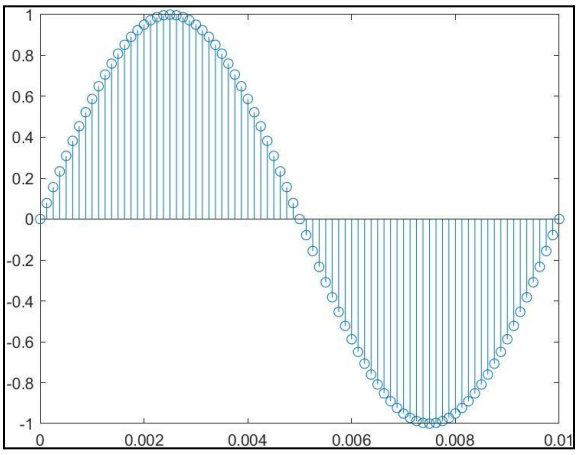
Signals & Systems - EE 3TP3
Lab #3

-

Stefan Tosti - Tostis - 400367761 - L08

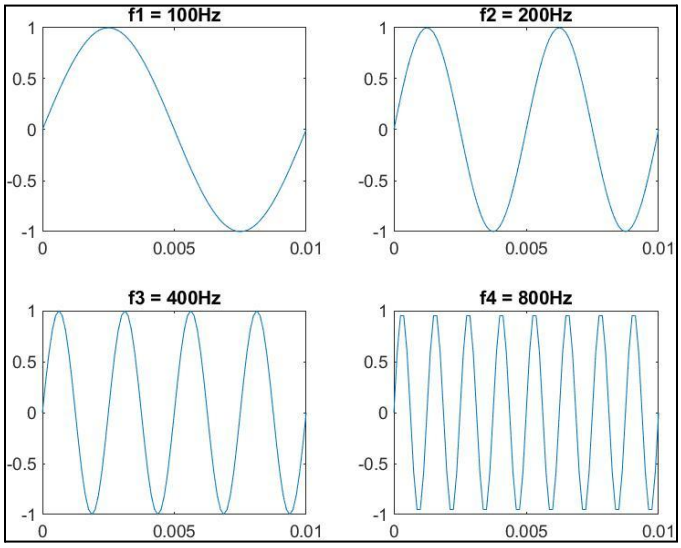
2023 - 11 - 17

Question 1A



The graph shown on the left was obtained from running the Matlab code in Step 1.1 of the lab manual. We can see that the function assumes a voice signal of the form $x(t) = sin(2\pi ft + \phi)$, which is then periodically sampled based on the defined sampling period, T_S, and plotted using the stem function in Matlab as the function, $x(nT_s) = sin(2\pi nf/f_s + \phi)$. This signal is a discretized version of the Sine wave over a single period.

Question 1B

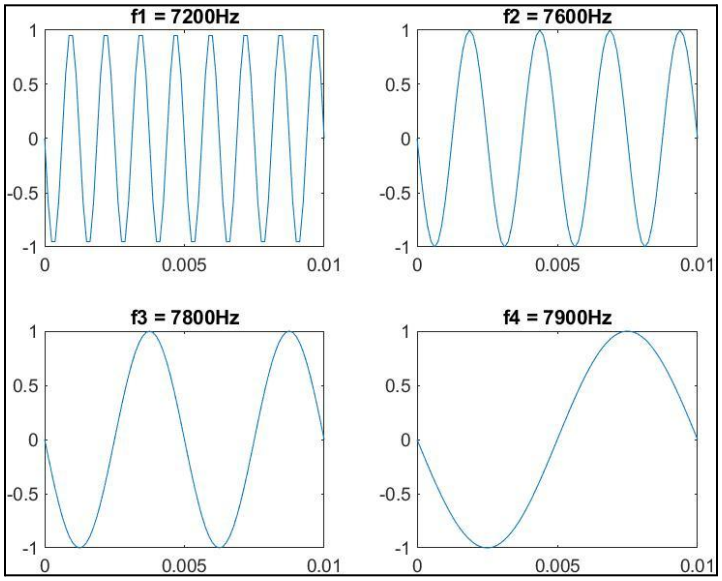


The graph on the left was generated using the Matlab code found below.

When the audio file is played, we can hear a humming noise that seems to change to a higher and higher pitch every 2 seconds. THis corresponds to us increasing the frequency of the signal every 2 seconds, since highest frequency corresponds to a higher pitch

```
% Define sinusoid frequencies as per the lab manual
f1 = 100;
f2 = 200;
f3 = 400;
f4 = 800;
% Sampling frequency and interval
fs = 8000;
Ts = 1/fs;
% Set time duration of plot, i.e., 10 msec.
tfinalplot = 10e-3;
% Make the time vector for the plot
nplot=0:Ts:tfinalplot;
% Play the spurt for 2 seconds
tfinal = 2;
t = 0: Ts : tfinal;

% f1 = 100Hz
subplot(2,2,1)
xnT1 = sin(2*pi*f1*t); % Sample the sinusoid
plot(nplot, xnT1(1:length(nplot)));
title('f1 = 100Hz')

% f2 = 200Hz
subplot(2,2,2)
xnT2 = sin(2*pi*f2*t); % Sample the sinusoid
plot(nplot, xnT2(1:length(nplot)));
title('f2 = 200Hz')

% f3 = 400Hz
subplot(2,2,3)
xnT3 = sin(2*pi*f3*t); % Sample the sinusoid
plot(nplot, xnT3(1:length(nplot)));
title('f3 = 400Hz')

% f4 = 800Hz
subplot(2,2,4)
xnT4 = sin(2*pi*f4*t); % Sample the sinusoid
plot(nplot, xnT4(1:length(nplot)));
title('f4 = 800Hz')

%export data
exportgraphics(gcf, 'Part1B.jpg');
audiowrite('Part1B.wav', cat(2, xnT1, xnT2, xnT3, xnT4), fs);
```

Question 1C



The graph on the left was generated with the below Matlab code

We can see that as we increase the frequency of the input sinusoid, the frequency of the output sinusoid decreases. It also follows that when listening to the audio file, we hear a pitch that decreases every 2 seconds rather than increases every 2 seconds. The reason that this is happening is because we did not change the sampling frequency from the last step. By the Nyquist rate, we know we should be sampling at 2 times the highest present frequency, however since we are still sampling at 8000Hz, we are unable to preserve the frequency data from the waveform in this case. This results in aliasing occurring, and the output waveforms we get are completely different than we would expect ideally.

```
% Define sinusoid frequencies as per the lab manual
f1 = 7200;
f2 = 7600;
f3 = 7800;
f4 = 7900;
% Sampling frequency and interval
fs = 8000;
Ts = 1/fs;
% Set time duration of plot, i.e., 10 msec.
tfinalplot = 10e-3;
% Make the time vector for the plot
nplot=0:Ts:tfinalplot;
% Play the spurt for 2 seconds
tfinal = 2;
t = 0: Ts : tfinal;

% f1 = 7200Hz
subplot(2,2,1)
xnT1 = sin(2*pi*f1*t); % Sample the sinusoid
plot(nplot, xnT1(1:length(nplot)));
title('f1 = 7200Hz')

% f2 = 7600Hz
subplot(2,2,2)
xnT2 = sin(2*pi*f2*t); % Sample the sinusoid
plot(nplot, xnT2(1:length(nplot)));
title('f2 = 7600Hz')

% f3 = 7800Hz
subplot(2,2,3)
xnT3 = sin(2*pi*f3*t); % Sample the sinusoid
plot(nplot, xnT3(1:length(nplot)));
title('f3 = 7800Hz')

% f4 = 7900Hz
subplot(2,2,4)
xnT4 = sin(2*pi*f4*t); % Sample the sinusoid
plot(nplot, xnT4(1:length(nplot)));
title('f4 = 7900Hz')

%export data
exportgraphics(gcf, 'Part1C.jpg');
audiowrite('Part1C.wav', cat(2, xnT1, xnT2, xnT3, xnT4), fs);
```

Question 1D

Without the use of anti-aliasing pre-filtering the sampling rate of telephone systems would not be very reliable. The sampling rate used is based on the estimate that all human voices will fall under 4KHz frequency, and thus incoming data is sampled at 8KHz. In order to make this sampling rate viable, pre-filtering is used to filter out any signal that is above the 3.5KHz range. If pre-filtering were not used, then any signal that the phone system picks up would be sampled with a sampling rate of 4KHz. As we saw in the previous question, when we sample a signal with a rate that is too small, significant distortion occurs, and transmitted sounds are not true to what they should ideally sound like.

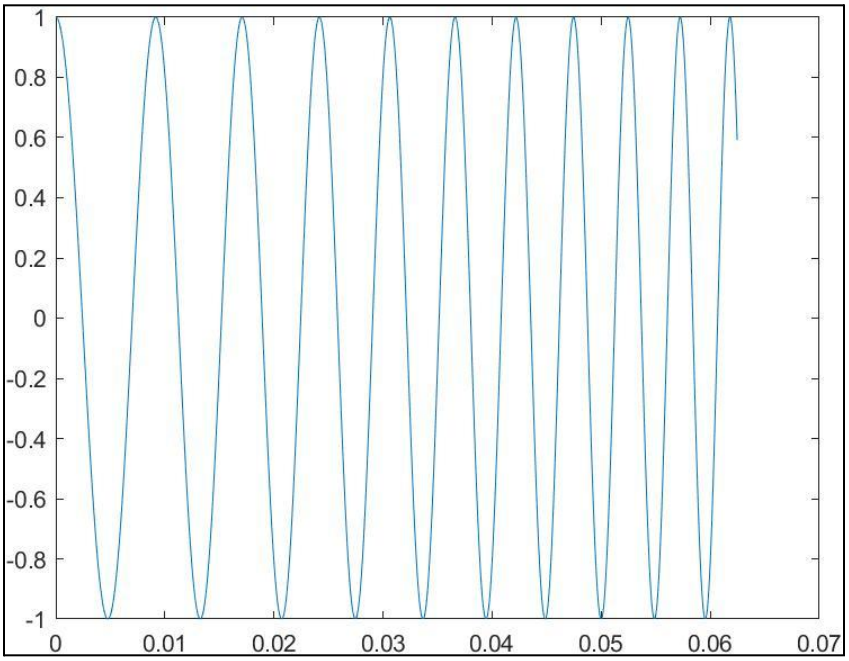
In the above experiments, we can imagine that all of the above signals might be sent into a phone system at the same time. Without pre-filtering, all the signals would be sampled with an 8KHz sampling rate, thus resulting in distortion from the 7200Hz, 7600Hz, 7800Hz, and 7900Hz signals. With pre-filtering, our system would filter these frequencies out and we would just be left with 100Hz, 200Hz, 400Hz, and 800Hz signals which can be accurately sampled with our sampling rate.

Question 2A

The below Matlab code was used to generate the 3 following graphs

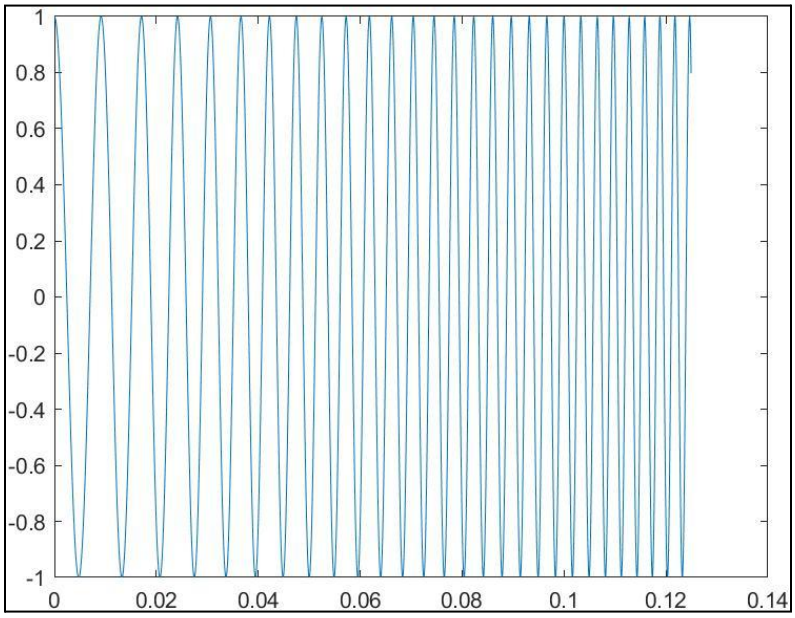
```
% Define the paramaters as per the lab manual
f1 = 100;
u = 2000;
fs = 32000;
ts = 1/fs;
NumOfSamples = 2000;
t = 0: ts : 8;

% Define our function, C(t)
cnT = cos(pi*u*t.^2 + 2*pi*f*t);

% Plot our function
plot(t(1:NumOfSamples), cnT(1:NumOfSamples));

% Export audio and graph
audiowrite('Part2A.wav', cnT', fs);
exportgraphics(gcf, 'Part2A.jpg');
```

By setting $f_s = 32000\ Hz$ in the code shown for question 2A, we obtain the following graph…



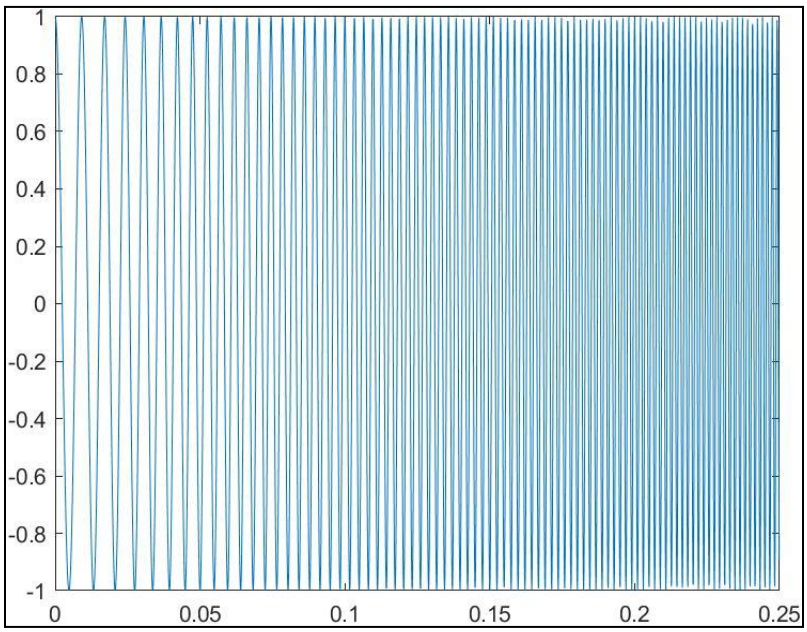
In the above case, the audio file sounds like a pitch that increases for the entire duration of the clip.

By setting $f_s = 16000\ Hz$ in the code shown for question 2A, we obtain the following graph…



In the above case, the audio file sounds like a pitch that reaches its highest pitch halfway through the audio sample, and then slowly decreases until reaching its original pitch at the end of the sample.

By setting $f_s = 8000\ Hz$ in the code shown for question 2A, we obtain the following graph…



In the above audio clip, the pitch sounds like two cycles of the 16,000 Hz clip. What this means is that instead of starting off low, peaking in the middle, and finishing where it starts, the audio clip peaks at ¼ of the way through and ¾ of the way through, and is at its lowest points at the beginning, middle, and end of the clip.

We can explain why the above is happening by looking further into the code that we have written. The chirp function that we have implemented has frequency that linearly increases according to $\mu t + f_1$. Since we have defined $f_1 = 100Hz \; and \; \mu = 2000$, then for the 8 second period that we are recording, our frequency spans values from 100Hz to 16,100Hz. Now that we know this, we can see that sampling at 32KHz satisfies the Nyuquist criteria (roughly, we should ideally be sampling at 32.2KHz), but 16KHz and 8KHz do not. We can combine this knowledge with what we learned in Question 1C, in that when we under-sample our signal the input frequency and output frequency are inversely proportional, to explain why we are hearing dips in the audio pitch for our 16KHz and 8KHz samples. Furthermore, since the 8KHz sample is sampling at half the rate of the 16KHz sample, we hear double the number of pitch dips in the 8KHz sample when compared to the 16KHz sample (I.e. one full cycle for 16KHz and 2 full cycles for 8KHz).

A telephone network that includes the anti-alias filtering, this aliasing that we are seeing would not occur. This is because the telephone network would be able to filter out everything outside of its acceptable sample rate, thus avoiding aliasing and removing excessively high frequencies.

By experimenting with the various values in our program, I found the following…
f_s: Controls how high the maximum pitch is, and the duration for how long this maximum pitch is heard
μ: Controls how often the waveform oscillates between high and low pitch
f_1: Controls the speed at which the output signal increases and decreases