# Lab One: Discrete-Time Signals in the Time Domain Using MATLAB

***Digital Signal Processing***

Dr. Pearlstein

Matthew Bohr

28 September 2022

| Student | Analysis | Development | Coding | Results | Writing |
|---|---|---|---|---|---|
| Matthew Bohr | 100% | 100% | 100% | 100% | 100% |

## Introduction

In this lab, the students were tasked with using MATLAB for simple digital signal processing purposes. The objectives of the lab were the following:

1. Create and visualize the harmonic series for a square wave.
2. Sample a sine wave at different sampling frequencies.
3. Create sine waves with increasing frequencies.
4. Reading a sound file and sampling a portion of it.
5. Use two different convolution algorithms for implementing a lowpass filter.

The students also analyzed various characteristics about the digital signals and whether the Nyquist Theorem is met.

## Background

A discrete-time signal can be expressed very similarly to a continuous-time signal. A continuous-time sinusoid with a frequency $f$ is shown below:

$$x(t) = Asin(2\pi ft)$$

The signal x(t) in continuous time corresponds to the sequence $\{x[n]\}$ in discrete time:

$$x(t) \triangleq \{x[n]\}$$

$$x[n] = Asin(2\pi \frac{f_0}{F_s} n)$$

In which $f_0$ is the fundamental frequency, $F_S$ is the sampling frequency, and n is the index of the sample.

Another parallel that can be drawn between continuous-time and discrete time is with the convolution integral and the convolution sum. Finding the output of a system after injecting a continuous-time signal into the input requires us to define the impulse response as:

$$h(t) = F[x(t)]|_{x(t)=\delta(t)} = F[\delta(t)]$$

The impulse response $h(t) = F[\delta(t)]$ gives the output to a system given a single unit impulse as an input. Recalling an important property of the unit impulse function:

$$x(t) = \int_{-\infty}^{\infty} x(\lambda)\delta(t-\lambda)\,d\lambda$$

The above expression states that any input signal can be represented as the integral of the unit impulse at some instant in time multiplied by the value of the input signal at that time, for all instances of time.

The previous two equations can be used to derive the convolution integral to describe the output signal given the input signal and system response:

$$y(t) = F[x(t)]$$

$$y(t) = F\left[\int_{-\infty}^{\infty} x(\lambda)\delta(t-\lambda)\,d\lambda\right]$$

$$y(t) = \int_{-\infty}^{\infty} x(\lambda)F[\delta(t-\lambda)]\,d\lambda$$

$$y(t) = \int_{-\infty}^{\infty} x(\lambda)h(t-\lambda)d\lambda = \int_{-\infty}^{\infty} h(\lambda)x(t-\lambda)d\lambda = x(t) * h(t)$$

Because the impulse response describes the system's response to a single impulse, the total system response can be expressed as the sum of system responses shifted to a particular moment in time multiplied by the amplitude of the input signal at that instant in time, summed for all instances of time to give the total response of the system y(t) over all time due to an input x(t).

The convolution sum in discrete-time is expressed in an almost identical manner as the convolution integral in continuous-time, replacing integrals with summations and continuous-time signals with discrete-time sequences. It is sometimes easier to understand the unit impulse and convolution since we are now dealing with discrete points.

First, the unit impulse sequence must be defined as the system's response to a unit impulse sample:

$$h[n] = F\big[x[n]\big]|_{x[n]=\delta[n]} = F[\delta[n]]$$

An input sequence is the summation of the multiplication of a unit impulse sample shifted by some index with the input sequence at that same index for all indexes in the sequence:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

The above two expressions can be used to derive the output sequence $y[n]$ given an input sequence $x[n]$:

$$y[n] = F\big[x[n]\big]$$

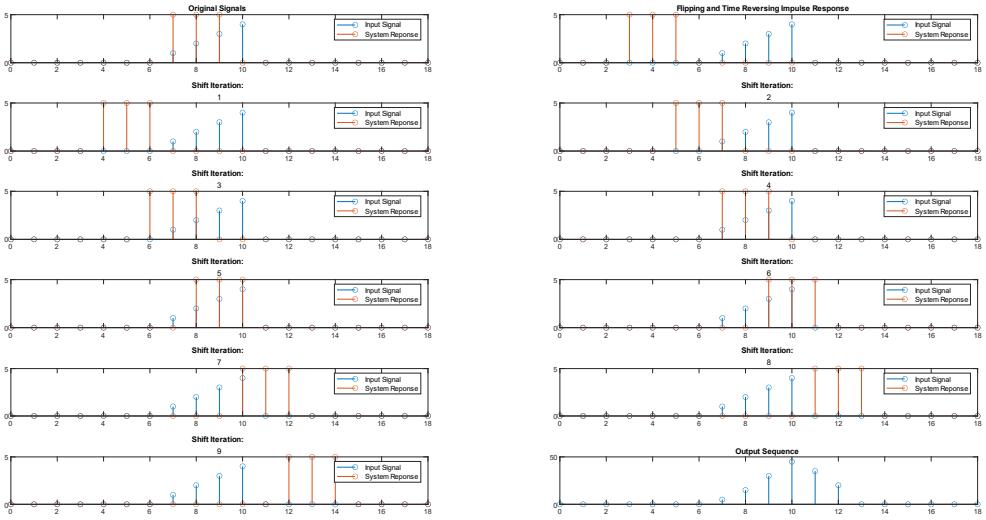
$$y[n] = F\left[\sum_{k=-\infty}^{\infty} x[k]\delta[n-k]\right]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]F[\delta[n-k]]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] = x[n] * h[n]$$

The above expression states the output response is the discrete summation of the system's impulse response at an index k multiplied by the amplitude of the input sequence at index k for all indexes. Figure 1 shows an example of this sliding mechanism involved with the convolution operation.

*Figure 2: Graphical Example of the Convolution Sum.*




Convolution is an important topic as it tells us how a particular system will shape an input signal to yield an output signal while working in the time domain. With regard to this lab, convolution is used to implement a low-pass filter on an audio signal. Due to the discrete-time nature of digital signal processing, implementing a low-pass filter is easier done by convolving the

input signal with the low-pass filter's impulse response (a sinc wave), which is equivalent to multiplying the input signal by a low-pass square box in the frequency domain. One can see the contrast between this discrete-time low-pass filter implemented using convolution and a continuous-time low-pass filter realized using an RC circuit which attenuates higher frequency components "automatically" through the act of integrating the current through the capacitor.

## Procedure

In the first part of this lab the students were tasked with generating and visualizing the harmonic series for a square wave while using discrete-time sampling. The duration of the signal was 2 seconds, the tone frequency was $f_0 = 440\,Hz$, and the sampling frequency was $f_s = 8000\,Hz$. Harmonics were added to the fundamental wave in increasing groups of two harmonics. A dense signal with a sampling frequency of $F_s = 80000\,Hz$ was used to overlap the original signal. The code is shown in Appendix A and it consists of a single for loop which has the sole purpose of adding on additional harmonics, graphing the results, and playing an audio form of the signal out loud.

In part two of the lab the students needed to sample a sine wave and then subsample it multiple times via decimation. The signal needed to be 2 seconds long, have a tone frequency of $f_0 = 1000\,Hz$, and a sampling frequency of $f_s = 48000\,Hz$. The sampling frequency was sampled at a $6^{th}$, $11^{th}$, $16^{th}$, $21^{st}$, $26^{th}$, and $31^{st}$ of the initial sampling rate. Appendix B shows the code for part 2 and it consists of a for loop that is used for running through the sampling frequency divisors from 6 to 31 in steps of 5 and then applying the new sampling rate to the sine wave.

In part three the students needed to sample multiple sine waves, each with a frequency an integer value times the frequency of the initial tone. The signal needed to be 2 seconds long and be sampled at a rate of $f_s = 8000\,Hz$. Part three needed to be done twice, once with a tone frequency of $f_0 = 995\,Hz$ and another with a tone frequency of $f_0 = 1000\,Hz$. The code is shown in Appendix 3.

In part four the students needed to read an audio file into MATLAB and graph the entire signal as well as 1000 samples from somewhere interesting in the signal.

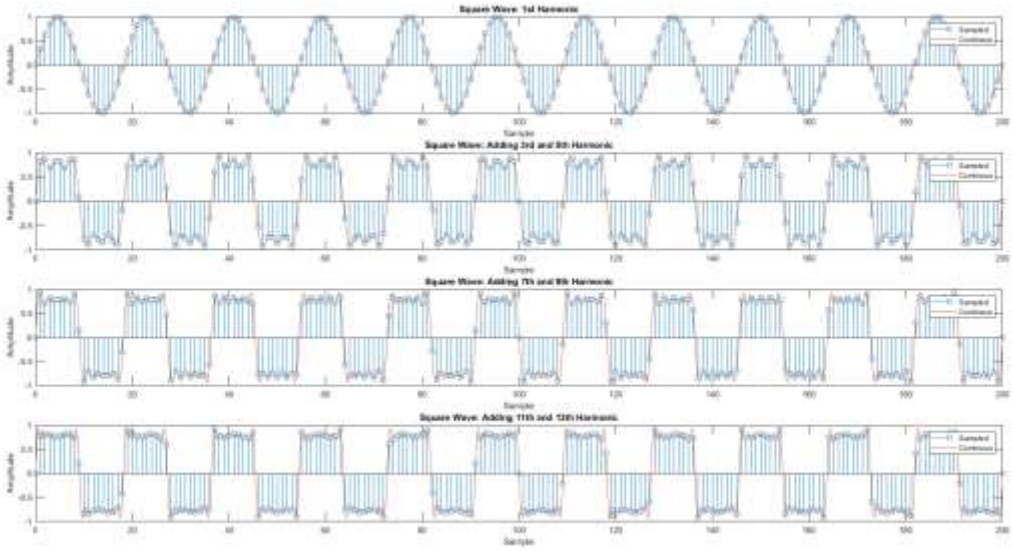
In part 5 the students needed to pass the audio signal through a low pass filter. This low pass filter was to be implemented in two ways. Once with a built-in MATLAB convolution function, the other with a hardcoded convolution function.

## Results

**Part One:**

Figure 2 shows the graphs for part one. The blue stem plot shows the sampled wave while the orange continuous plot shows the "continuous" form of the wave. The signal in the first graph is a simple sine wave because the square wave only has its first harmonic. In the second, third, and fourth graphs, adding more and more harmonics causes the signal to become a more accurate approximation of a square wave.

*Figure 2: Generating a Square Wave by Summing Together Increasing Harmonics*



Qualitative and quantitative descriptions of the various waveforms are given below in Table 1. Adding the higher harmonics caused the square waveform to appear to have a higher tone. This makes sense since additional harmonics with higher frequencies are being superimposed on the signal, meaning there is some higher frequency content.

*Table 1: Description of Approximated Square Wave Signals*

| Harmonics | Description | Maximum frequency (Hz) | Sampling rate | Nyquist met? | Sound |
|---|---|---|---|---|---|
| 1st | Single note | 440 | 8000 | Yes | Yes |
| 1st – 5th | Increased tone, multiple tones overlayed, louder | 2200 | 8000 | Yes | Yes |
| 1st – 9th | Increased tone, multiple tones overlayed, louder | 3960 | 8000 | Yes | Yes, aliasing |
| 1st – 13th | Increased tone, multiple tones overlayed. | 5720 | 8000 | No | Yes, aliasing |

A few questions arose:

*Question One: Does the sampling satisfy the Nyquist criterion? Why or why not?*

From looking at Table 2, it is obvious of which of the four approximated square waves does not satisfy Nyquist's Theorem. The rate at which the signal is sampled must be at least double that of the highest frequency component in the wave. The maximum frequency column in Table 2 shows the highest frequency component for each square wave approximation. The final approximation does not satisfy the Nyquist criterion since its maximum frequency times 2 is greater than the sampling frequency.

*Question Two: Were you able to hear the implications of aliasing?*

Yes, on the 3rd and especially the 4th approximation, there was a humming noise present that seemed to disturb the cleanness of the audio.

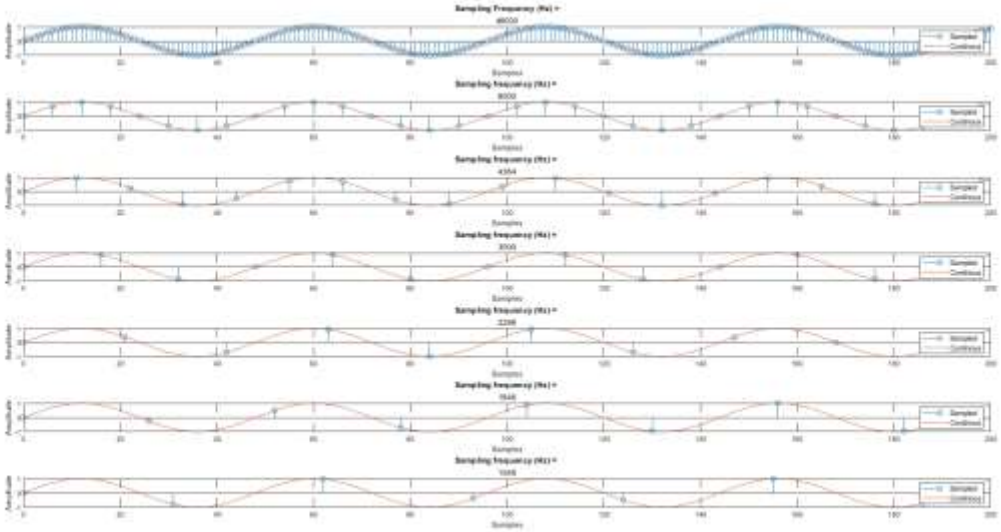
*Question Three:* How can you relate the sampling rates vs. frequency of wiggles in harmonics series with aliasing?

More squiggles means more aliasing.

**Part Two:**

Figure 3 shows the graphs of the sine wave when sampled at different rates. In the last two graphs, notice how there is under two samples per period, indicating that the Nyquist criterion is not met for these sampling divisors.

*Figure 3: Sampling a Sine Wave at Decreasing Sampling Rates*



Below in Table 2 shows a description of each of these sine waves and whether they satisfy the Nyquist criterion. As expected, the final two rows in the table do not satisfy the Nyquist theorem because twice their highest frequency is more than the sampling frequency.

Another observation stems from the sound each signal made. Although aliasing was not really observed until the final signal, a decrease in sound quality did become apparent as the sampling frequency decreased. Moreover, although it was the same sine wave, a decreased sampling rate also caused an apparent increase in the tone of the audio.

*Table 2: Description of Effects of Decreasing Sampling Rate*

| **Sampling divisor** | **Effective sampling rate (Hz)** | **Description** | **Nyquist met?** | **Sound?** |
|---|---|---|---|---|
| 1 | 48000 | Smooth tone | Yes | Yes |
| 6 | 8000 | High pitch | Yes | Yes |

| | | | | |
|---|---|---|---|---|
| 11 | 4364 | Very high pitch | Yes | Yes |
| 16 | 3000 | Extremely high pitch | Yes | Yes |
| 21 | 2286 | Scattered static noise | Yes | No |
| 26 | 1846 | Scattered static noise | No | No |
| 31 | 1548 | Scattered static noise | No | No, aliasing |

From this part of the lab, a few questions naturally arose:

*Question One:* For each of the steps above, does the sampling satisfy the Nyquist criterion.

The Nyquist Criterion is satisfied for all of the steps except for when the sampling rates were 1846 Hz and 1548 Hz as indicated in Table 1. This is because the tone frequency is 1000 Hz, meaning the sampling frequency must be at least double that to satisfy the Nyquist Criterion.

*Question Two:* Were you able to hear the implications of aliasing?

I was able to hear the implications of aliasing when the sampling rate was 1548. It sounded like an additional tone was overlayed another tone.

*Question Three:* How can you relate the sampling rates vs. frequency of wiggles in harmonic series with aliasing?

More squiggles means more aliasing.

**Part Three:**

In part three, the students needed to multiply the frequency of some sine wave by an integer while keeping the sampling frequency constant. This was done with two tone frequencies, one of 1000 Hz as shown in Figure 4 and one of 995 Hz of Figure 5.

*Figure 4: Increasing the Signal Frequency while Keeping the Sampling Rate at 8000 Hz (Fundamental Frequency = 1000Hz)*

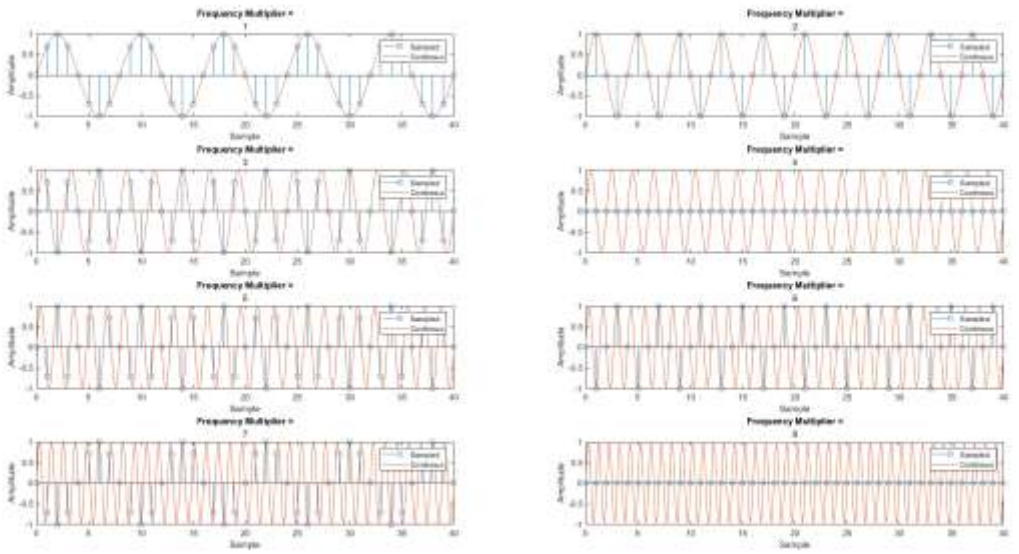


*Figure 5: Increasing the Signal Frequency while Keeping the Sampling Rate at 8000 Hz (Fundamental Frequency = 995Hz)*

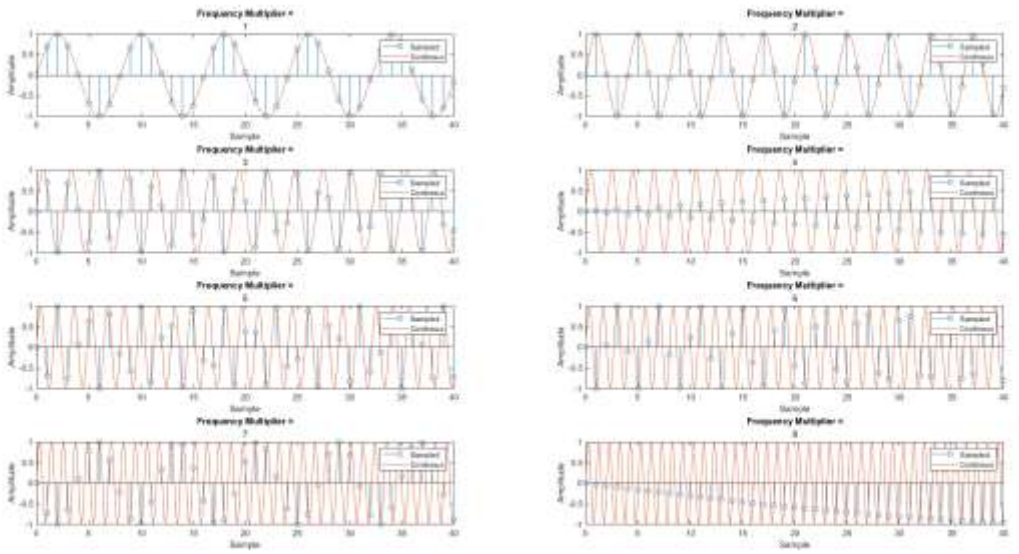


Table 3 shows a description of each signal when the base frequency was 1000 Hz and Table 4 shows a description of each signal when the base frequency was 995 Hz. The students tested frequencies that would violate the Nyquist theorem to hear the impact of aliasing on the audio signal and to see the visual consequences through the graphs.

*Table 3: Description of Increasing Frequency Sine Waves (Fundamental Frequency = 1000 Hz)*

| Initial Tone Frequency of 1000 Hz | | | | |
|---|---|---|---|---|
| **Frequency multiplier** | **Signal frequency (Hz)** | **Sampling rate (Hz)** | **Nyquist met?** | **Sound** |
| 1 | 1000 | 8000 | Yes | Yes |
| 2 | 2000 | 8000 | Yes | Yes |
| 3 | 3000 | 8000 | Yes | Yes |
| 4 | 4000 | 8000 | Yes | No |
| 5 | 5000 | 8000 | No | Yes |
| 6 | 6000 | 8000 | No | Yes |
| 7 | 7000 | 8000 | No | Yes |
| 8 | 8000 | 8000 | No | No |

*Table 4: Description of Increasing Frequency Sine Waves (Fundamental Frequency = 995 Hz)*

| Initial Tone Frequency of 995 Hz | | | | |
|---|---|---|---|---|
| **Frequency multiplier** | **Signal frequency (Hz)** | **Sampling rate (Hz)** | **Nyquist met?** | **Sound** |
| 1 | 995 | 8000 | Yes | Yes |
| 2 | 1990 | 8000 | Yes | Yes |
| 3 | 2985 | 8000 | Yes | Yes |
| 4 | 3980 | 8000 | Yes | Yes |
| 5 | 4975 | 8000 | No | Yes |
| 6 | 5970 | 8000 | No | Yes |
| 7 | 6965 | 8000 | No | Yes |
| 8 | 7960 | 8000 | No | Low hum |

During this part of the lab, some questions naturally arose:

*Question One:* At what point did the sounds go from increasing to decreasing frequency, and why?

When the tone frequency is 1000 Hz, the apparent tone of the sound starts to decrease when the frequency multiplier is 6 and above. The reason for this is because the sampling is occurring on the original sine wave in such a way that it is enveloping another sine wave with a lower frequency as shown in Figure 4. Moreover, there is no sound when the frequency multiplier is 4 and 8. This occurs because the sine wave is being sampled at half a period and a full period for those two multipliers respectively. A sine wave is always 0 at half its period and after one full period. This

means that the samples will always be 0, which is evident in Figure 4, so no sound is produced.

For the 995 Hz wave, the apparent tone started to decrease when the frequency multiple was 4.

*Question Two: What major difference did you observer between f0 = 995Hz and f0 = 1000 Hz.*

*It seems that the apparent frequency started decreasing at different frequency multipliers. Also, there was no sound for both cases when the frequency multiplier was 8, but there was only no sound for the tone frequency multiplier of 4 for the tone frequency of 1000 Hz.*

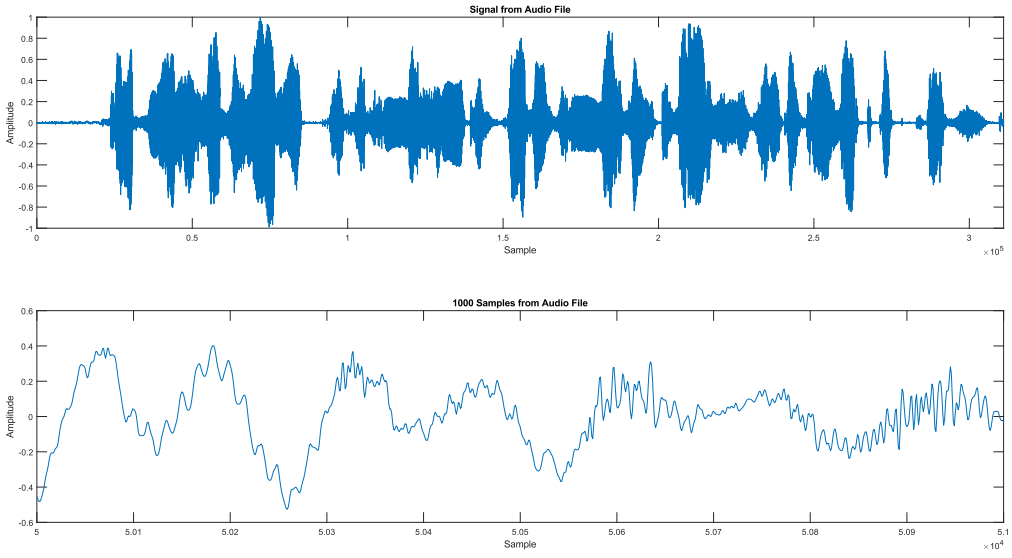
*Question Three:* Why do you think that you couldn't hear any?

See explanation in question one.

**Part Four:**

Figure 6 shows the results of graphing an audio signal as well as graphing 1000 samples in the middle of it.

*Figure 6: Reading in an Audio Signal and Plotting its Samples*



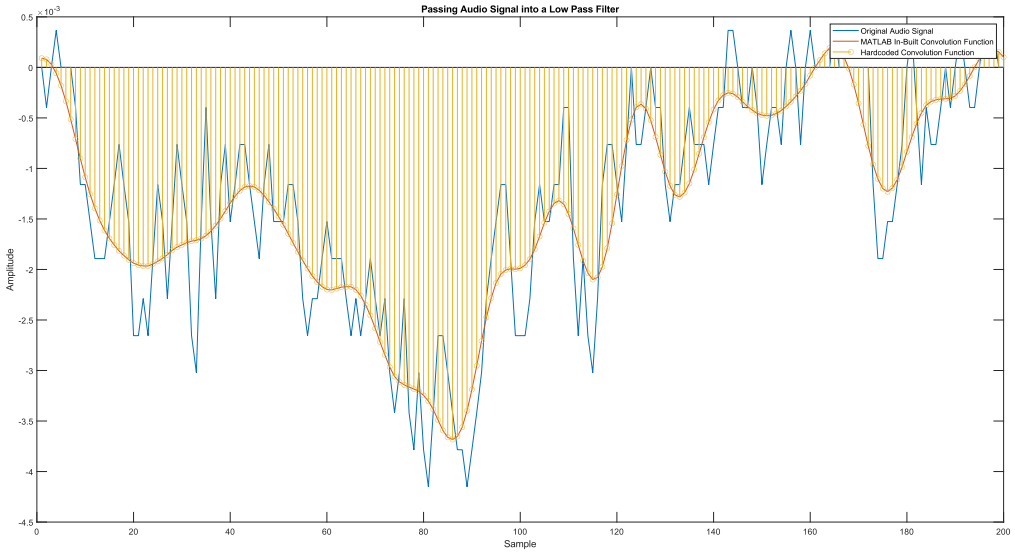

**Part Five:**

In part five, the students were tasked with taking the audio signal and passing it through a low pass filter implemented using

convolution. An ideal low pass filter in the frequency domain assumes the shape of a rectangle. In the time domain, the impulse response then becomes a sinc wave.

MATLAB was used convolve the audio signal with a portion of the sinc wave. In the frequency domain, this corresponds to multiplying the Fourier transform of the input signal by a square wave centered about f = 0 Hz, essentially cutting off any frequencies falling outside the bounds of the box thereby acting as a low pass filter.

*Figure 7: Using Two Convolution Methods for Implementing a LPF*




Lastly, the maximum difference between each of the three signal is shown below:

```
%finding the max difference between the signals
max(abs(y-yf))
max(abs(y-yfl))
max(abs(yf-yfl))
```

```
ans =

    0.3345


ans =

    0.3371


ans =

    0.0158
```

## Conclusion:

In this lab, the students gained important exposure implementing various digital signal processing techniques. This includes methods for sampling signals, manipulating audio files, and using convolution for implementing a low pass filter.

# Appendix:

## Appendix A-Part One Code:

```
close all
clear

%sampling frequency
Fs = 8000;
%tone frequency
f0 = 440;
%2 second signal duration,need twice as many samples as Fs
n = 0:Fs*2 - 1;
%vector for approximated square wave
x = 0;

%sampling frequency for "continous" sine wave
Fs_dense = Fs*10;
%2 second signal duration,need twice as many samples as Fs_dense
n_dense = 0:Fs_dense*2 - 1;
%vector for "continous" square wave
x_dense = 0;

%used for incrementing harmonics and plot number
plot_num = 1;
HARMONIC_FINAL = 13;

%using a for loop to add additional harmonics
for HARMONIC = 1:2:HARMONIC_FINAL
    %adding additional harmonics to sampled sqaure wave
    x = x + sin(2*pi*HARMONIC*(f0/Fs)*n)/HARMONIC;
    %adding additional harmonics to continous square wave
    x_dense = x_dense + sin(2*pi*HARMONIC*(f0/Fs_dense)*n_dense)/HARMONIC;
    %only adding harmonic if it odd or smaller than HARMONIC_FINAL = 13
    if HARMONIC == 1 ||HARMONIC == 5 || HARMONIC == 9 || HARMONIC == 13
        subplot(4, 1, plot_num)
        stem(n, x)
        hold on
        plot(n_dense/10, x_dense)
        xlabel('Sample')
        ylabel('Amplitude')
        legend('Sampled', 'Continous')
        xlim([0 200])
        %using if statment for helping put title on graphs
        if HARMONIC == 1
            title('Square Wave: 1st Harmonic')
        elseif HARMONIC == 5
            title('Square Wave: Adding 3rd and 5th Harmonic')
        elseif HARMONIC == 9
            title('Square Wave: Adding 7th and 9th Harmonic')
        elseif HARMONIC == 13
            title('Square Wave: Adding 11th and 13th Harmonic')
        end
        plot_num = plot_num + 1;
    end
    %playing each tone for 2 seconds
```

```
    sound(x, Fs);
    pause(2);
end
```

## Appendix B-Part Two Code:

```
close all
clear

%sampling frequency of 48000 Hz
Fs = 48000;
%tone frequency of 1000 Hz
f0 = 1000;
%2 second signal duration, need twice as many samples as Fs
n = 0:Fs*2 - 1;

%sampling frequency for "continuous" sine wave
Fs_dense = Fs*10;
%2 second signal duration, need twice as many samples as Fs_dense
n_dense = 0:Fs_dense*2 - 1;

%creating vectors to hold value of sine waves at different indexes
x = sin(2*pi*(f0/Fs)*n);
x_dense = sin(2*pi*(f0/Fs_dense)*n_dense);

%plotting sine wave sampled at base frequency
subplot(7,1,1);
stem(n,x);
hold on;
plot(n_dense/10, x_dense);
hold off;
title('Sampling Frequency (Hz) =', Fs);
xlabel('Samples')
ylabel('Amplitude')
legend('Sampled', 'Continous')
xlim([0 200]);

%playing first tone out loud.
sound(x, Fs);
pause(2)

%int for incrementing plots
num = 1;
for sample_div = 6:5:31
    %dividing sampling frequency by some divisor.
    Fs_sample_slow = Fs/sample_div;
    %vector to hold indexes under new slow sampling rate
    n_sample_slow = 0:Fs_sample_slow*2 - 1;
    %vector to hold value of sine wave using slow sampling rate
    x_sample_slow = sin(2*pi*(f0/Fs_sample_slow)*n_sample_slow);

    %incrementing plot number
    num = num + 1;
    %converting freq to int16 to help with naming titles
```

```
    sampling_freq = int16(Fs/sample_div);
    %plotting the slowly sampled wave
    subplot(7,1,num)
    stem(n_sample_slow*sample_div,x_sample_slow)
    hold on
    %plotting dense sine wave
    plot(n_dense/10, x_dense)
    hold off
    title('Sampling frequency (Hz) = ', sampling_freq)
    xlabel('Samples')
    ylabel('Amplitude')
    legend('Sampled', 'Continous')
    xlim([0 200])

    %playing slowly sampled sine wave aloud.
    sound(x_sample_slow, Fs)
    pause(2)
end
```

**Appendix C-Part Three Code:**

```
%tone frequency
f0 = 995;
%sampling frequency
Fs = 8000;
%2 second signal duration, need twice as many samples as Fs
n = 0:Fs*2 - 1;

%sampling frequency for "continous sine wave"
Fs_dense = Fs*10;
%2 second signal duration,need twice as many samples as Fs_dense
n_dense = 0:Fs_dense*2 - 1;

%using a for loop to multiply the frequencies by an integer
for freq_multiplier = 1:1:8
    %vectors for plotting sampled and continous sine wave
    %notice how freq_multiplier multiplies the tone freq
    x = sin(2*pi*freq_multiplier*(f0/Fs)*n);
    x_dense = sin(2*pi*freq_multiplier*(f0/Fs_dense)*n_dense);

    %plotting the sample sine wave and continous sine wave
    subplot(4,2,freq_multiplier)
    stem(n, x);
    hold on;
    plot(n_dense/10, x_dense);
    hold off;
    title('Frequency Multiplier =', freq_multiplier);
    xlabel('Sample')
    ylabel('Amplitude')
    legend('Sampled', 'Continous')
    xlim([0 40])

    %playing sampled signal out loud
    sound(x, Fs)
    pause(2)
```

```
end
```

## Appendix D-Part Four Code:

```
%reading in the audio file
[y, Fs] = audioread('good_news.wav');
%just keep left channel
%transpose to row vector
y = y(:,1)';
%normalize to max value of 1.0
y = y/max(y);

%calculating number of samples for audio signal and our sampled audio
n = 1:1:length(y);
%want to graph 1000 samples of signal
samples_of_audio = 1000;
%graphing the 50,000th to 51,000th sample
n_sample = 50000:1:50000+samples_of_audio;

%plotting
subplot(2,1,1)
plot(n,y)
title('Signal from Audio File')
ylabel('Amplitude')
xlabel('Sample')
xlim([0 length(y)])
subplot(2,1,2)
%want to plot 50,000th to 51,000th sample
plot(n_sample, y(50000:50000+samples_of_audio))
title('1000 Samples from Audio File')
ylabel('Amplitude')
xlabel('Sample')

%playing audio signal aloud
sound(y, Fs)
```

## Appendix E-Part Five Code:

```
close all
clear

%reading in the audio file
[y, Fs] = audioread('good_news.wav');
%just keep left channel
%transpose to row vector
y = y(:,1)';
%normalize to max value of 1.0
y = y/max(y);

%get lowpass filt coeffs in Num
load('filt_3300_48000_coeffs.mat')
%lowpass filter to 1KHz cutoff
```

```
yf = conv(y, Num, 'same');
%normalize to max value of 1.0
yf = yf / max(yf);


%convolution hard way
%declaring vector to hold inpulse response for low pass filter
impulse_resp = Num;
%finding length of impulse response
impulse_resp_len = length(impulse_resp);
num_padding = impulse_resp_len - 1;
%padding on both sides of signal to help with convolution shift
lft_padding = floor(num_padding/2);
rgt_padding = num_padding - lft_padding;
%filling the pads with placeholder zeros.
y_pad = [ zeros(1,lft_padding), y, zeros(1,rgt_padding) ];

%finding length of audio signal
NUM_SAMPS = length(y);
yf1 = zeros(1, NUM_SAMPS);

%performing the convolution sum
for n=1:NUM_SAMPS
   %value used to hold each system response at a particular time
   conv_value = 0;
   %want to convolve with sinc wave from it's -49th to 49th index
   for k=-49:1:49
       %MATLAB does not like negatives or 0s as indexes in arrays
       %must shift iterator to start at 1
       %There are 99 total samples
       %when k = -49, k_i = (99-1)/2+(-49)+1 = 1
       k_i = (impulse_resp_len-1)/2+k+1;
       %need additional room after audio signal for
       %impulse response to "exit" as it slides when convolving
       n_i = n + impulse_resp_len;
       %multiplying input signal by system response at that time
       conv_value = conv_value + impulse_resp(k_i)*y_pad(n_i-k_i);
   end
   %vector used to store convolution value at each instance of time
   yf1(n) = conv_value;
end

%finding the max difference between the signals
max(abs(y-yf))
max(abs(y-yf1))
max(abs(yf-yf1))

%plotting 200 samples of the audio sample
plot(y(1:200))
hold on
%plotting 200 samples of audio signal with MATLAB LPF
plot(yf(1:200))
hold on
%plotting 200 samples of audio signal with hardcoder convolution
stem(yf1(1:200))
hold off
```

```
xlim([0 200])
title('Passing Audio Signal into a Low Pass Filter')
xlabel('Sample')
ylabel('Amplitude')
legend('Original Audio Signal', 'MATLAB In-Built Convolution Function',
'Hardcoded Convolution Function')

%playing audio signals out loud
sound(y, Fs)
pause(7)
sound(yf, Fs)
pause(7)
sound(yf1, Fs)
```