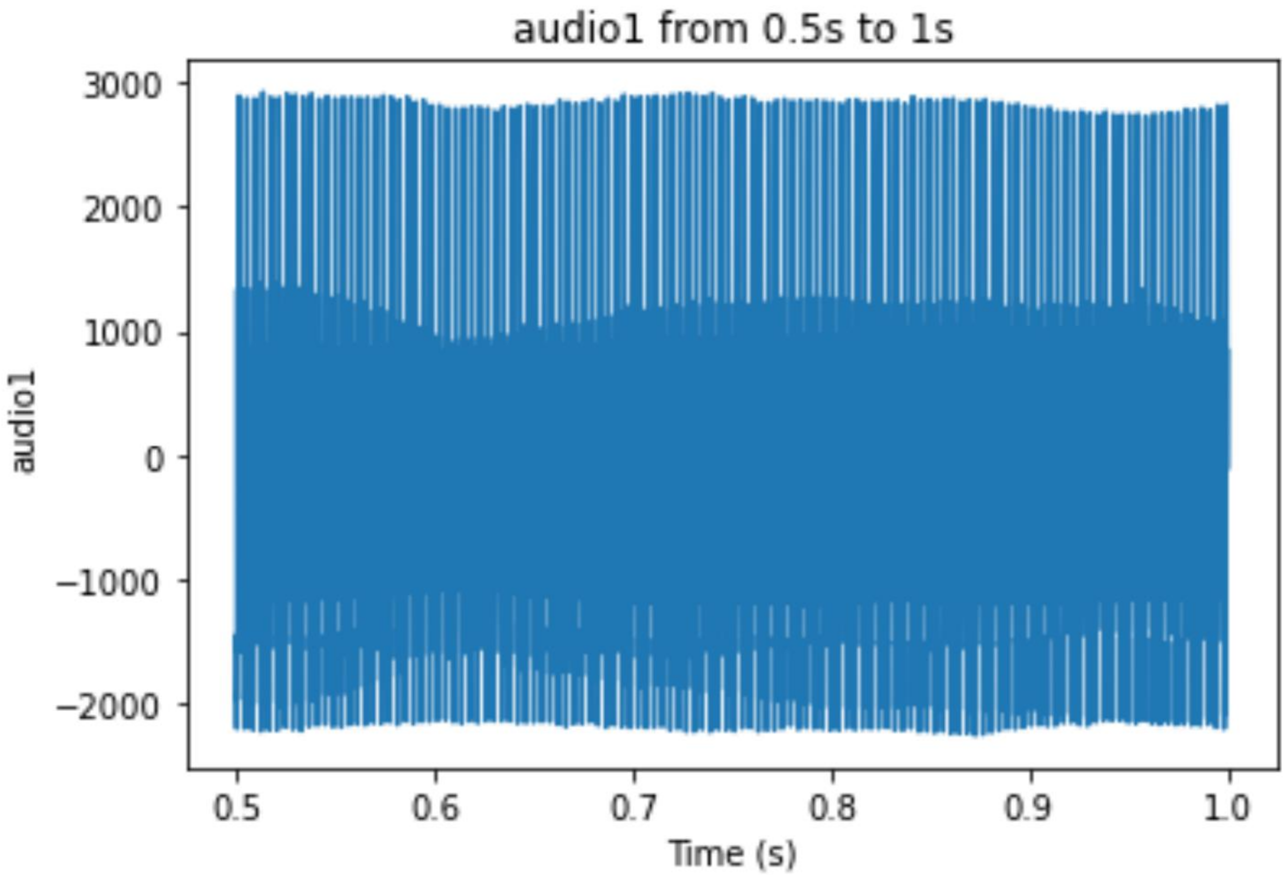
The highest frequency in DFT plot is the sampling frequency Fs and the lowest frequency is Fs/N. So with a fixed number of N, higher Fs results in lower frequency resolution.

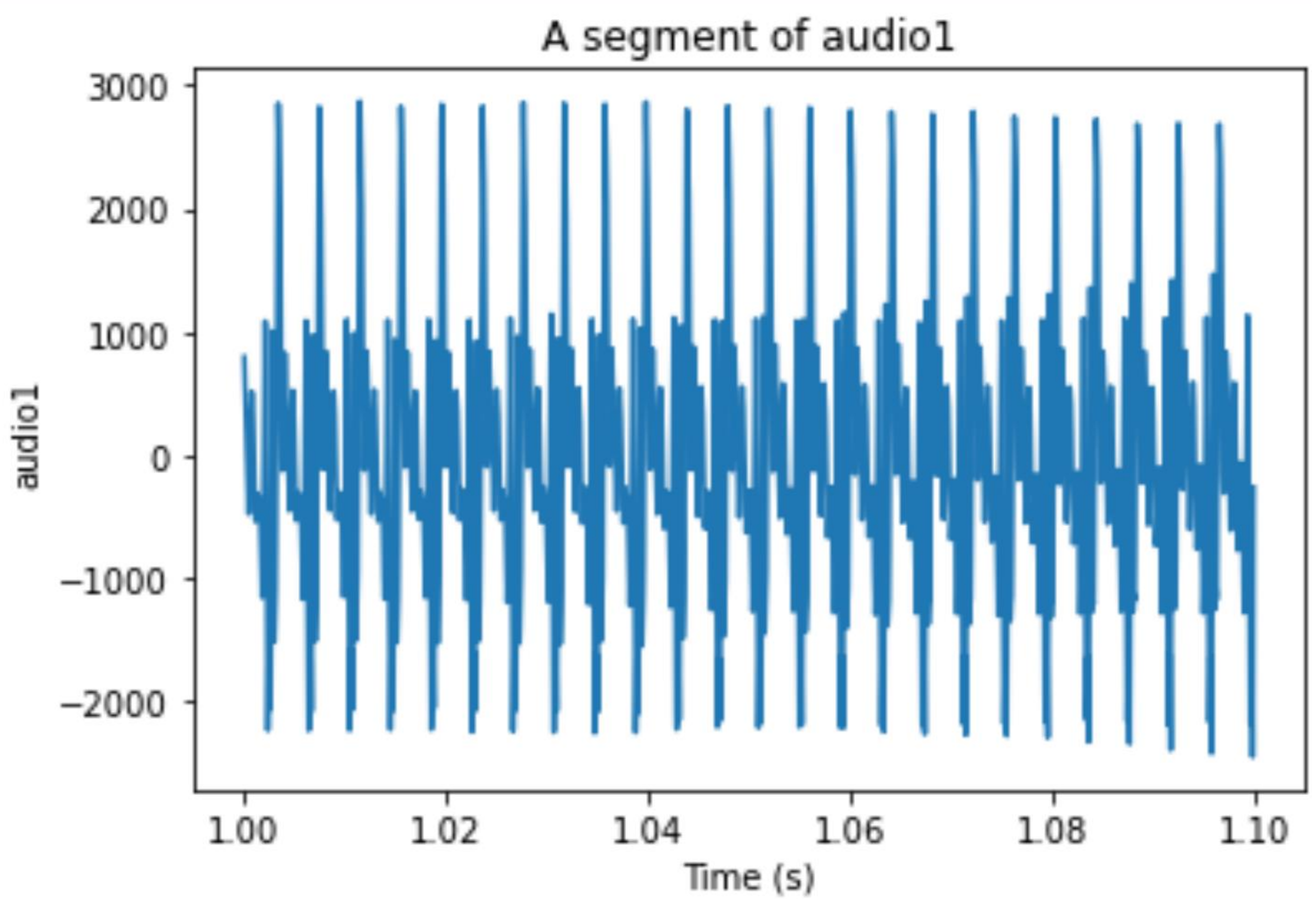
In resampled version, DFT plot is same as original one with sampled with Fs' = Fs/2. As can be seen from 2 plots, the resampled one are more precise, especially in range about 0-1000Hz. It can be because of above reason, the resampled one has lower Fs so higher frequency resolution.

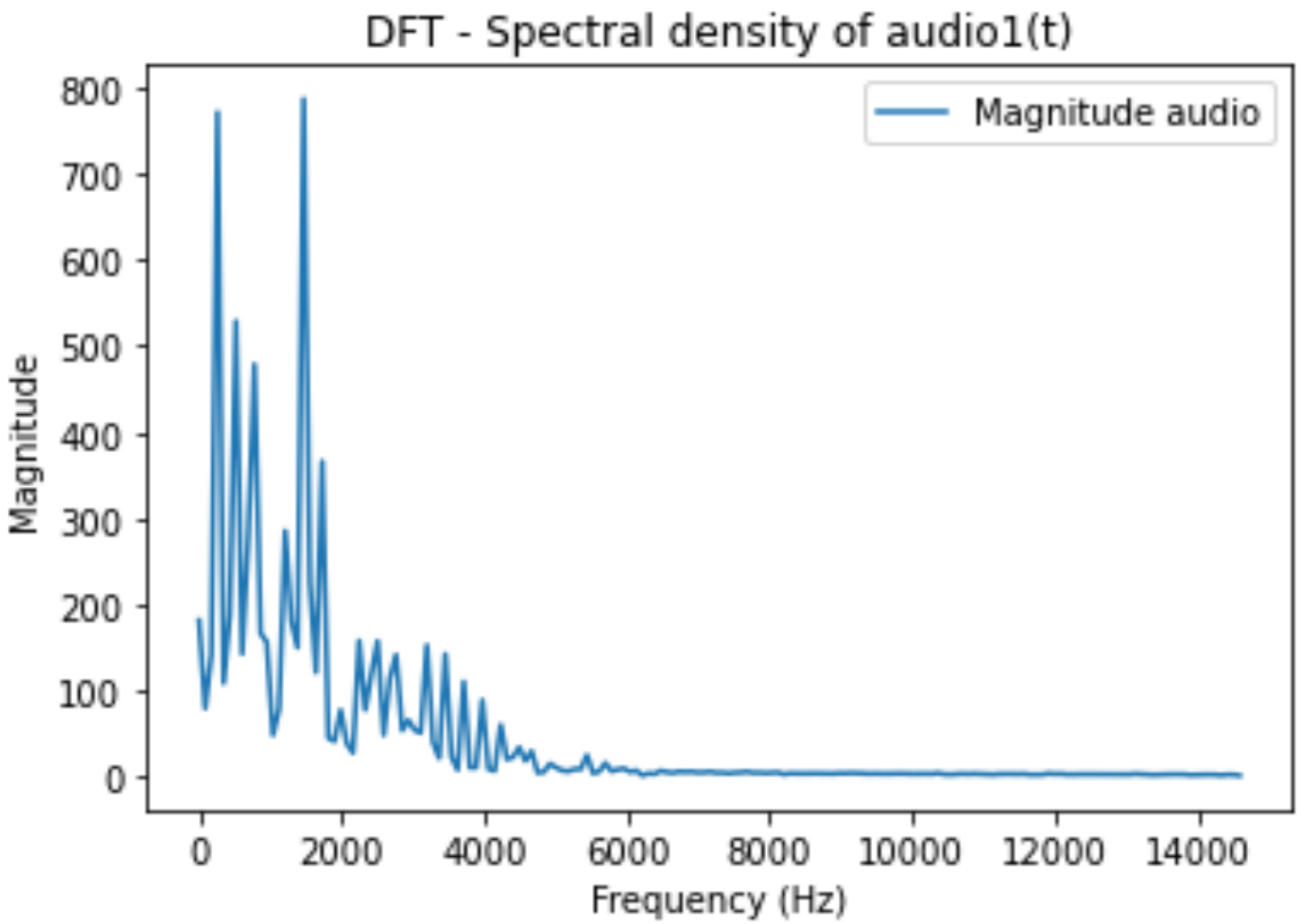
Problem 2:

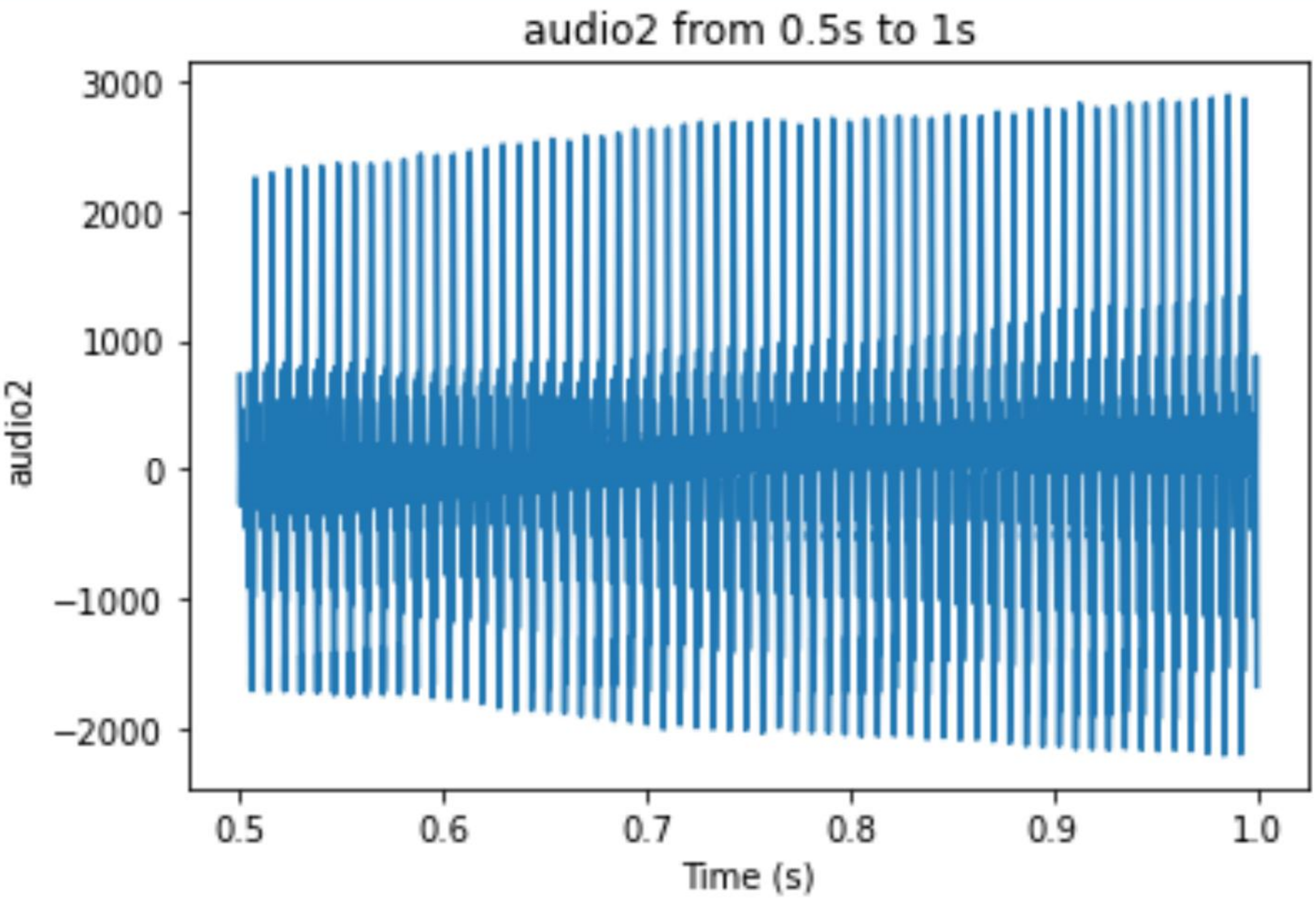
```python
#!/usr/bin/python
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
from scipy.fftpack import fft

# This number is used to count plot windows
figure_index = 0

# Define DFT-point
N_point = 512

# Define segment period (in second)
period = 0.1

def DFTandPlot(Fs, audio, audio_index):
    global figure_index
    global N_point
    global period
    NumOfSample = len(audio)
    # Loop from 1s to the end, each iteration last "period (s)"
    # NumOfSample/Fs1 return the time in seconds;
    #   subtract 1 as it is the first second.
    # Div by "period" to transform into multiple of "period"
    for i in range(0,int((NumOfSample/Fs - 1)/period)):
        # Define a time step which last period start from second 1
        # Extract a segment from audio
        step_time =  np.linspace(1+(i*period),1+(i+1)*period,int(Fs*period))
        step_amp = audio[(int(i*period*Fs) +Fs): (int((i+1)*period*Fs) +Fs)]

        if (i < 1):
            # Plot the segment
            figure_index+=1
            plt.figure(figure_index)
            plt.plot(step_time,step_amp)
            plt.title('A segment of audio' + audio_index)
            plt.xlabel('Time (s)')
            plt.ylabel('audio1')

            # Calculate FFT with the segment
            audio_f = np.linspace(0, int(Fs), N_point)
            AUDIOinF = fft(step_amp,N_point)
            # Plot FFT
            figure_index+=1
            plt.figure(figure_index)
            plt.plot(audio_f[0: int(N_point/3)], 2/N_point * np.abs(AUDIOinF)[0: int(N_point/3)],
                     label ='Magnitude audio')
            plt.title('DFT - Spectral density of audio' + audio_index +'(t)')
            plt.xlabel('Frequency (Hz)')
            plt.ylabel('Magnitude')
            plt.legend()
```

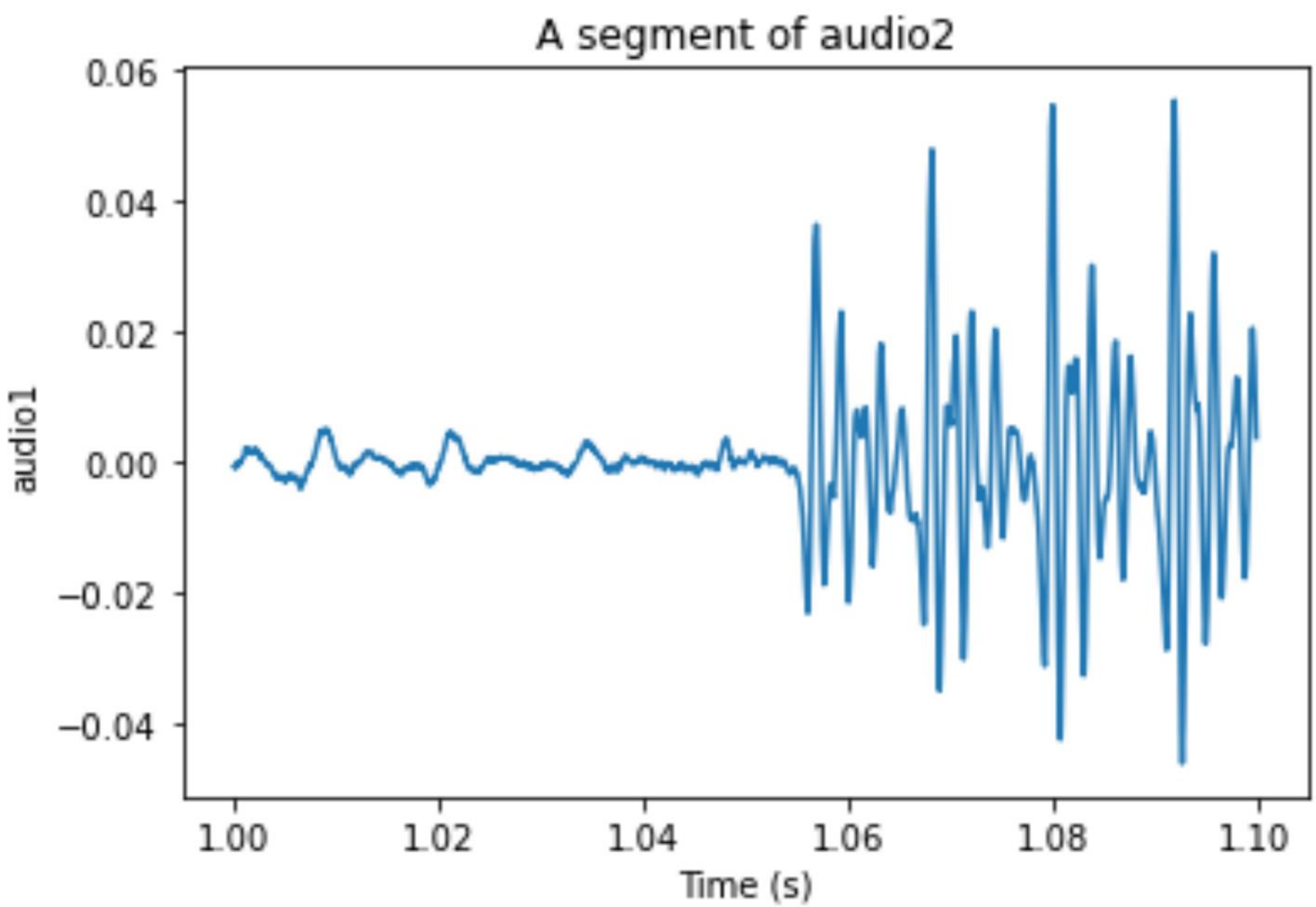
```python

def main():
    global figure_index


    # Load audio1.wav with sample rate and signal
    [Fs1, audio1] = wavfile.read('audio1.wav')

    # Define time range from 0.5s to 1s
    # There are "Fs" samples in 1s, so for 0.5s, number of samples is Fs/2
    audio1_t = np.linspace(1/2, 1, int(Fs1/2))
    audio1_x = audio1[int(Fs1/2): int(Fs1)]

    # Plot the audio1 in range of 0.5s to 1s
    figure_index+=1
    plt.figure(figure_index)
    plt.plot(audio1_t,audio1_x)
    plt.title('audio1 from 0.5s to 1s')
    plt.xlabel('Time (s)')
    plt.ylabel('audio1')
    # Calculate DFT for each segment in audio1
    # and plot the 1st segment
    DFTandPlot(Fs1, audio1,'1')

    # Load audio2.wav with sample rate and signal
    [Fs2, audio2] = wavfile.read('audio2.wav')

    # Define time range from 0.5s to 1s
    # There are "Fs" samples in 1s, so for 0.5s, number of samples is Fs/2
    audio2_t = np.linspace(1/2, 1, int(Fs2/2))
    audio2_x = audio1[int(Fs2/2): int(Fs2)]

    # Plot the audio2 in range of 0.5s to 1s
    figure_index+=1
    plt.figure(figure_index)
    plt.plot(audio2_t,audio2_x)
    plt.title('audio2 from 0.5s to 1s')
    plt.xlabel('Time (s)')
    plt.ylabel('audio2')
    # Calculate DFT for each segment in audio2
    # and plot the 1st segment
    DFTandPlot(Fs2, audio2,'2')

    plt.show()



if __name__== "__main__":
    main()
```

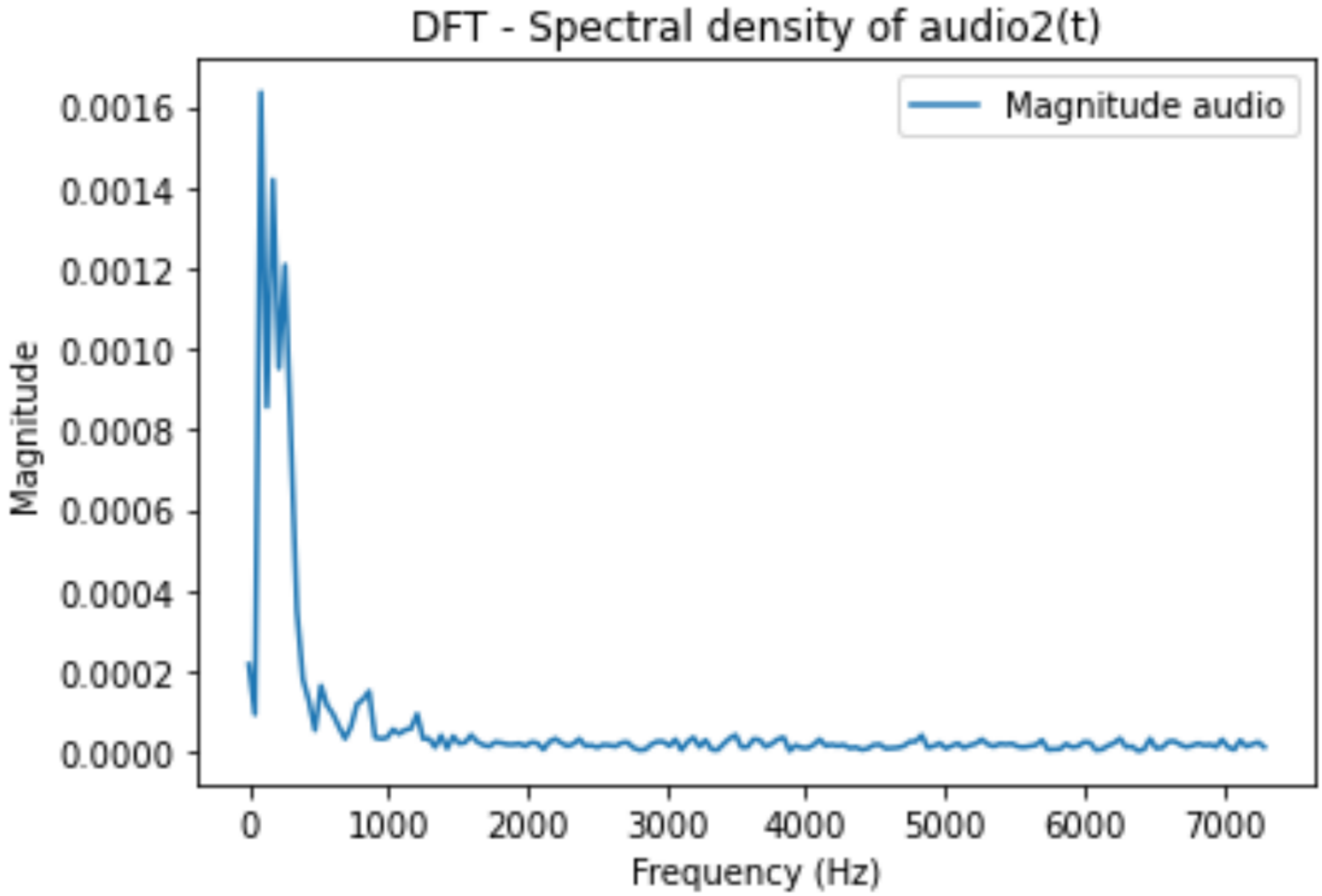

audio1 from 0.5s to 1s
3000
2000
1000
0
−1000
−2000
audio1
0.5
0.6
0.7
0.8
0.9
1.0
Time (s)


A segment of audio1
3000
2000
1000
0
−1000
−2000
audio1
1.00
1.02
1.04
1.06
1.08
1.10
Time (s)


DFT - Spectral density of audio1(t)
Magnitude audio
Magnitude
Frequency (Hz)
800
700
600
500
400
300
200
100
0
0
2000
4000
6000
8000
10000
12000
14000


audio2 from 0.5s to 1s
audio2
Time (s)
3000
2000
1000
0
−1000
−2000
0.5
0.6
0.7
0.8
0.9
1.0


A segment of audio2
audio1
Time (s)
0.06
0.04
0.02
0.00
−0.02
−0.04
1.00
1.02
1.04
1.06
1.08
1.10



Comment:

The DFT plots of both signals indicate that the 2 audios consist of multiple sinusoid waves with different frequencies within. This is described by the fluctuation of DFT plot throughout frequency range, despite the highest around 0 – 500 Hz. On the other hand, sinusoids' DFT just shows that there are only 4 signals clearly seen in the sine.wav