ELECENG 3TP3

Lab 3: Aliasing in Signal Sampling

Instructor: Dr. Kiruba

Jasmine Dosanjh – dosanj5 – 400531879

Warisha Noushad – noushadw – 400519903

December 2nd, 2025

Question 1: Aliasing in the Telephone System

Part 1.

The MATLAB code below takes the analog (continuous-time) sinusoid voice input:

$$x(t) = \sin(2\pi f t + \phi)$$

and samples it at periodic intervals of the sampling period, creating the discrete-time sampled signal:

$$x[n] = x(nT_s) = \sin(2\pi n f/f_s + \phi)$$

This script initially assigns a sinusoid $x(t) = \sin(2\pi f t)$ with frequency $f = 100$ Hz. To create the sampled signal, a sampling rate $f_s = 8000\ Hz$ is defined, with the corresponding sampling period $T_s = \frac{1}{f_s} = \frac{1}{8000} = 0.000125\ s$. Samples are taken at periodic intervals of the sampling period T_s. This is implemented through the vector `nplot`, which goes from 0 to 10 ms in steps of T_s: $nT_s = 0, 0.000125, 0.00025, \ldots, 0.01$. Lastly, the sampled sinusoid $x(nT_s)$ is plotted using the `stem` command and exported.

Listing 1: Part 1

```
%%Question 1
% Do a plot of a sampled sinusoid with frequency f = 100 Hz
f = 100
% 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;
% Sample the sinusoid.
xnT = sin(2*pi*f*nplot);
% Make the plot
stem(nplot, xnT);
text(6.5e-3,0.9, 'Jasmine Dosanjh - 400531879', 'FontSize',8)
text(6.5e-3,0.8, 'Warisha Noushad - 400519903', 'FontSize',8)
xlabel('Time (s)');
ylabel('Amplitude');
title('Sampled Sinusoid at 100 Hz');
%Export the graph
exportgraphics(gcf, "Q1.jpg")
```

Formatted: French (France)

The output discrete-time sinusoid is shown below. It exists for a duration of 10 ms, with an amplitude of 1, and no phase shift.

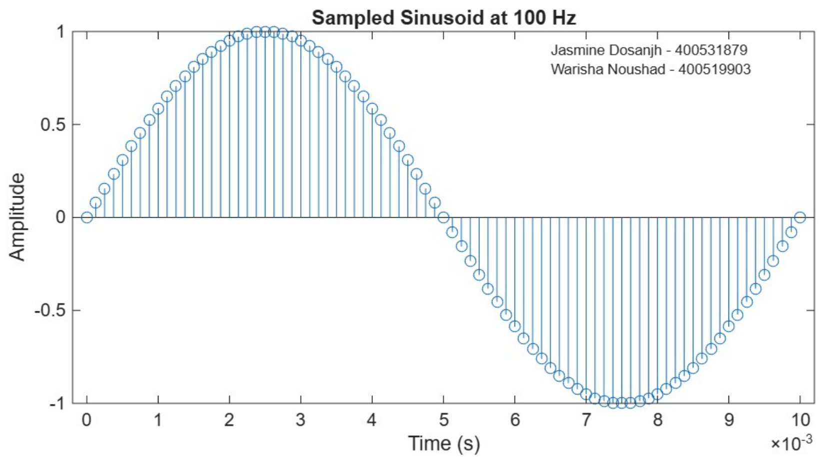

Figure 1. Discrete-Time Sinusoid at frequency 100 Hz

Part 2.

The MATLAB code in Listing 2 observes the effect on the discrete-time signal $x[n]$ from Part 1 at varying frequencies. Lines 1-10 remain the same as Part 1, to create $x[n]$ and generate the short-time vector `nplot`. A second, longer time vector, `nsound,` is created to generate a 2-second audio tone for each frequency.

The sinusoid is then sampled at four frequencies, $f = 100, 200, 400, 800\ Hz$. For each frequency, a discrete-time signal x[n] is created. These four tone segments are concatenated sequentially into a single 8-second vector `sound_created`. All four sinusoids are plotted using the `subplot` command and exported.

Listing 2: Part 2

```
 1. %% Question 2
 2. % Use sinusoid frequency f = 300 Hz
 3. %f = 100
 4. % Sampling frequency and interval
 5. fs = 8000;
 6. Ts = 1/fs;
 7. % Set time duration of plot, i.e., 10 msec.
 8. tfinalplot = 10e-3;
 9. % Make the time vector for the plot
10. nplot=0:Ts:tfinalplot;
11. % Make the time vector for replayed sound spurt
12. % Play the spurt for 2 seconds
```

```matlab
tfinal = 2;
nsound=0:Ts:tfinal;
%Frequency for sin
f1 = 100;
f2 = 200;
f3 = 400;
f4 = 800;
% Sample the sinusoid.
xnT1 = sin(2*pi*f1*nsound);
xnT2 = sin(2*pi*f2*nsound);
xnT3 = sin(2*pi*f3*nsound);
xnT4 = sin(2*pi*f4*nsound);
%Plot Making
figure;
sgtitle('Warisha Noushad - 400519903 & Jasmine Dosanjh - 400531879');
%For 100Hz Plot
subplot(2,2,1);
plot(nplot, xnT1(1:length(nplot)));
title('Frequency at 100Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%For 200Hz Plot
subplot(2,2,2);
plot(nplot, xnT2(1:length(nplot)));
title('Frequency at 200Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%For 400Hz Plot
subplot(2,2,3);
plot(nplot, xnT3(1:length(nplot)));
title('Frequency at 400Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%For 800Hz Plot
subplot(2,2,4);
plot(nplot, xnT4(1:length(nplot)));
title('Frequency at 800Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%Export the graph
exportgraphics(gcf, "Q2.jpg")
sound_created = [xnT1,xnT2,xnT3,xnT4];
% Save xnT as a wav sound file, soundfile.wav.
audiowrite('Lab3_Q2_soundwave.wav', sound_created, fs);
```

The output discrete-time sinusoids are shown below:

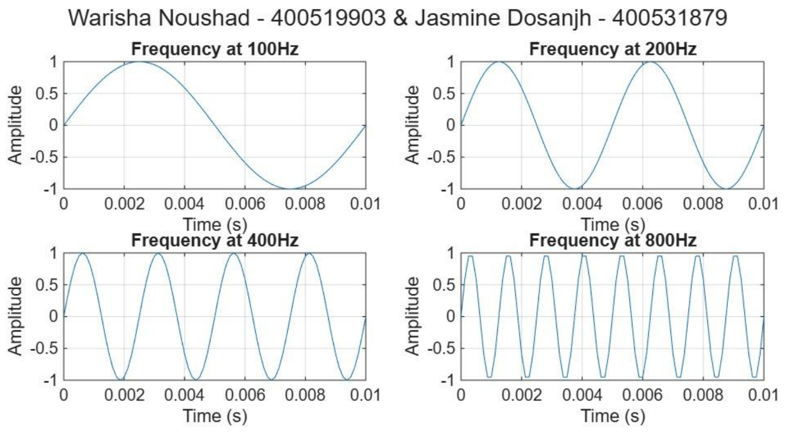

Figure 2. Discrete-Time Sinusoid at Varying Frequencies

The output sound file is 8 seconds long, with each 2-second segment playing a sinusoid of increasing input frequency in the order $f = 100, 200, 400, 800\ Hz$. As the input frequency increases, so does the output frequency, as the tone becomes higher-pitched and less deep/lower-pitched.

Part 3.

The MATLAB code in Listing 3 again observes the effect on the discrete-time signal $x[n]$ from Part 1 at varying frequencies. The code remains the same as Part 2, but is sampled at four new frequencies, $f = 7200, 7600, 7800, 7900\ Hz$.

Listing 3: Part 3

```
%% Question 3
% 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;
% Make the time vector for replayed sound spurt
% Play the spurt for 2 seconds
tfinal = 2;
nsound=0:Ts:tfinal;
%Frequency for sin
f1 = 7200;
f2 = 7600;
f3 = 7800;
f4 = 7900;
```

```
% Sample the sinusoid.
xnT1 = sin(2*pi*f1*nsound);
xnT2 = sin(2*pi*f2*nsound);
xnT3 = sin(2*pi*f3*nsound);
xnT4 = sin(2*pi*f4*nsound);
%Plot Making
figure;
sgtitle('Warisha Noushad - 400519903 & Jasmine Dosanjh - 400531879');
%For 100Hz Plot
subplot(2,2,1);
plot(nplot, xnT1(1:length(nplot)));
title('Frequency at 7200Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%For 200Hz Plot
subplot(2,2,2);
plot(nplot, xnT2(1:length(nplot)));
title('Frequency at 7600Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%For 400Hz Plot
subplot(2,2,3);
plot(nplot, xnT3(1:length(nplot)));
title('Frequency at 7800Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%For 800Hz Plot
subplot(2,2,4);
plot(nplot, xnT4(1:length(nplot)));
title('Frequency at 7900Hz');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%Export the graph
exportgraphics(gcf, "Q3.jpg")
sound_created = [xnT1,xnT2,xnT3,xnT4];
% Save xnT as a wav sound file, soundfile.wav.
audiowrite('Lab3_Q3_soundwave.wav', sound_created, fs);
```

The output discrete-time sinusoids are shown in Figure 3. These waveforms resemble the ones from Part 2 but appear in reverse order and with the opposite progression of frequency. Instead of having a higher frequency as the input frequency increases, the plotted signals output a lower frequency.

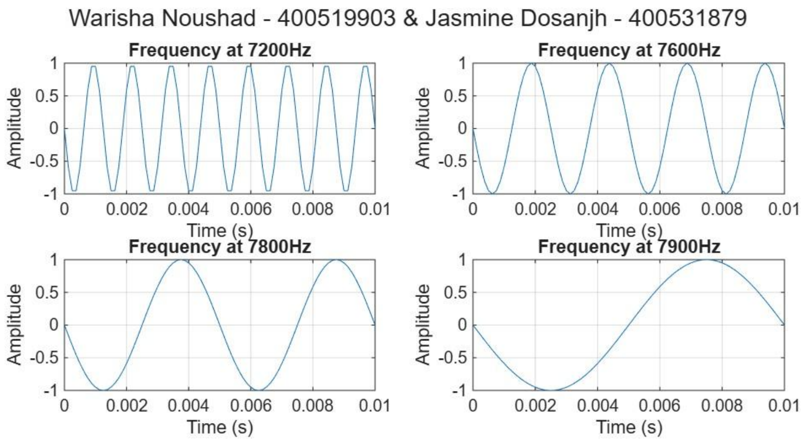

Figure 3. Discrete-Time Sinusoid at Higher Varying Frequencies

For the input frequencies $f = 7200, 7600, 7800, 7900\ Hz$. The output behaves unexpectedly. Instead of hearing higher-pitched tones when the input frequency increases, the output frequencies decrease, producing deeper, lower-pitched sounds. This is the opposite of what was observed in Part 2, where the input frequencies $f = 100, 200, 400, 800\ Hz$ were all below the Nyquist limit, and therefore reproduced correctly (no alias).

This reversal occurs because all frequencies in Part 3 are above the Nyquist limit:

$$f_{Nyquist} = \frac{f_s}{2} = 4000\ Hz$$

Meaning they cannot be represented without aliasing. Since the input frequencies are close to the sampling frequency $f_s = 8000\ Hz$, the sampled system wraps these high frequencies back into the valid range [0, 4000] Hz. These folding produces alias frequencies, calculated as:

$$|7200 - 8000| = 800\ Hz$$

$$|7600 - 8000| = 400\ Hz$$

$$|7800 - 8000| = 200\ Hz$$

$$|7900 - 8000| = 100\ Hz$$

Thus, although the input frequencies increase, their aliases move in the opposite direction, decreasing from 800 Hz down to 100 Hz. These are the same frequencies from Part 2, but in reverse. This is why the output sound becomes progressively deeper and lower in pitch.

Part 4.

The performance of the telephone system would degrade significantly if anti-aliasing pre-filtering were not used, as sounds above 4000 Hz that enter the system would not be correctly replicated. As seen in Part 3, the aliasing would convert high-frequency input signals into a low-frequency output. This would cause distortions at the receiving end of the telephone, as the original input frequency is altered.

This filtering prevents these negative effects by removing all frequency components above 3.5 kHz, so that the signal contains no frequency above the Nyquist limit of 4 kHz. This ensures that significant aliasing does not occur, meaning that the sound at the receiving end will exhibit its expected behavior, as higher input frequencies will be transmitted at higher output frequencies.

If anti-aliasing filtering were applied in these experiments, the high-frequency signals from Part 3 ($f = 7200, 7600, 7800, 7900\ Hz$) would be filtered out entirely before sampling, since they lie above the filter cutoff of 3.5 kHz. As a result, none of these frequencies would reach the sampler, and the alias frequencies observed earlier would never appear. The output would therefore contain no audible noise (silence) for all four of the inputs.

Question 2: Aliasing of a Frequency Chirp Signal

Part 1.

The frequency-chirp signal used in this Question is defined as,

$$c(t) = \cos\left(\pi\mu t^2 + 2\pi f_1 t + \phi\right)$$

whose instantaneous frequency is,

$$f(t) = \mu t + f_1$$

In the script, the frequency is set to $f_1 = 100$ Hz and the chirp rate $\mu = 2000$. A high sampling rate of $f_s = 32000\ Hz$ is chosen to ensure accurate representation of the chirp. The sampling period is $T_s = \frac{1}{f_s} = 3.125 \times 10^{-5}\ s$. Two time vectors are created, `t_short` which produces the first 2000 samples of the chirp and `t_long` which generates the full 8-second chirp for audio playback. The chirp is then generated using `chirp.` Finally, the complete 8-second chirp is exported and plotted as a graph.

Listing 4: Part 1

```
%% Question 5
% Frequency for sin
f5 = 100; % New frequency for Question 5
mu = 2000;
%Sampling frequency
fs = 32000; %Sampling rate
Ts = 1/fs; %Sampling Period
%Plot the first 2000 samples
nplot = 2000;
t_short = (0:nplot-1)*Ts;
%Plot for 8 second signal
t = 8;
t_long = 0:Ts:t;
%Create the chirp signal c(t)
chirp = cos(pi*mu*(t_long.^2) + 2*pi*f5*t_long);
% Plot the 2000 samples
figure;
plot(t_short, chirp(1:nplot));
title('Aliasing of Frequency Chirp Signal Q1');
text(0.0605,-0.5, 'Jasmine Dosanjh', 'FontSize',8)
text(0.0605,-0.6, '400531879', 'FontSize',8)
text(0.0605,-0.7, 'Warisha Noushad', 'FontSize',8)
text(0.0605,-0.8, '400519903', 'FontSize',8)
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%Export graph
exportgraphics(gcf, "Q5.jpg")
%Export audio
audiowrite('Lab3_Q5_soundwave.wav', chirp, fs);
```

Formatted: French (France)

The audio file began with a high frequency signal that increased until it could no longer be heard. The output signal is shown in Figure 4. It plots the first 2000 samples and its frequency increases with time (the period decreases). This graph follows the same trend as the audio clip.

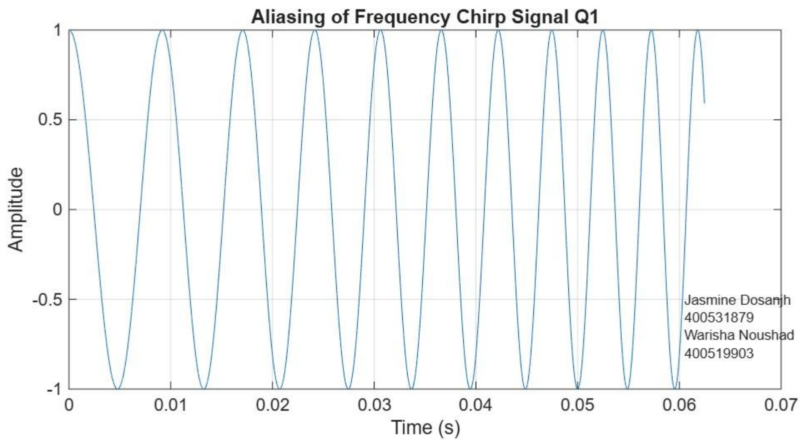

Figure 4. Aliasing of Frequency-Chirp Signal f=100 Hz

Part 2.

The MATLAB code in Part 2 is identical to Part 1, but with the sampling frequency f_s decreased from 32 kHz twice to observe the effects on the signal. Listing 5 is reduced by a factor of half where $f_s = 16\ kHz$. Listing 6 is reduced by a half again where $f_s = 8\ kHz$, which indicates what would happen if this signal were sent through the digital telephone network without anti-aliasing pre-filtering.

Listing 5: Question 2 for 16 kHz

```
%% Question 5 for 16KHz
% Frequency for sin
f5 = 100; % New frequency for Question 5
mu = 2000;
%Sampling frequency
fs = 16000; %Sampling rate
Ts = 1/fs; %Sampling Period
%Plot the first 2000 samples
nplot = 2000;
t_short = (0:nplot-1)*Ts;
%Plot for 8 second signal
t = 8;
t_long = 0:Ts:t;
%Create the chirp signal c(t)
chirp = cos(pi*mu*(t_long.^2) + 2*pi*f5*t_long);
% Plot the 2000 samples
figure;
plot(t_short, chirp(1:nplot));
```

```
title('Aliasing of Frequency Chirp Signal at 16KHz');
text(0.122,-0.5, 'Jasmine Dosanjh', 'FontSize',8)
text(0.122,-0.6, '400531879', 'FontSize',8)
text(0.122,-0.7, 'Warisha Noushad', 'FontSize',8)
text(0.122,-0.8, '400519903', 'FontSize',8)
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%Export graph
exportgraphics(gcf, "Q5(16KHz).jpg");
%Export audio
audiowrite('Lab3_Q5(16kHz)_soundwave.wav', chirp, fs);
```

Formatted: French (France)

Listing 6: Question 2 for 8 kHz

```
%% Question 5 (8KHz)
% Frequency for sin
f5 = 100; % New frequency for Question 5
mu = 2000;
%Sampling frequency
fs = 8000; %Sampling rate
Ts = 1/fs; %Sampling Period
%Plot the first 2000 samples
nplot = 2000;
t_short = (0:nplot-1)*Ts;
%Plot for 8 second signal
t = 8;
t_long = 0:Ts:t;
%Create the chirp signal c(t)
chirp = cos(pi*mu*(t_long.^2) + 2*pi*f5*t_long);
% Plot the 2000 samples
figure;
plot(t_short, chirp(1:nplot));
title('Aliasing of Frequency Chirp Signal at 8KHz');
text(0.001,0.5, 'Jasmine Dosanjh', 'FontSize',8)
text(0.001,0.6, '400531879', 'FontSize',8)
text(0.001,0.7, 'Warisha Noushad', 'FontSize',8)
text(0.001,0.8, '400519903', 'FontSize',8)
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%Export graph
exportgraphics(gcf, "Q5(8KHz).jpg");
%Export audio
audiowrite('Lab3_Q5(8KHz)_soundwave.wav', chirp, fs);
```

Formatted: French (France)

The output signals are shown below:

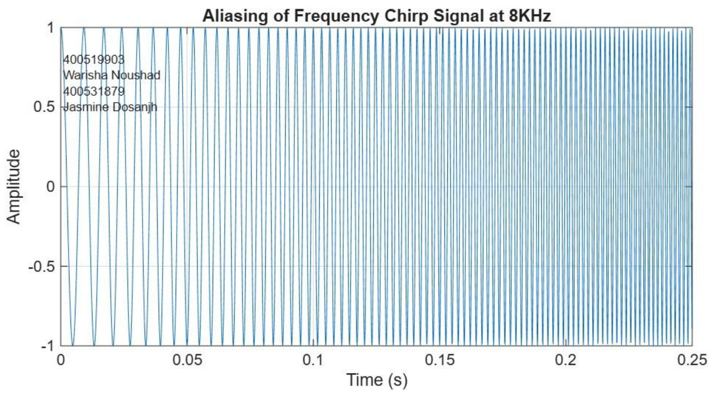

Figure 5. Aliasing of Frequency-Chirp Signal f=8 kHz

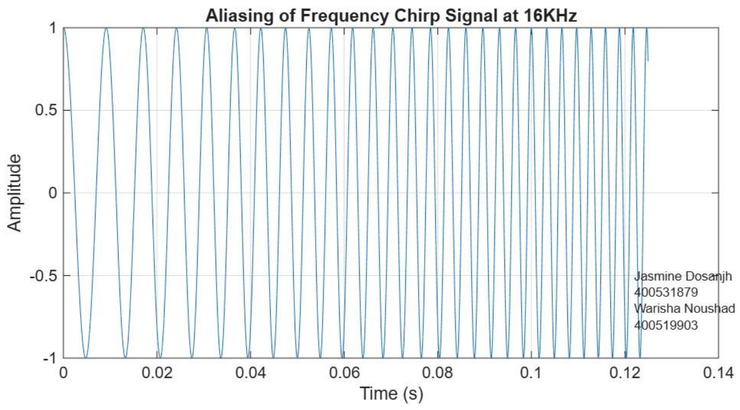

Figure 6. Aliasing of Frequency-Chirp Signal f=16 kHz

For a frequency of 16 kHz, the first half of the audio file behaved identically to the 32 kHz audio as it began with a high frequency signal that increased but took half the original time to become inaudible. For the remaining half the sound reversed, and its frequency decreased back down again until it reached the beginning frequency. Whereas for a frequency of 8 kHz the audio file repeated the behaviour of the 16 kHz audio but at twice the rate for the first half of the audio file. Then it repeats for the remaining half.

In a telephone connection that includes anti-aliasing filtering, the high-frequency signals above the Nyquist limit would be filtered out entirely before sampling. This would produce a signal that begins with a high frequency signal and increases until it surpasses 3.5 kHz, becoming inaudible. As a result, there will be no component of the frequency that decreases as seen previously.

This final part consists of experimenting with other frequency values f_1, f_s and chirp rates μ, and observing their effects on the audio signal. Recall that the instantaneous frequency of the chirp is,

$$f(t) = \mu t + f_1$$

Increasing f_1 causes the audio to begin at a higher frequency. As f_1 becomes large enough that $f(t)$ exceeds the Nyquist frequency, the signal aliases and the beginning of the audio jumps to a lower pitch. Conversely, decreasing f_1 shifts the chirp downward, causing the signal to begin at a lower frequency. This behaviour follows directly from the linear dependence of instantaneous frequency on f_1.

As the sampling rate f_s decreases, the output behavior is identical to Part 1. It begins with a high-frequency signal that increases and decreases, with this pattern repeating multiple times. This behaviour is due to aliasing, as the output signal distorts as f_s decreases below the Nyquist sampling rate of the original signal. As f_s As the increases, the output sounds more like a true chirp: smooth, continuous, with fewer repeats and a gradual decrease throughout. This behavior is due to the sampling frequency being high enough to reproduce the input signal, causing less aliasing.

As the chirp rate μ decreases, the sound alternates from high to low frequencies at a slower rate. When μ increases, the frequency signal has a higher rate of increase and decrease, causing the signal to repeat itself more. This is due to μ being the slope (rate of change) of the instantaneous frequency of the signal.