EECE 323 – Fundamentals of Digital Signal Processing

Spring 2013

Section A

Practical Homework

MATLAB Application on Aliasing and Antialiasing

Student Name: Sharbel Dahlan

ID: 1004018456

Instructor: Dr. Jinane Biri

Due Date: Sunday, March 31, 2013

Table of Contents

Part (a)

The function that creates music was written in MATLAB. This function takes the frequency of a musical note, the duration of the note, and the frequency at which it was sampled at. It then returns the tone, after it processes its arguments. Processing the function inputs involves generating several tone harmonics and combining them and putting them in an envelope. The musical tone can be heard using the `soundsc` function.

The following is the code of the note function:

```
function tone = note(freq, dur, fs)
% Generates musical notes
%   freq: frequency
%   dur: duration
%   fs: sampling frequency

t = 0 : 1/fs : dur; %Time variable

x = t/dur;

% Realistic note amplitude envelope:
envelope = x.*(1-x).*(exp(-8*x) + 0.5*x.*(1-x));

% Realistic beating ratio variable:
beat = 0.08;

% Genertion of various tone harmonics:
harmonic0 = sin(2*pi*freq*t*(1-beat)) + sin(2*pi*freq*t*(1+beat));
harmonic1 = sin(2*pi*2*freq*t*(1-beat)) + sin(2*pi*2*freq*t*(1+beat));
harmonic2 = sin(2*pi*3*freq*t*(1-beat)) + sin(2*pi*3*freq*t*(1+beat));

% Combination of envelope and harmonics
tone = envelope.*(harmonic0 + 0.2*harmonic1 + 0.05*harmonic2);
```

```
% Amplitude normalization
tone = tone/max(tone);
end
```

The function was called to generate a tone with a frequency of 440 Hz, sampling frequency of 16 kHz, and a duration of 1 second.

The following code tests the function:

```
% call note with freq = 440, dur = 1, and fs = 16kHz
mynote = note(440, 1, 16000);
soundsc(mynote); % listen to the tone
plot(mynote); % Plot a continuous time representation of the note
grid;
xlabel('n');
ylabel('Amplitude');
```

The function plot is shown in *Figure 1*:

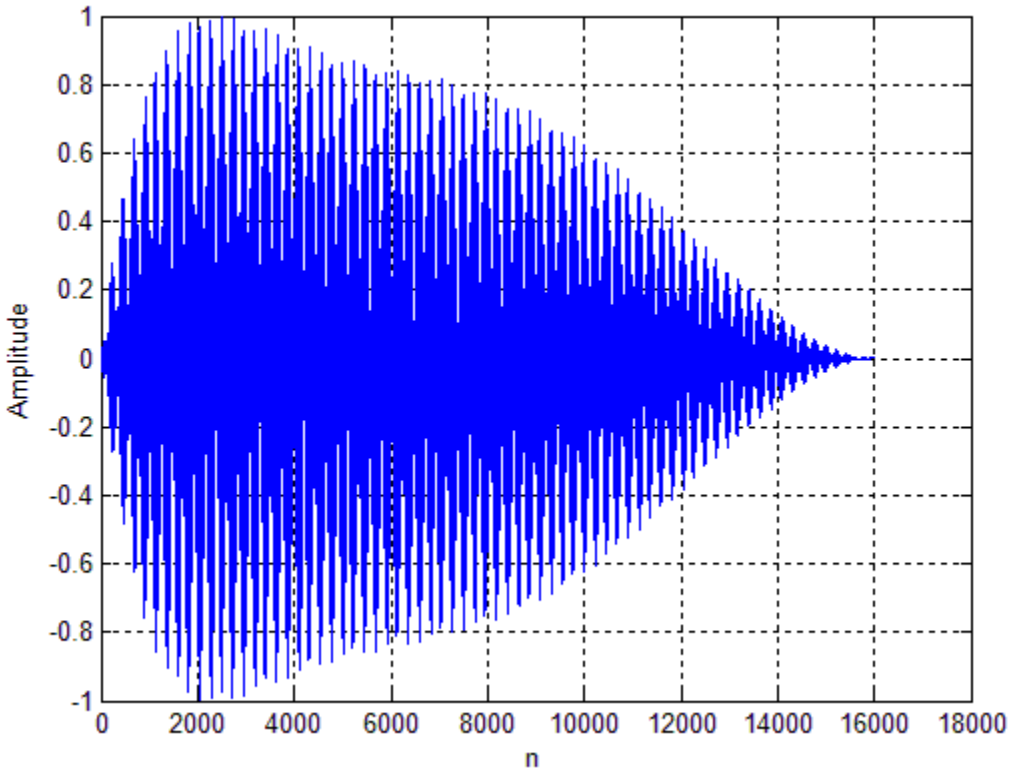

Figure 1: Part a

We can play with the envelope and see how changing it affects the tone. If the envelope was changed to `envelope = exp(12*x)`, the plot in *Figure 2* will be obtained. Changing the envelope affects the tone by having the amplitude of the tone change. This corresponds to the change in the envelope amplitude. When the envelope has low amplitude, the tone will have a lower volume. When the envelope goes higher in amplitude, the volume of the tone increases (until it reaches 1, the maximum).

Notice in the figure how the envelope was changed to an exponential. The way the sound has changed is exactly the same as adding a "fade-in" effect to the sound, which is an option that is very common in all audio editing software.

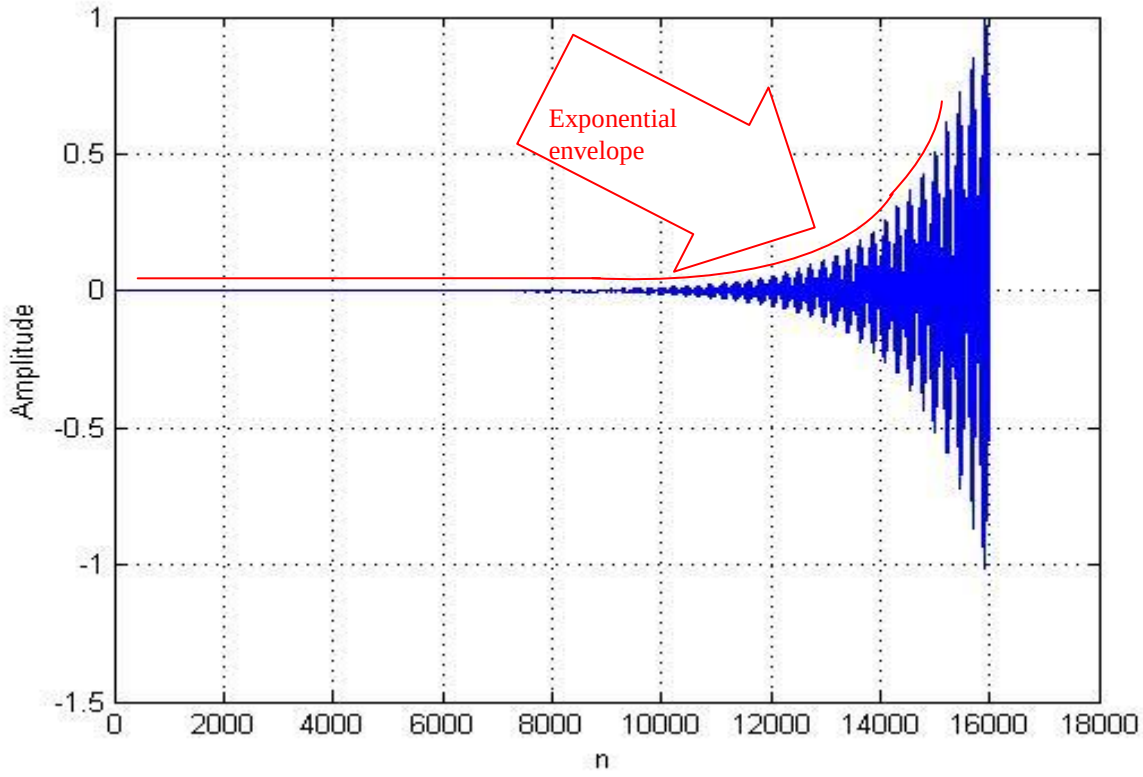

Figure 2: Part a after changing the envelope (to a simple exponential)

Part (b)

In this part, the first 15 notes of Beethoven's 9th Symphony are generated. This is done by creating two argument arrays, one array carrying the frequencies of the notes and another carrying the duration of each note. The corresponding tones that generate are concatenated in an array called `music`. The sound is played with the same frequency and sampling frequency settings of before.

Following is the code used:

```
freq=[1319 1319 1397 1568 1568 1397 1319 1175 1047 1047 1175 1319 1319 1175 1175];
dur=[0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.6 0.2 0.5];

music =[];

Len = length(dur);
% loop to generate the music array that contains the tones concatenated
for i = 1:1:Len;
    music = [music , note(freq(i),dur(i),16000)] ;
end
```

```
soundsc(music,16000);

plot(music);
xlabel('n')
ylabel('Amplitude')
grid;
```

In the loop that was used in the above code iterates `Len` times, where `Len` is the length of the duration array (`dur`). In each iteration, the loop adds the tones to the music array by calling the note function and taking the current frequency and duration values as the note function arguments (the sampling frequency remains 16kHz).

Following is the obtained plot:

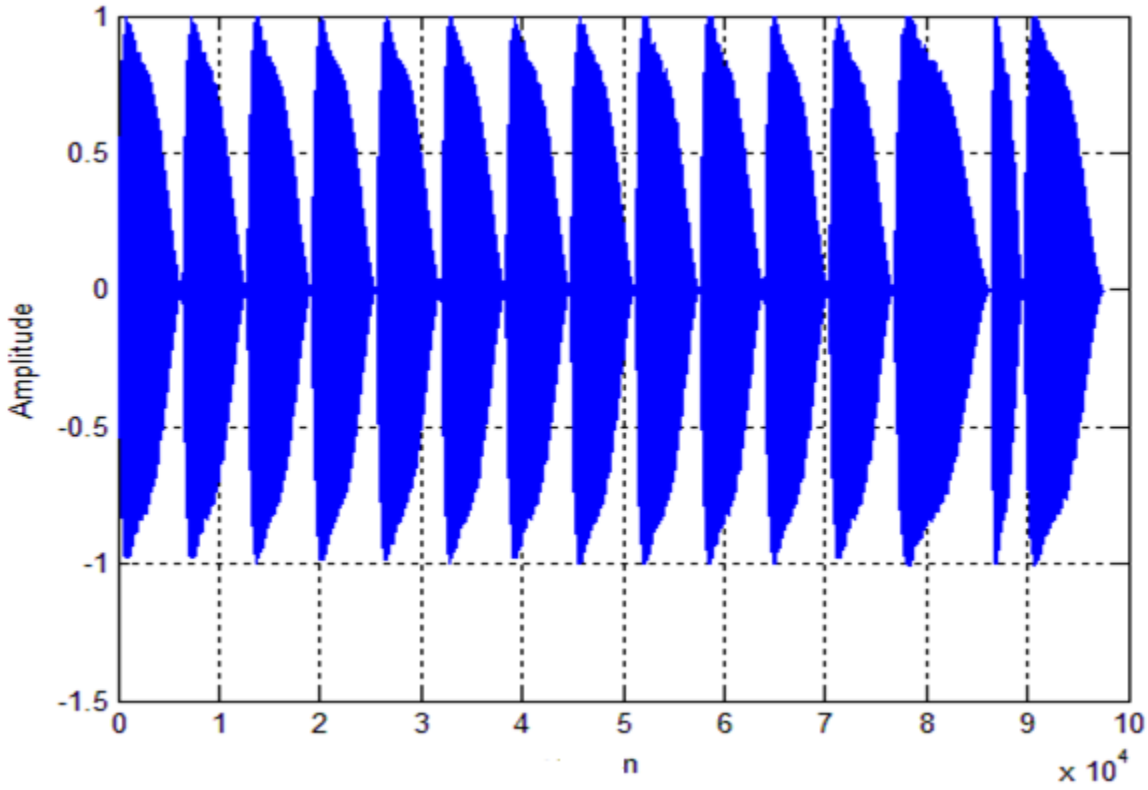

Figure 3: Plot of the first 15 notes of Beethoven's 9th Symphony

Part (c)

In this part, the sound signal generated in the previous part is downsampled at different downsampling rates, and the effect is recognized.

The following is the code used.

```
dsr = 2; %dsr: downsampling rate
new_samp_freq = (16000/dsr);
```

```
music_ = music(1:dsr:length(music));
soundsc(music_,new_samp_freq);

plot(music_);
xlabel('n')
ylabel('Amplitude')
grid;
```

In the above code, the downsampling rate (`dsr`) was chosen to be 2. *Figure 4* shows the plot of the signal after it was being downsampled by a rate of 2:

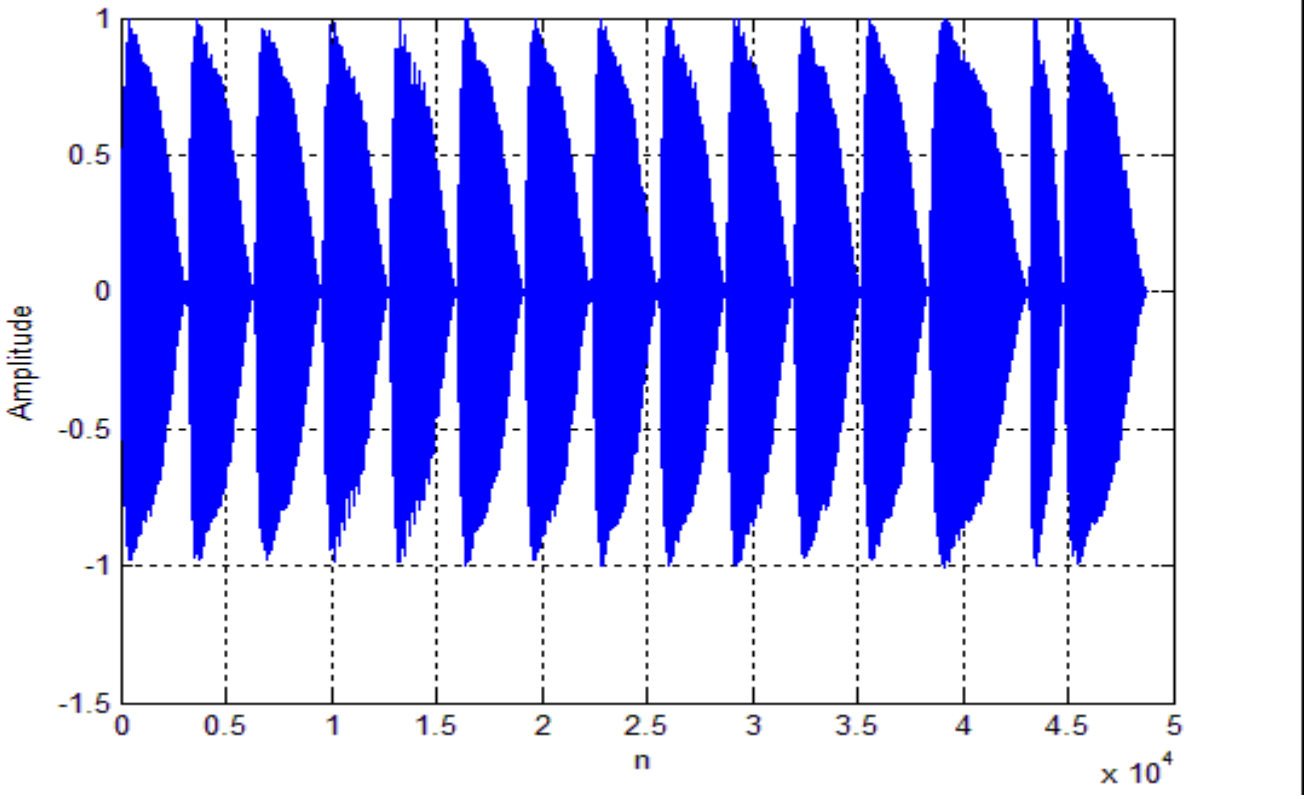

Figure 4: Beethoven downsampled, dsr = 2

After downsampling, the music will differ in the sense that it will have lower quality. The higher the downsampling ratio, the lower is the quality of the music. The higher the ratio of downsampling, the lower the quality of the music will be. Aliasing will occur whenever the resulting signal after downsampling is not more like the original one. *Figures 5* to *8* will show the plot of the signal representing Beethoven's 9th symphony with different downsampling rates. Aftert a downsampling rate of 6 is reached, the signal is aliased and the tone that is generated is much different than the original (only a few tones resemble the corresponding ones from the original signal, while others are totally different).

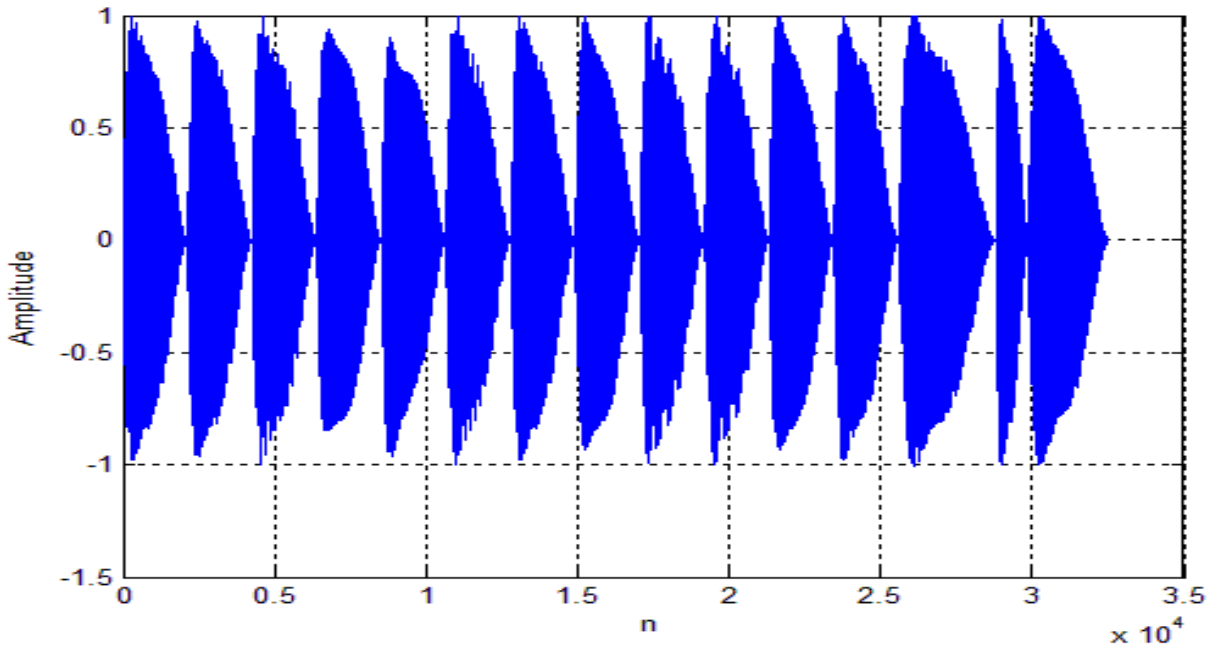

Figure 5: downsampled signal, dsr = 3

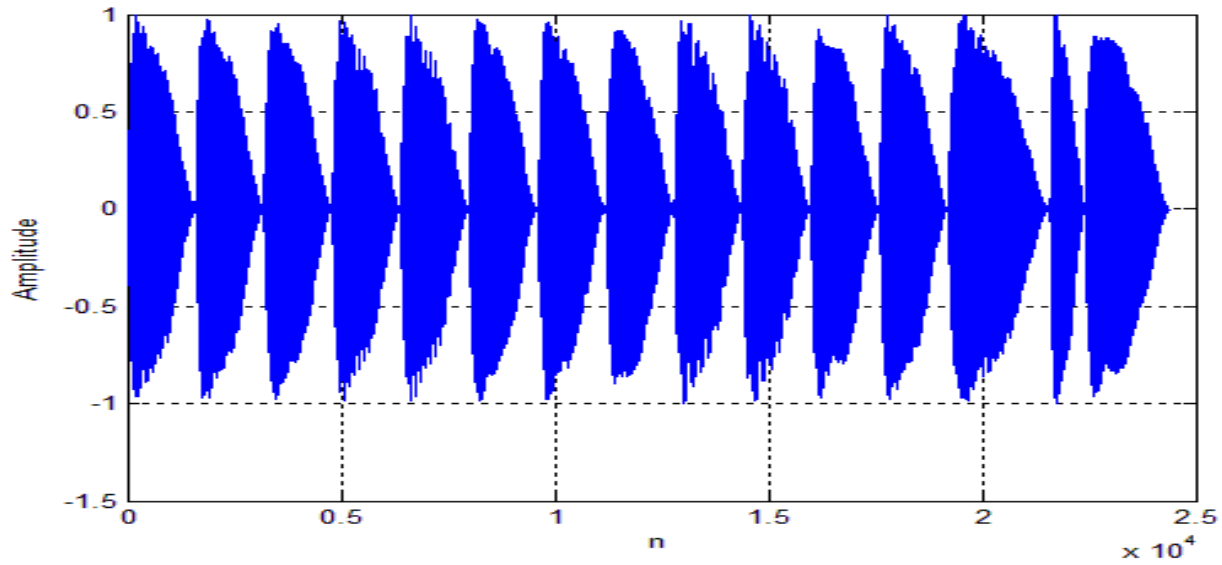

Figure 6: downsampled signal, dsr = 4

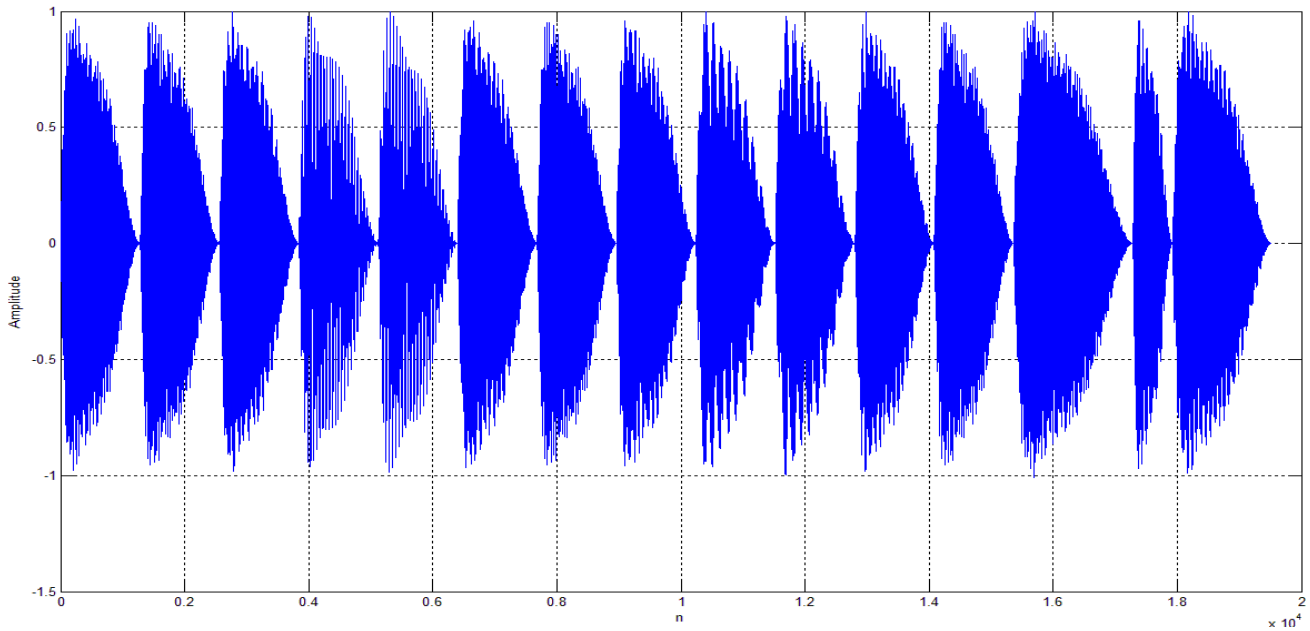

Figure 7: downsampled signal, dsr = 5

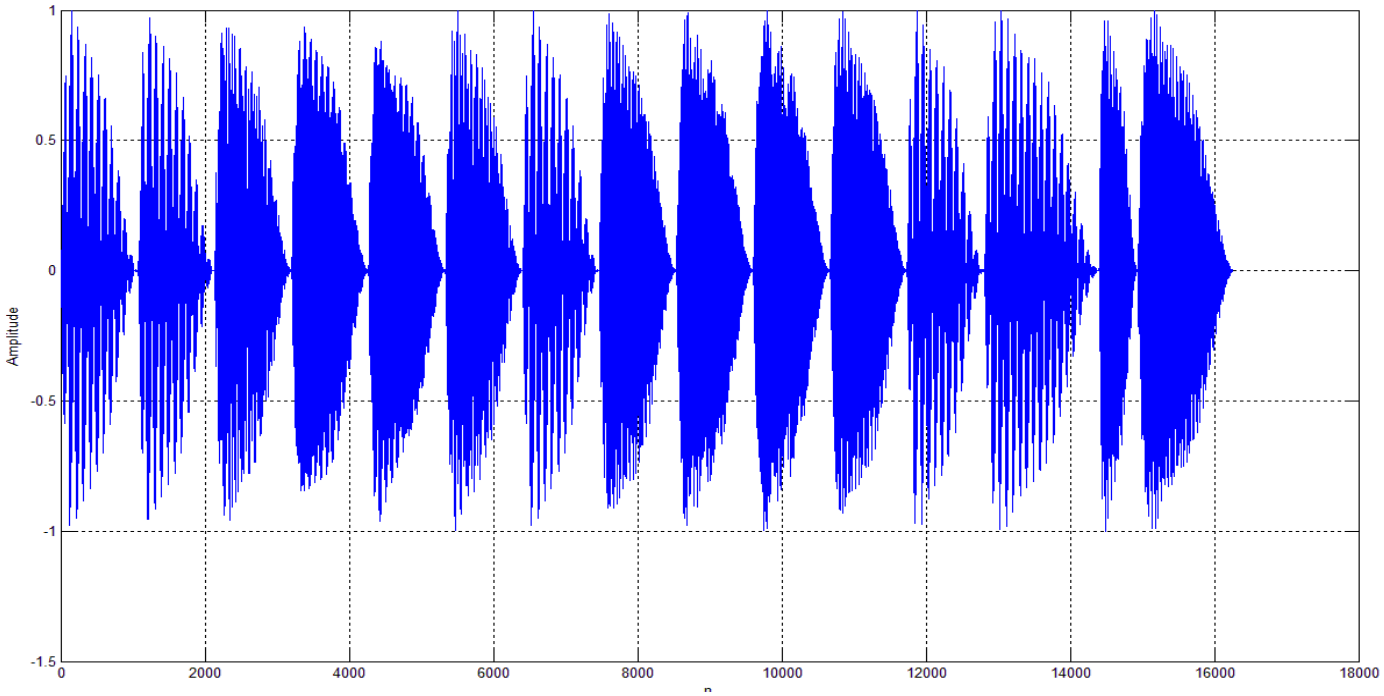

Figure 8: downsampled signal, dsr = 6 (aliased)

Notice the decrease in the number of samples (shown on the horizontal axis) the higher the downsampling rate is.

Part (d)

In this part, the signal will be filtered then downsampled. Following is the code used:

```
freq=[1319 1319 1397 1568 1568 1397 1319 1175 1047 1047 1175 1319 1319 1175 1175];
dur=[0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.6 0.2 0.5];

music =[];
original_samp_freq = 16000;

Len = length(dur);
for i = 1:1:Len;
    music = [music , note(freq(i),dur(i),original_samp_freq)] ;
end

soundsc(music,original_samp_freq);
subplot(3,1,1);
plot(music);
xlabel('n')
ylabel('Amplitude')
title('Original')
grid;

dsr = 3; %dsr: downsampling rate
new_samp_freq = (16000/dsr);

music_ = music(1:dsr:length(music));
soundsc(music_,new_samp_freq);

subplot(3,1,2);
plot(music_);
xlabel('n')
ylabel('Amplitude')
title('Downsampled')
grid;

filter_coeff = fir1(64, 1/dsr);
music = filter( filter_coeff, 1, music );

dsr = 3; %dsr: downsampling rate
new_samp_freq = (16000/dsr);

music_ = music(1:dsr:length(music));
soundsc(music_,new_samp_freq);

subplot(3,1,3);
plot(music_);
xlabel('n')
ylabel('Amplitude')
title('Filtered and Downsampled')
grid;
```

Figure 9 shows the three plots: 1) the original signal, 2) the downsampled signal and 3)the filtered then downsampled signal:

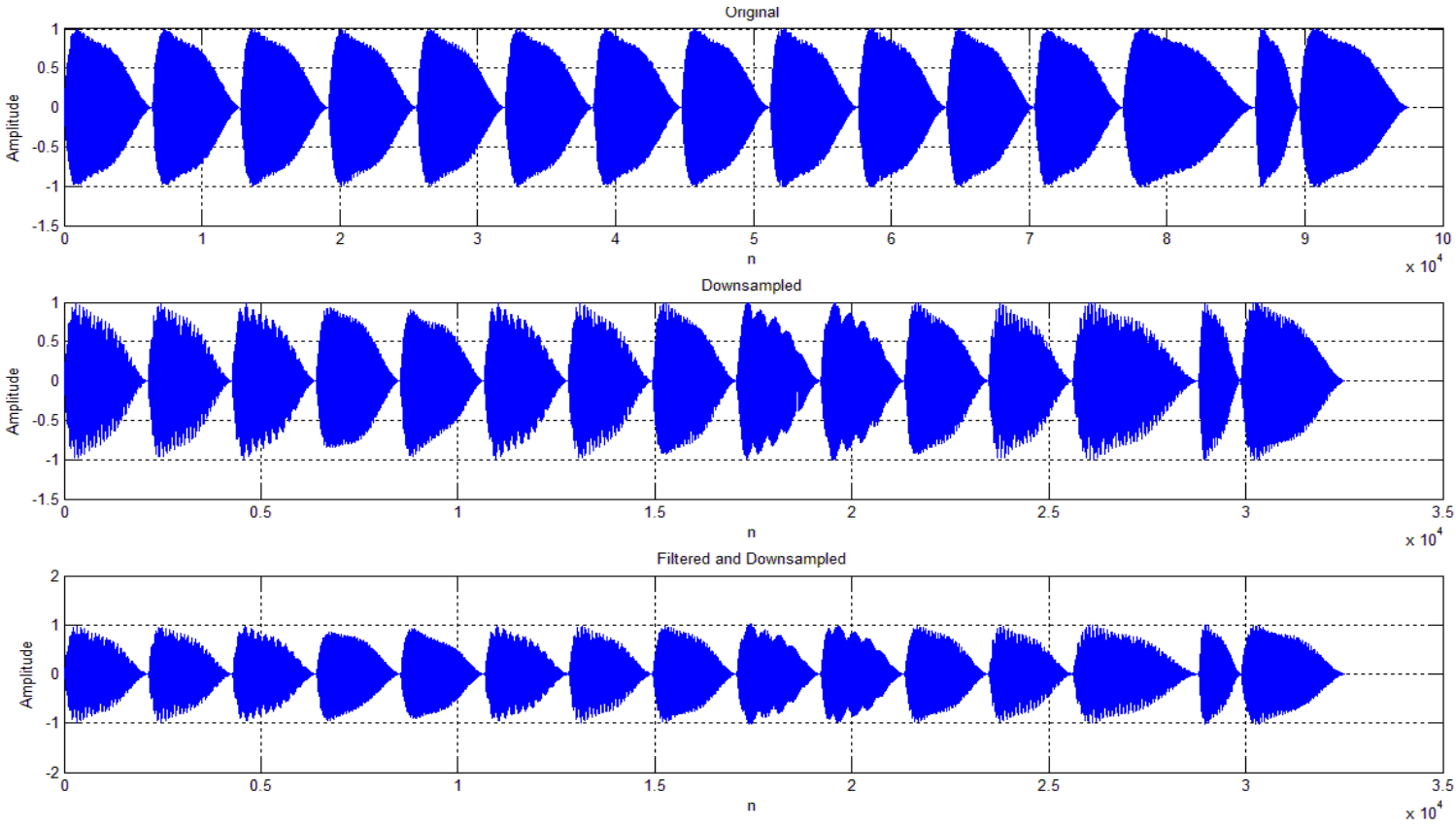

Figure 9: Filtering then downsampling, dsr 3

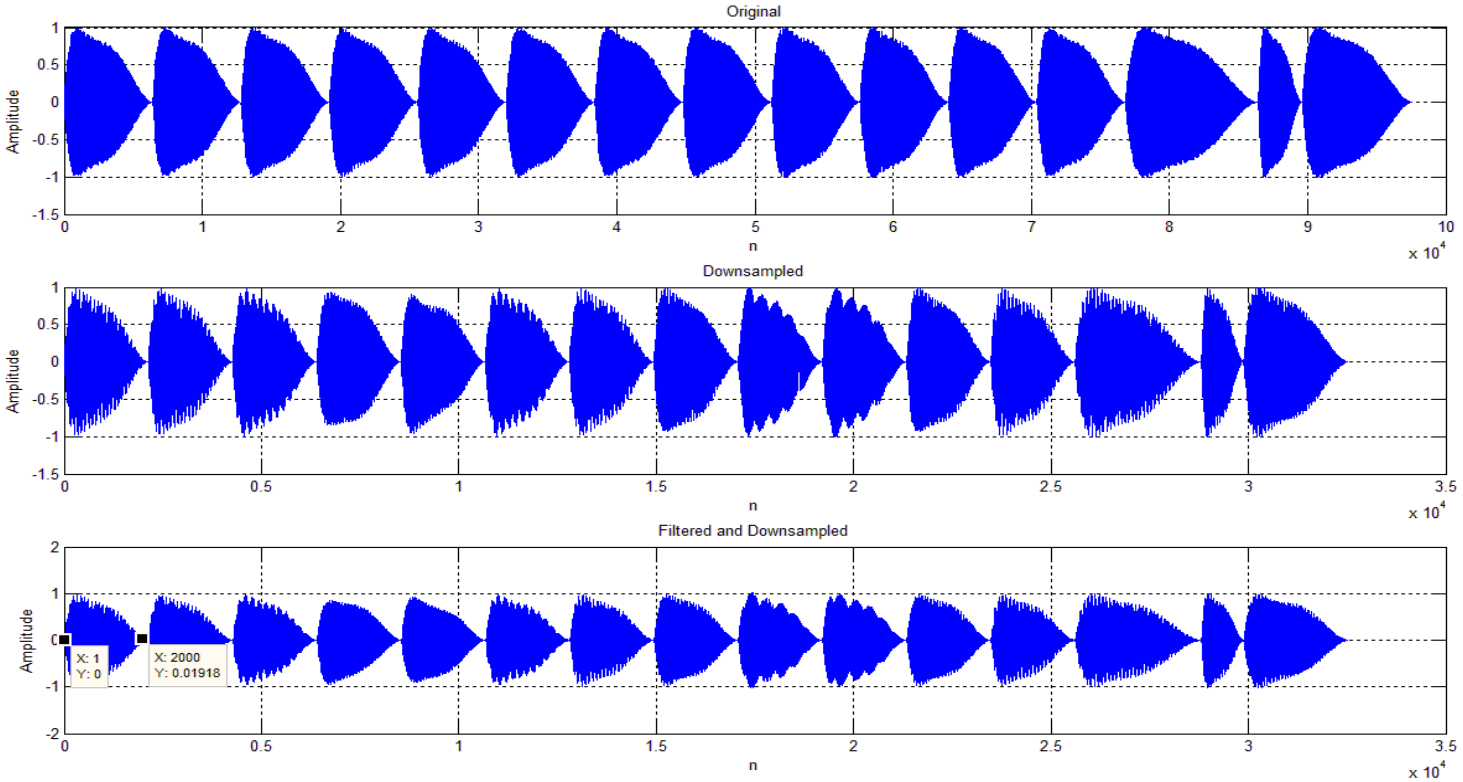

Figure 10: checking for the cut-off frequency of the filter

The built in MATLAB function `firl` will design a 64-point FIR low pass filter. Using help firl, it is found that the FIR lowpass filter has a cut-off frequency $0 < \omega_n < 1.0$, where 1.0 is half of the sample rate. If our sample rate is 16kHz, then 8kHz is half of our sample rate. Knowing that the cut-off frequency of the filter is ω_n/dsr, then

$$0 < \omega_n < \frac{16000}{2} = 8000$$

$$\frac{\omega_n}{dsr} = \frac{8000}{3} = 2666.67$$

Which is close to what we see from the plot (*Figure 10*): *x = 2000 – 1=1999, which is between 0 and 2666.7.*

When filtering before downsampling, the quality of the tone becomes better than only downsampling alone (distortion is less). For dsr values of 2, 3, and 4, the anti-aliasing technique was affective. For dsr value of 5, the anti-aliasing will not help because the signal is already damaged; there are not enough samples to represent the original signal, so the signal is already aliased and the filtering before downsampling does not help anymore.

Part (e)

In this part, an audio file with the .wav extension is created and read by MATLAB using the `waveread` function. The sampling frequency of the audio file should be 16 kHz. Again, the sound file is experimented with by first downsampling only, and then filtering before downsampling.

Following is the code used (only when testing dsr = 3):

```
myrecording = wavread('D:\sharbel.wav');
subplot(3,1,1);
plot(myrecording);
xlabel('n')
ylabel('Amplitude')
title('Original Recording')
grid;


dsr = 5; %dsr: downsampling rate
new_samp_freq = 16000/dsr;

music_ = myrecording(1:dsr:length(myrecording));
soundsc(music_,new_samp_freq);

subplot(3,1,2);
plot(music_);
xlabel('n')
ylabel('Amplitude')
title('Downsampled w/ dsr = 10')
grid;
```

```
filter_coeff = fir1(64, 1/dsr);
myrecording = filter( filter_coeff, 1, myrecording );

dsr = 5; %dsr: downsampling rate
new_samp_freq = (16000/dsr);

music_ = myrecording(1:dsr:length(myrecording));
soundsc(music_,new_samp_freq);

subplot(3,1,3);
plot(music_);
xlabel('n')
ylabel('Amplitude')
title('Filtered and Downsampled w/ dsr = 10')
grid;
```

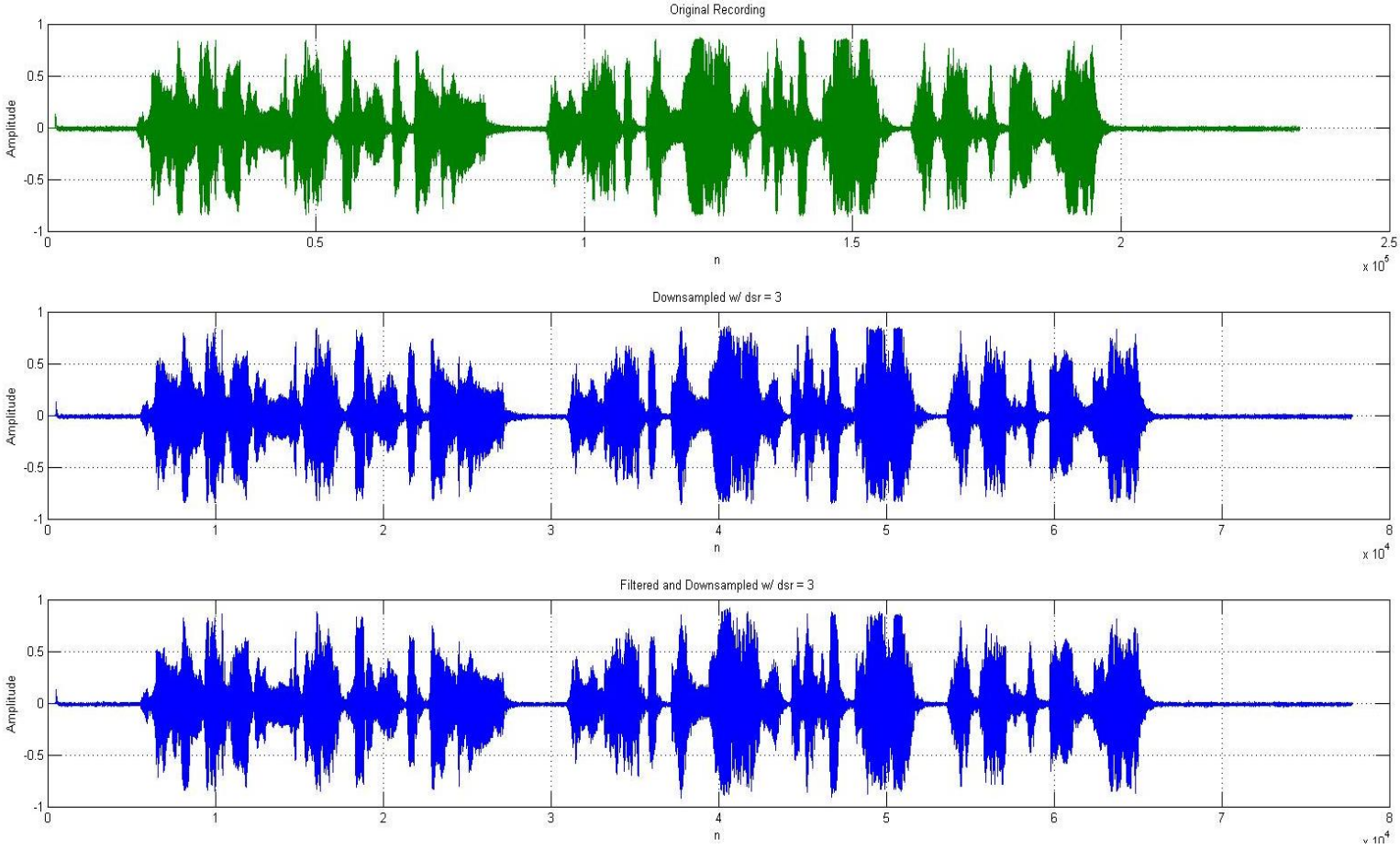

Figure 12: The 16 kHz recording (original, downsampled, and filtered then downsampled)

When the audio signal was downsampled, the audio quality has decreased (as expected) *Figure 12* shows the audio signal that is downsampled at dsr = 3. However, filtering and then downsampling has produced a better audio quality.

When filtering was applied before the downsampling, the anti-aliasing technique was effective only till dsr = 5. After reaching a dsr of 6, the audio signal was ruined, because the signal was aliased and the anti-aliasing was not very effective (as it was for less dsr's). *Figure 13* shows the aliased audio signal after making dsr = 6.

The reason for having results of this part similar to the previous parts is that the samling rate of this signal is 16kHz, which is the same as the one that was set for the tone in the previous parts.

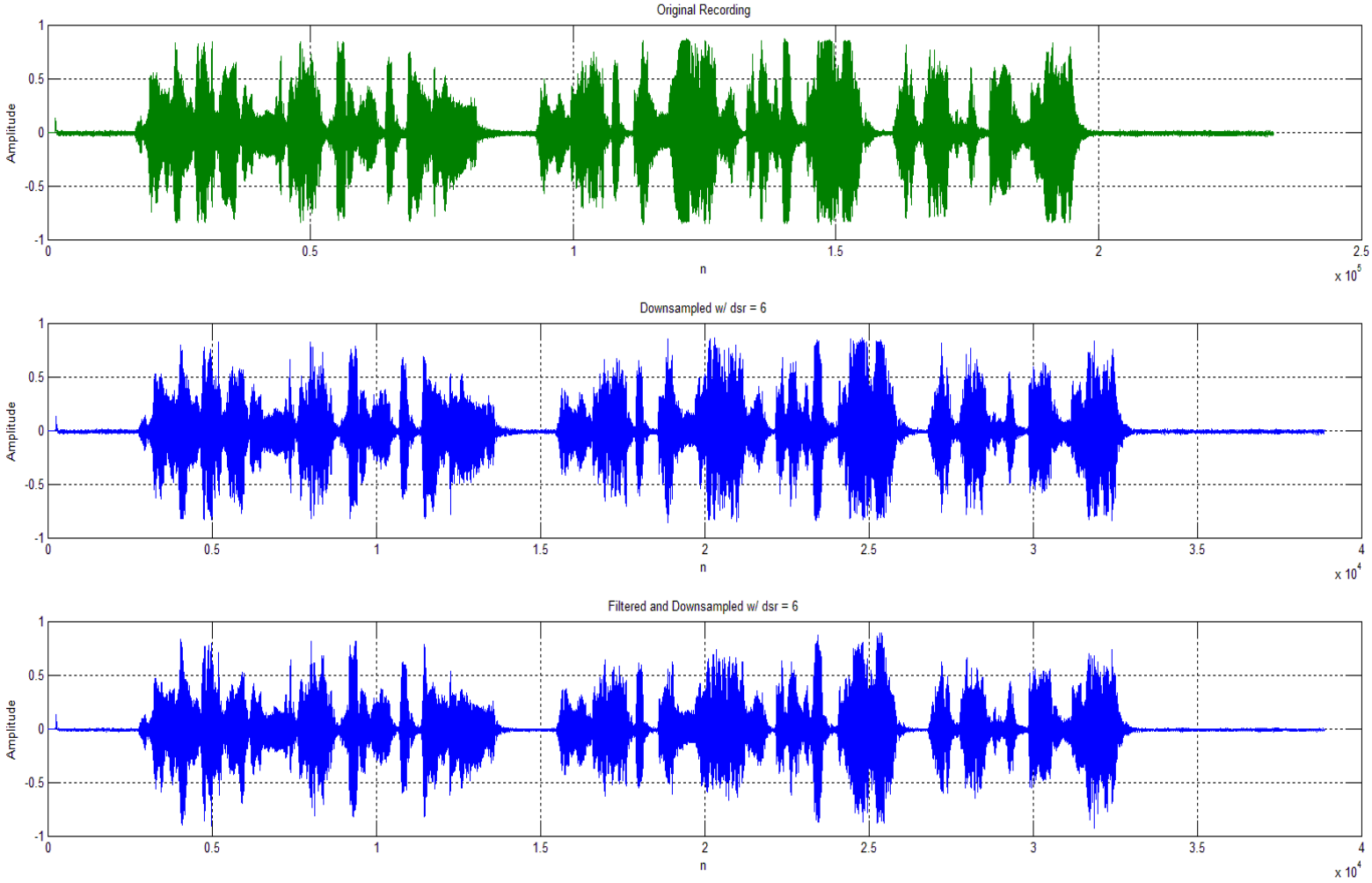

Figure 13: dsr = 6

The following figure shows the sound signal being downsampled with dsr = 10. Notice how the downsampled version is missing a lot of the original information.

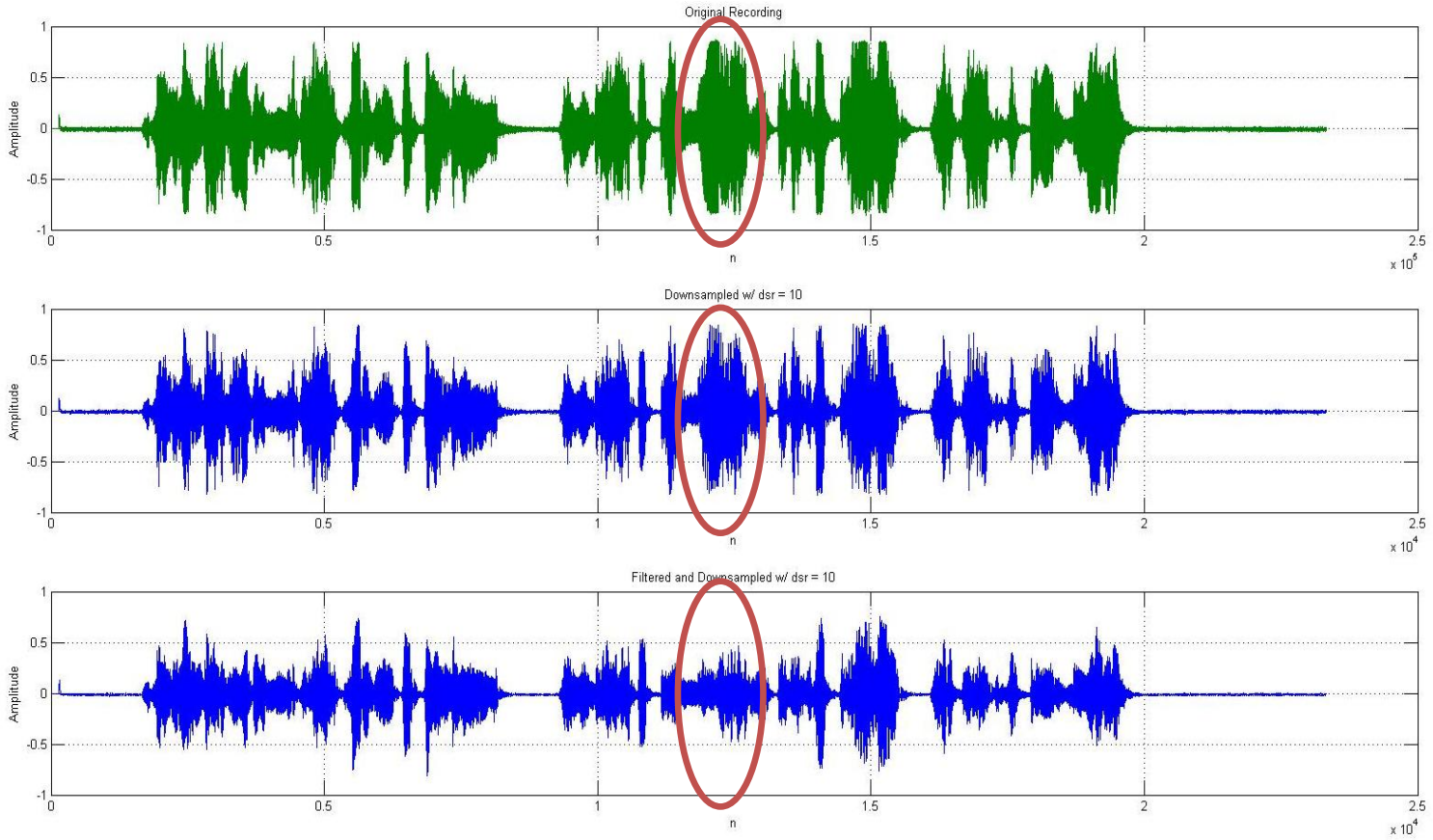

Figure 14: The 16 kHz recording (original, downsampled, and filtered then downsampled) w/ dsr = 10