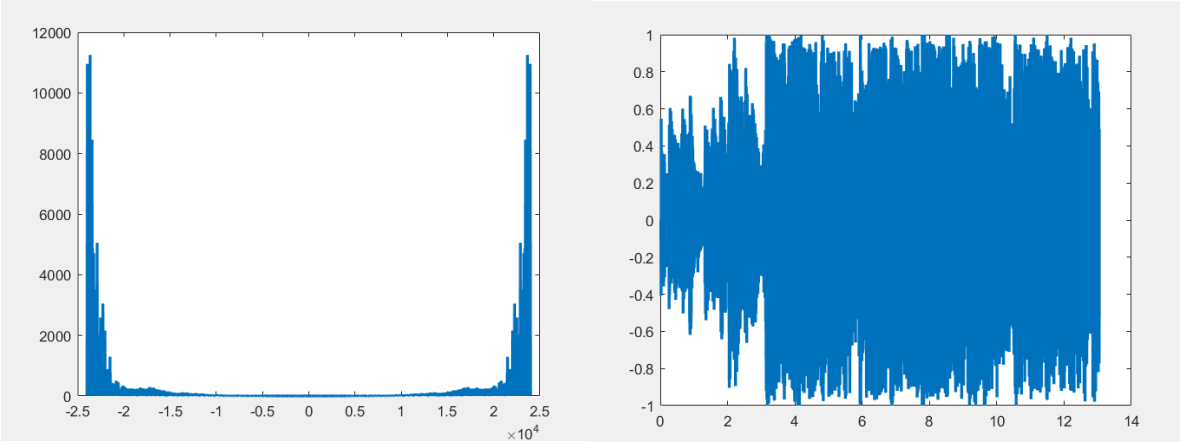
1. Read the music (2-channel stereo music) using MATLAB **audioread()**. For simplicity, I first convert the 2-channel stereo music into mono-channel one by selecting the 1st channel (i.e., the first column) of the music. Plot the signal in time domain and its frequency-domain representation (i.e., magnitude spectrum) using MATLAB **fft()**. Remember to plot with the correct time and frequency axes. Can you find the signals representing drum beats in time or frequency domain? If you can, explain why the signals of drum beats present in such a form.

```
X = fft(x);
mag_X = abs(X);

T = 1/Fs;
t_axis = t;
Npoint = length(x);
dF = Fs/Npoint;
f_axis = ((1:1:Npoint)-(Npoint+1)/2)*dF;

figure
plot(f_axis, mag_X);
figure
plot(t_axis, x,'linewidth',2);
```

Plot:



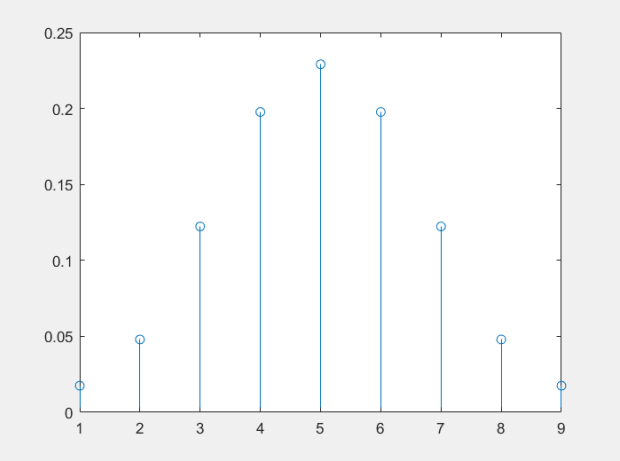

Drum beats locate at the low pitch area.

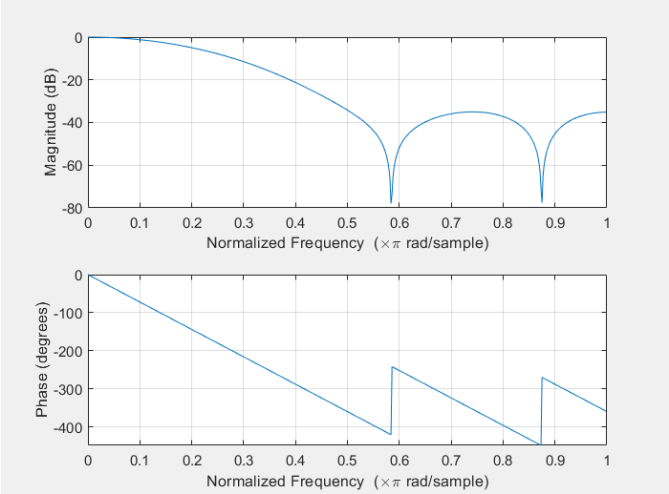
2. Filter the music. Apply the designed low-pass filter and high-pass filter to the music and listen to find the differences.

(a) Use the MATLAB function **fir1()**to obtain the impulse response **h** of a low-pass (or a high-pass) filter. Change the filter order to 8, 16, 32, 64, 128, and 256, and then briefly describe the difference between their frequency-domain (by MATLAB function **freqz()**) and time-domain responses (by MATLAB function **stem()**). Plot the frequency responses and impulse responses to find the differences. From your observation, please also tell the idea how the **fir1()** per forms the low-pass filter and the high pass filter design (Hint: compare the ideal low pass filter and the filter designed by **fir1()**). Remember to check whether the designed filter work by comparing the signal spectra before and after filtering.

(Low-pass) 8:

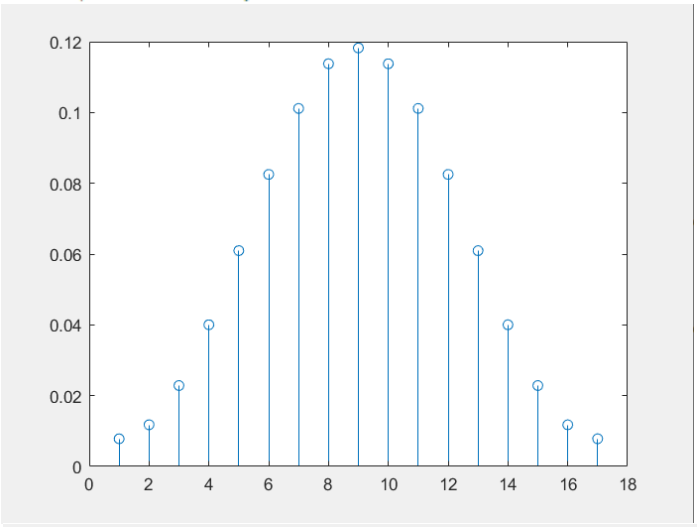
```
Fcut = 1000; % Hz, cut off frequency (you may try 4 kHz cutoff frequency for LPF and
FilterOrder = 8; % filter order
flags.lowpass = 1; % 1: low pass filter
% low-pass and high-pass filter design using firl()
% perform filtering using conv()
if flags.lowpass,
    h = firl(FilterOrder, Fcut/(Fs/2));   % help or doc firl(), frequency normalizat
else
    h = firl(FilterOrder, Fcut/(Fs/2), 'high'); % help firl(), frequency normalizati
end

h = h.'; % convert to column vector, because x is a column vector
y = conv(x, h,'same'); % filtering by convolution, why 'same'? remove the group dela
sound(y, Fs);
audiowrite('FilteredMusic.ogg',y,Fs);

figure
stem(h); % Plot the impulse response.
figure
freqz(h,1); % Plot the frequency response - log magnitude response and phase respons
```

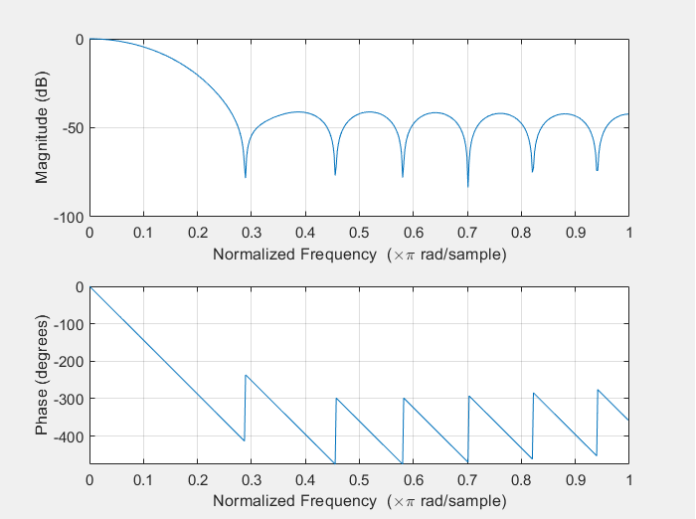




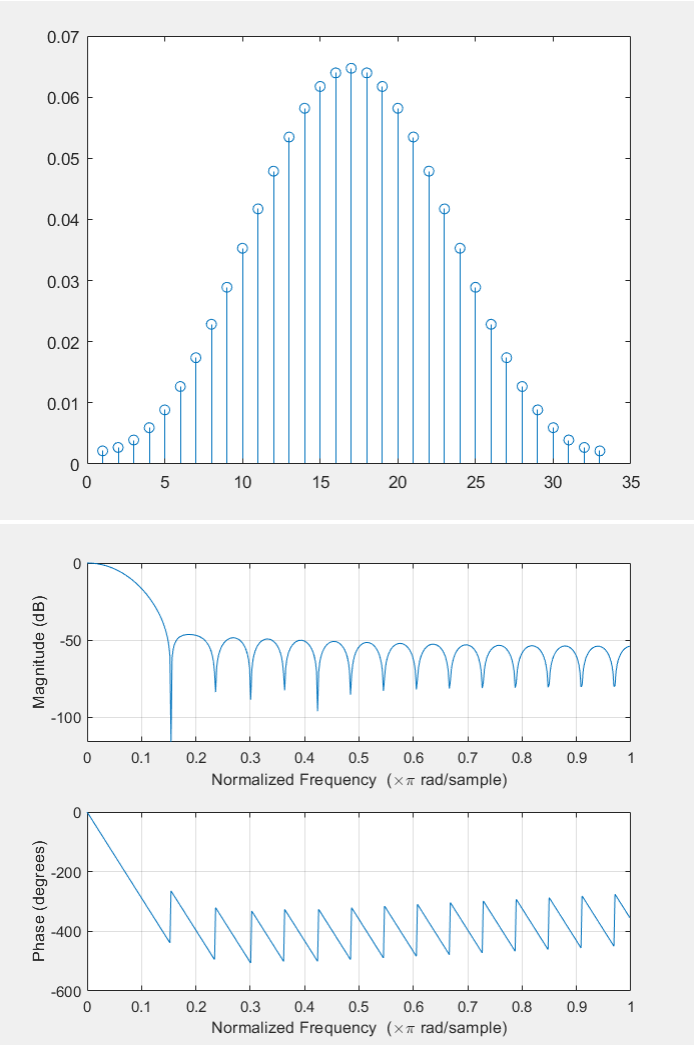

16:

```
Fcut = 1000; % Hz, cut off frequency (you may try 4 kHz cutoff frequenc
FilterOrder = 16; % filter order
flags.lowpass = 1; % 1: low pass filter
```

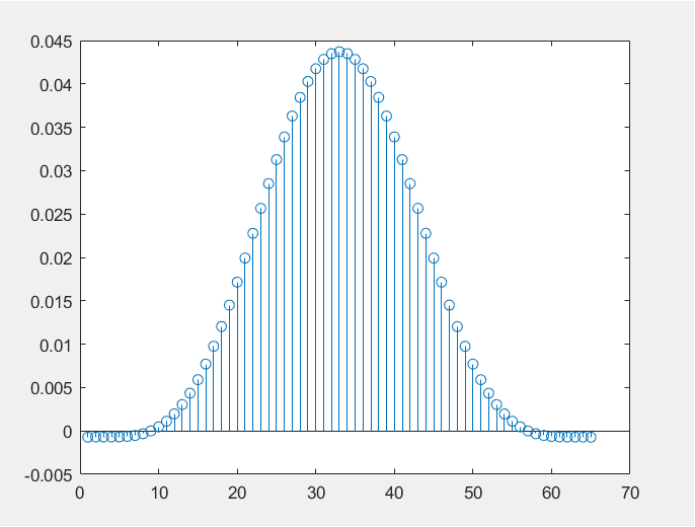


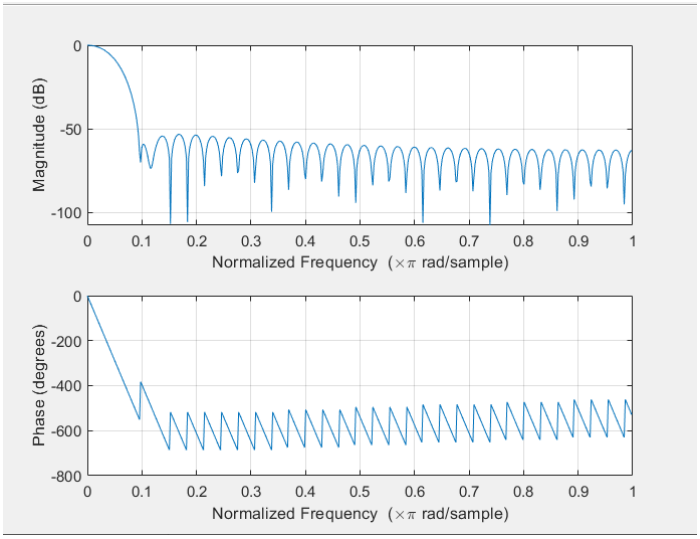


32:

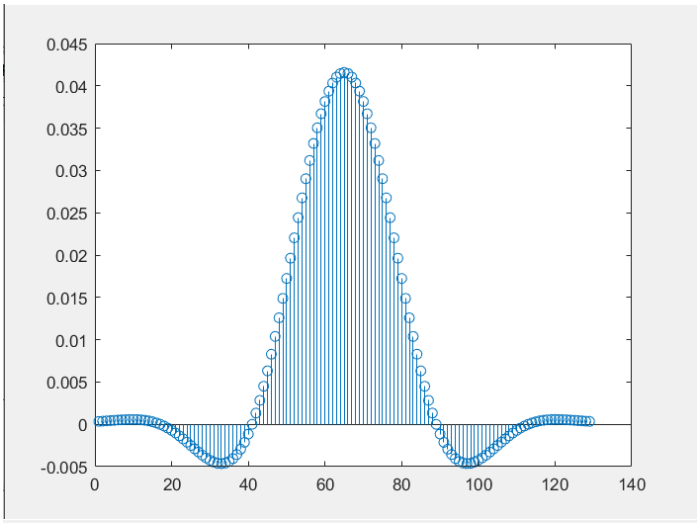

0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0 5 10 15 20 25 30 35
Magnitude (dB)
0
-50
-100
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)
Phase (degrees)
0
-200
-400
-600
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)

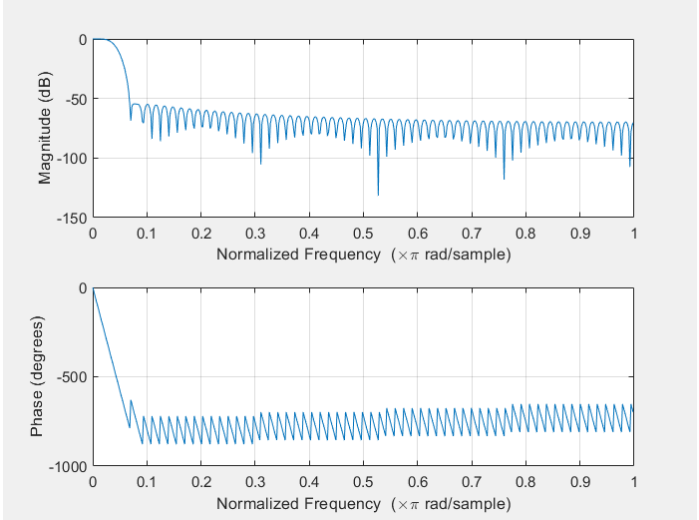
64:


0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-0.005
0 10 20 30 40 50 60 70

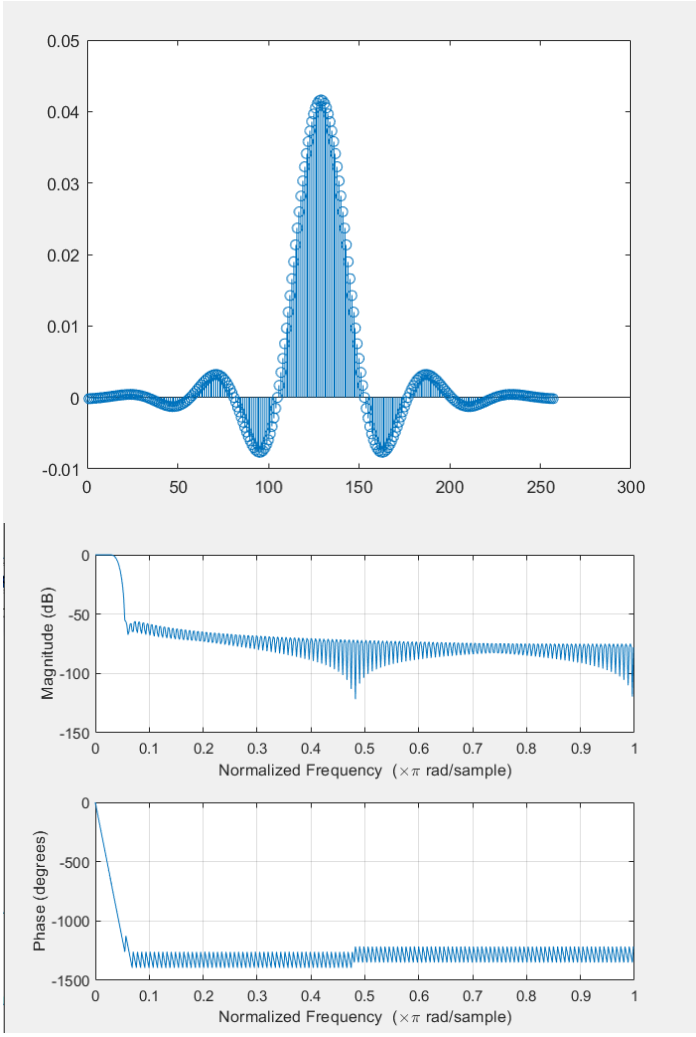

Magnitude (dB)
0
-50
-100
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)
Phase (degrees)
0
-200
-400
-600
-800
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)

128:


0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-0.005
0 20 40 60 80 100 120 140


Magnitude (dB)
0
-50
-100
-150
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)
Phase (degrees)
0
-500
-1000
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)

256:




The larger the filter order grows, the sharper the cut-off frequency gets (in frequency domain), while the sinc function grows wider and wider (in time domain).

Furthermore, the area is always 1, in other words, the stem would add up to 1 at any time.

Comparing fir1() with a multiply of a hamming window and a sinc function, they are almost the same except for the scalar.

(b) Listen to your results carefully and answer the following questions. How differently do the low-pass and high-pass versions sound? Does the filter length affect the sound quality of the output? How and why? Note that the FIR filter length is the support length of the impulse response and is equal to the filter order + 1. Note that you had better vary the cut-off frequency of your filter in order to get better feeling of the processed music.

(1)

high-pass:

```
Fcut = 1000; % Hz, cut off frequency (you may try 4 kHz cutoff frequency for LPF and 0.5 kHz cutoff for HPF
FilterOrder = 256; % filter order
flags.lowpass = 0; % 1: low pass filter
% low-pass and high-pass filter design using firl()
% perform filtering using conv()
if flags.lowpass,
    h = firl(FilterOrder, Fcut/(Fs/2));   % help or doc firl(), frequency normalization is done by normalizatio
else
    h = firl(FilterOrder, Fcut/(Fs/2), 'high'); % help firl(), frequency normalization is done by normalization
end

h = h.'; % convert to column vector, because x is a column vector
y = conv(x, h,'same'); % filtering by convolution, why 'same'? remove the group delay introduced by the FIR LPF
sound(y, Fs);
audiowrite('FilteredMusic.ogg',y,Fs);
```

low-pass:

```
Fcut = 1000; % Hz, cut off frequency (you may try 4 kHz cutoff frequency for LPF and 0.5 kHz cutoff for HPF
FilterOrder = 256; % filter order
flags.lowpass = 1; % 1: low pass filter
% low-pass and high-pass filter design using firl()
% perform filtering using conv()
if flags.lowpass,
    h = firl(FilterOrder, Fcut/(Fs/2));   % help or doc firl(), frequency normalization is done by normaliza
else
    h = firl(FilterOrder, Fcut/(Fs/2), 'high'); % help firl(), frequency normalization is done by normalizat
end

h = h.'; % convert to column vector, because x is a column vector
y = conv(x, h,'same'); % filtering by convolution, why 'same'? remove the group delay introduced by the FIR
sound(y, Fs);
audiowrite('FilteredMusic.ogg',y,Fs);
```

We can hear music mainly composed of different pitch, low-pass is composed of low-frequency sound when high-pass is opposite of, no matter for the vocals, piano or any instrument.

Furthermore, high-pass still have a little delay, making it chaos.

(2)

```
Fcut = 1000; % Hz, cut off frequency (you may try 4 kHz cutoff frequency for LPF and
FilterOrder = 8; % filter order
flags.lowpass = 1; % 1: low pass filter
% low-pass and high-pass filter design using firl()
% perform filtering using conv()
if flags.lowpass,
    h = firl(FilterOrder, Fcut/(Fs/2));   % help or doc firl(), frequency normalizat
else
    h = firl(FilterOrder, Fcut/(Fs/2), 'high'); % help firl(), frequency normalizati
end

h = h.'; % convert to column vector, because x is a column vector
y = conv(x, h,'same'); % filtering by convolution, why 'same'? remove the group dela
sound(y, Fs);
audiowrite('FilteredMusic.ogg',y,Fs);
```

```
Fcut = 1000; % Hz, cut off frequency (you may try 4 kHz cutoff frequency for LPF
FilterOrder = 256; % filter order
flags.lowpass = 1; % 1: low pass filter
% low-pass and high-pass filter design using firl()
% perform filtering using conv()
if flags.lowpass,
    h = firl(FilterOrder, Fcut/(Fs/2));   % help or doc firl(), frequency normal
else
    h = firl(FilterOrder, Fcut/(Fs/2), 'high'); % help firl(), frequency normali
end

h = h.'; % convert to column vector, because x is a column vector
y = conv(x, h,'same'); % filtering by convolution, why 'same'? remove the group
sound(y, Fs);
audiowrite('FilteredMusic.ogg',y,Fs);
```

We can hear significant effects on certain frequency removal as the longer the filter gets. In the figures, there would be sharper edges.

3. Re-sampling the music. The sampling rate of the original music is 48 kHz. Try to re-sampling the music without aliasing in frequency domain so that the new sampling rate is 72 kHz and 32/3 kHz, respectively. Detail your signal processing procedure along with your codes. If in the procedure, you apply filtering, tell how you design the filter. Output the music files as “HigherSamplingRateMusic.ogg” and “LowerSamplingRateMusic.ogg.” Note that you can verify your results by using MATLAB **resample()**.

72kHz:

```
I = 3;
D = 2;
y_resampled = resample(x, I, D); % used to verify your own resampler
sound(y_resampled, Fs*I/D);
audiowrite('HigherSamplingRateMusic.ogg',y_resampled,Fs*I/D);
```

32/3 kHz:

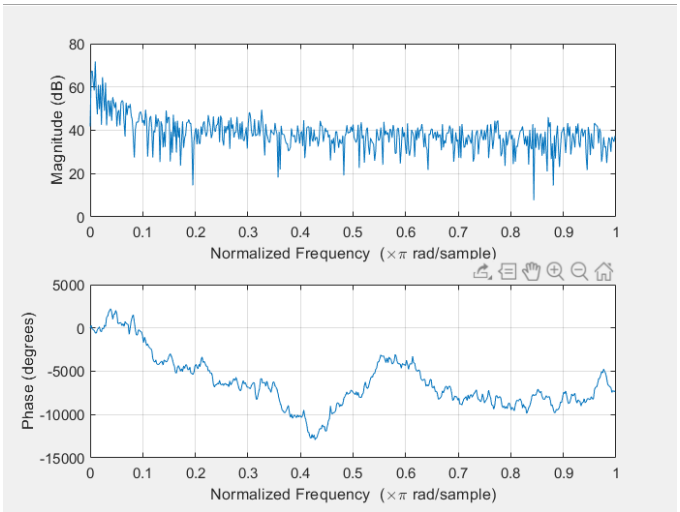
```
I = 2;
D = 9;
y_resampled = resample(x, I, D); % used to verify your own resampler
sound(y_resampled, Fs*I/D);
audiowrite('LowerSamplingRateMusic.ogg',y_resampled,Fs);
```

4. Noise the music to test your hearing sensitivity and also get the feeling of the noise in an audio signal

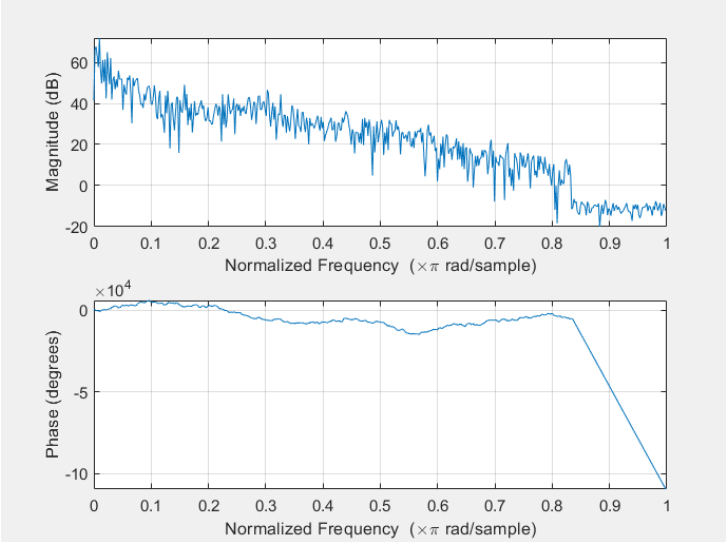
(a) Add a Gaussian white noise using MATLAB **randn()**to the original signal **x** such that the signal to noise ratio (SNR) is 30 dB. For the SNR estimation, assume the maximum signal magnitude is 1and the SNR is defined as the maximum signal divided by the standard deviation of the noise with dB conversion, i.e., take $20\log_{10}()$. **randn()**which generates normal/Gaussian distributed noise with zero mean and standard deviation = 1. Plot and compare the frequency-domain representation with and without adding noise. What do you hear?

```
SNR = 20; % in dB
noise_std = 10^(-SNR/20); % standard deviation of the noise
y_noise = x + noise_std*randn(M,1);  % add noise
sound(y_noise, Fs);
audiowrite('NoisyMusic.ogg',y_noise,Fs);

figure
freqz(y_noise,1);
```

With the noise:




Without the noise:




I can clearly hear noise between the original music.

(b) Tune the SNR and tell under what SNR you can “hear” the same quality as the original music.

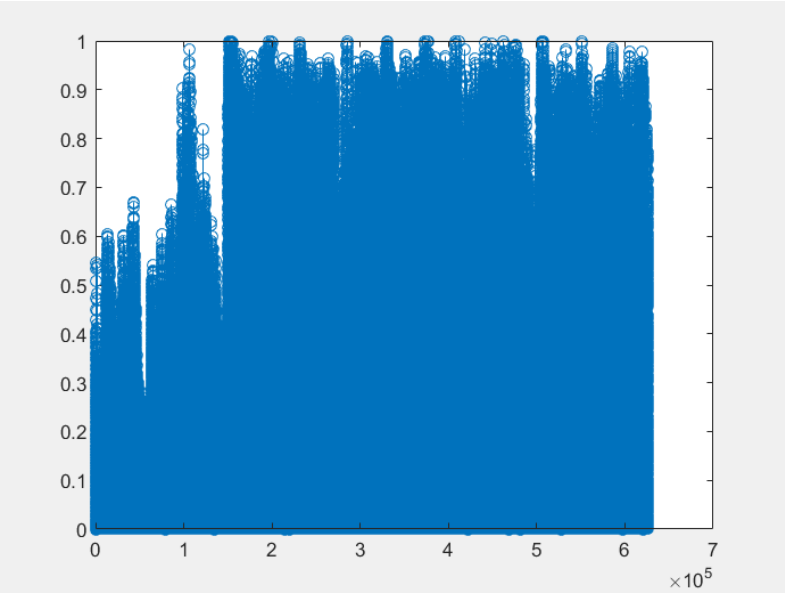
I search for the answer by adding SNR by 5 every time, and the effect by the noise decreases as I tried. At about 50, it remains only the original music.

5. Watermark the music: testing your hearing frequency band and generate inaudible signals. Add a single tone (i.e., a single frequency, cosine signal) to the original data so that you CANNOT hear this single tone. That is, you hear exactly the same as the original music after adding the single tone. Detail how you add this single tone and what is the frequency of the single tone. Plot and compare the spectra of the music with and without single-tone watermarking (This is how we decode the watermark and find if there is any copyright tissue). Save the music as “MusicWithInaudibleWaterMark.ogg”.

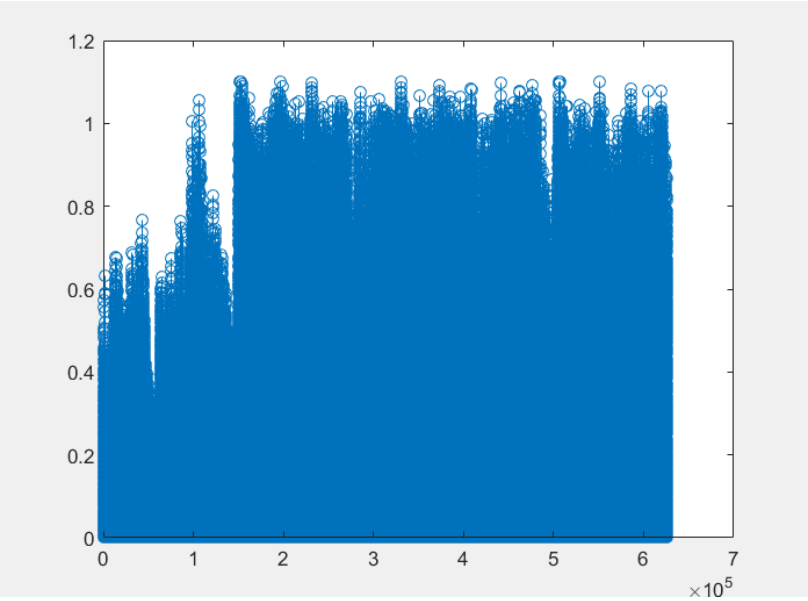
I search for the answer by adding Fc by 5*10^3 every time, and at about 65*10^3, it remains only the original music for me.

```
y_watermarked = x + Mag*cos(2*pi*Fc*[0:1:M-1]'/Fs); % Add this cosine signa
sound(y_watermarked,Fs);
audiowrite('MusicWithInaudibleWaterMark.ogg',y_watermarked,Fs);
figure
stem(abs(x));
figure
stem(abs(y_watermarked));
figure
freqz(y_watermarked,1);
```

Without the cosine:



With the signal:




65*10^3 seems to be a common answer among us, because I haven't heard of another reply yet.