Mamadou Kaba

27070179

ELEC 342 CN-X

25/07/2024

01/08/2024

Nadia Abdolkhani

Objective

This lab aims to explore the use of MATLAB functions and investigate the effect of sampling rate on the Fourier transform of a signal. Through this experiment, we will define and utilize user-defined functions in MATLAB to perform various tasks, including implementing the Sampling Theorem. By examining the impact of different sampling rates on signal reconstruction, we aim to understand the critical role of sampling in digital signal processing.

Theory

User-Defined Functions in MATLAB

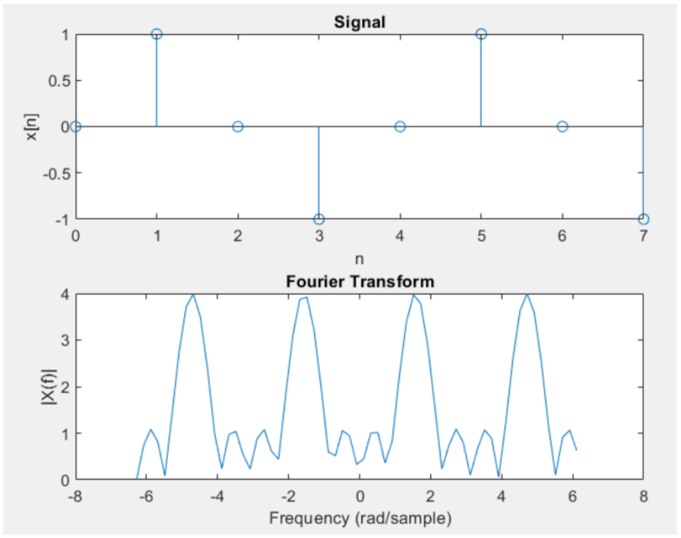
MATLAB allows users to create custom functions, which can be called from within other scripts or functions. This is essential for modular programming and code reuse. Functions in MATLAB are defined using the keyword function, followed by the output variable(s), the function name, and the input variable(s). For example, the function inc(x) increments the value of x by 1. Functions can handle scalar, vector, and matrix inputs, and MATLAB supports both pass-by-value and pass-by-reference mechanisms, depending on whether the function modifies the input argument.

Sampling Theorem

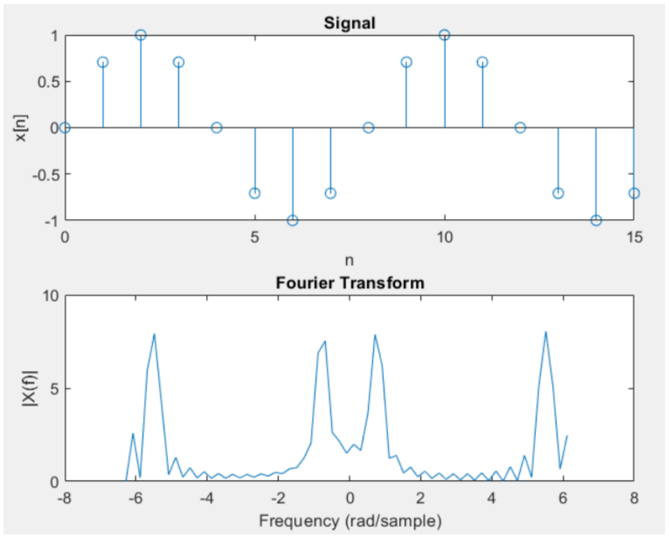
The Sampling Theorem, or Nyquist-Shannon Sampling Theorem, states that a continuous signal can be completely represented by its samples and fully reconstructed if it is sampled at a rate at least twice the highest frequency component present in the signal (the Nyquist rate). Sampling below this rate leads to aliasing, where different frequency components become indistinguishable. In this lab, we will examine the effect of different sampling rates on a sinusoidal signal and observe the changes in its Fourier transform.

Results/Discussion

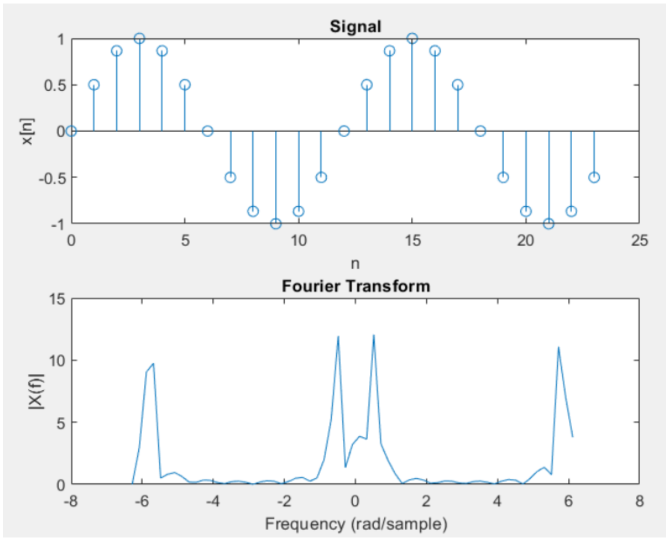
Question 1 a)



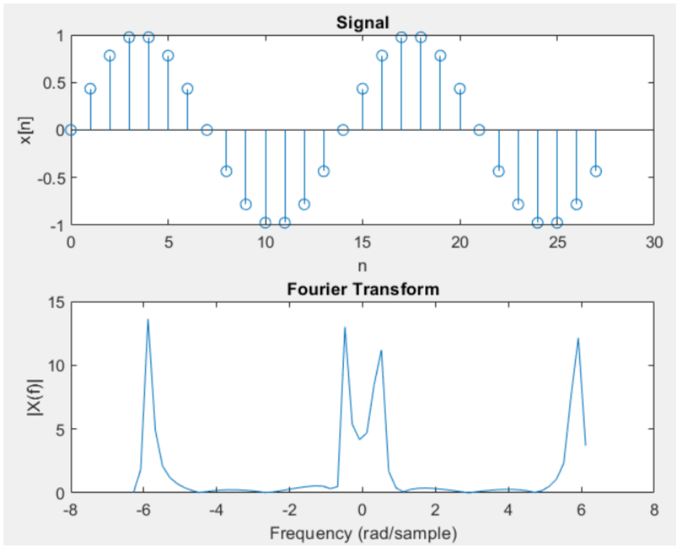
Sampling rate = 2



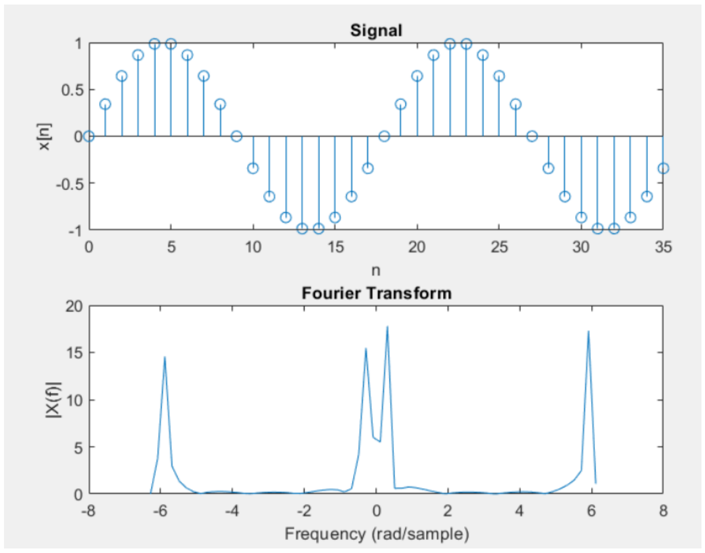
Sampling rate = 4

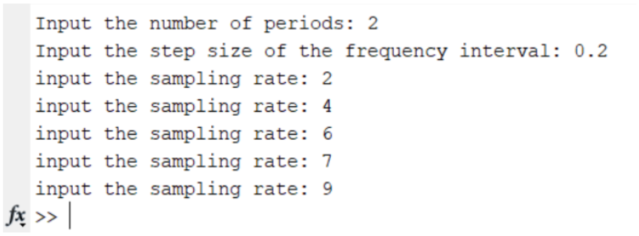


Sampling rate = 6



Sampling rate = 7





```
Input the number of periods: 2
Input the step size of the frequency interval: 0.2
input the sampling rate: 2
input the sampling rate: 4
input the sampling rate: 6
input the sampling rate: 7
input the sampling rate: 9
fx >> |
```

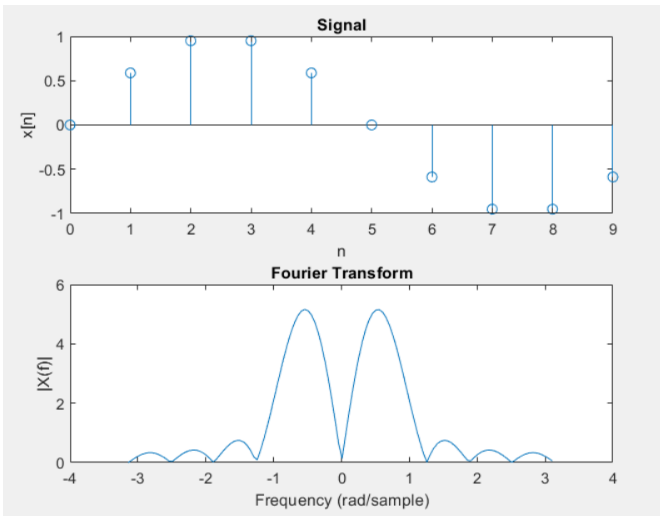
Sampling rate = 9

At a sampling rate of 2, the signal is sparsely sampled, leading to a less smooth sinusoidal wave. The corresponding Fourier Transform shows more spread and less distinct peaks due to undersampling. Increasing the sampling rate to 4 results in a smoother signal compared to the previous rate, but it still lacks sufficient resolution. The frequency representation at this rate shows more prominent peaks, indicating better frequency resolution but still some aliasing effects.

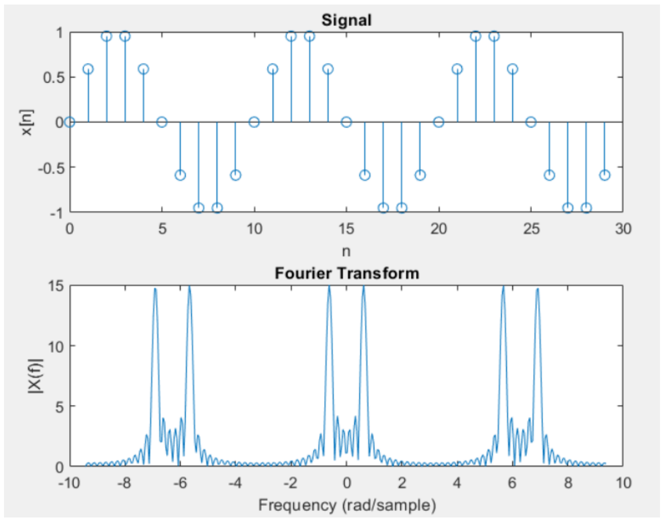
With a sampling rate of 6, the sinusoidal signal becomes even smoother, with more data points capturing the waveform accurately. The frequency spectrum shows clearer peaks, reduced aliasing, and better-defined frequency components. At a sampling rate of 7, the waveform is well-resolved with minor distortions, and the peaks in the frequency spectrum are distinct, indicating a good representation of the signal's frequency content. Finally, at a sampling rate of 9, the sinusoidal signal is very smooth and accurately captures the waveform. The frequency spectrum at this rate shows clear and sharp peaks, indicating a high-quality representation of the frequency content.

We can say that, increasing the sampling rate improves the resolution and accuracy of both the time-domain signal and its frequency-domain representation. Higher sampling rates lead to smoother signals and clearer frequency spectra, reducing aliasing and providing a more accurate depiction of the original continuous-time signal.

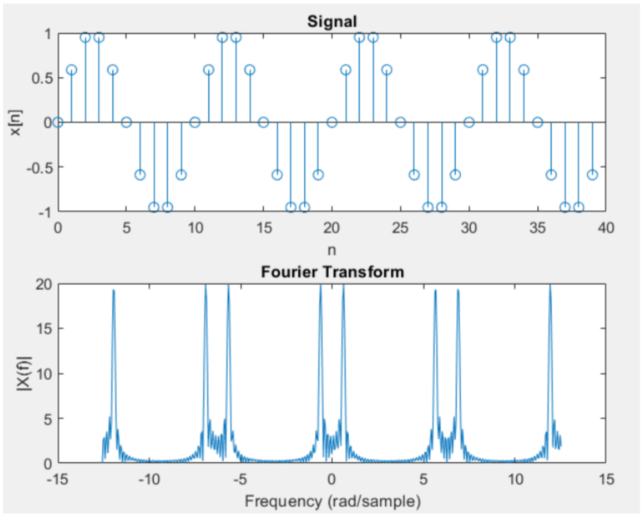
Question 1 b)



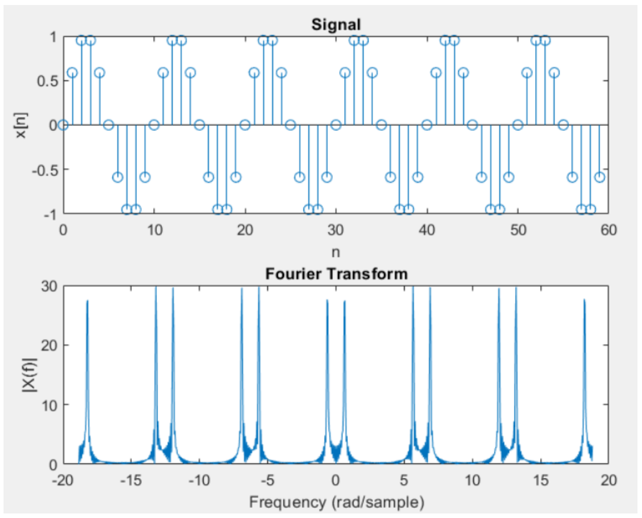
Window size = 1



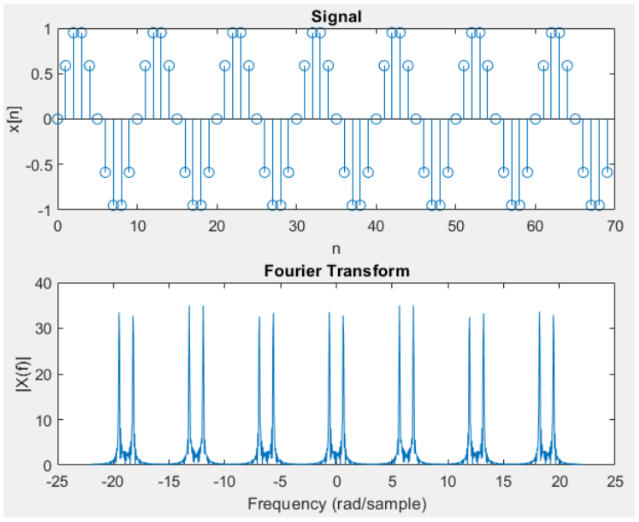
Window size = 3



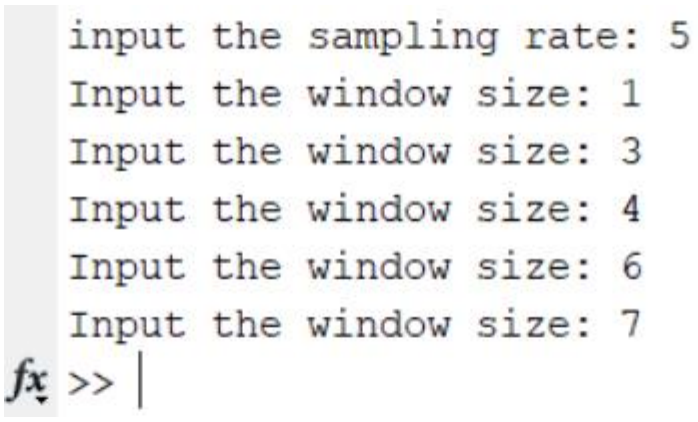
Window size = 4



Window size = 6



Window size = 7



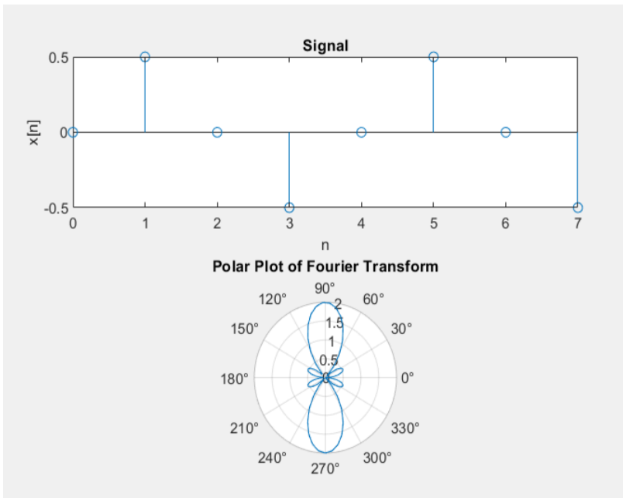
```
input the sampling rate: 5
Input the window size: 1
Input the window size: 3
Input the window size: 4
Input the window size: 6
Input the window size: 7
fx >> |
```

The input signal x[n]=sin(2πn/N) is analyzed using different window sizes while maintaining a constant sampling rate of 5. The window sizes varied as 1, 3, 4, 6, and 7, significantly impacting the number of samples and the Fourier Transform's resolution. With a window size of 1, the signal captures only a few samples of the sine wave, leading to broad peaks in the Fourier Transform, indicating poor frequency resolution. As the window size increases to 3 and 4, the signal encompasses more cycles, resulting in sharper peaks in the Fourier Transform. This demonstrates better frequency resolution and separation of components.

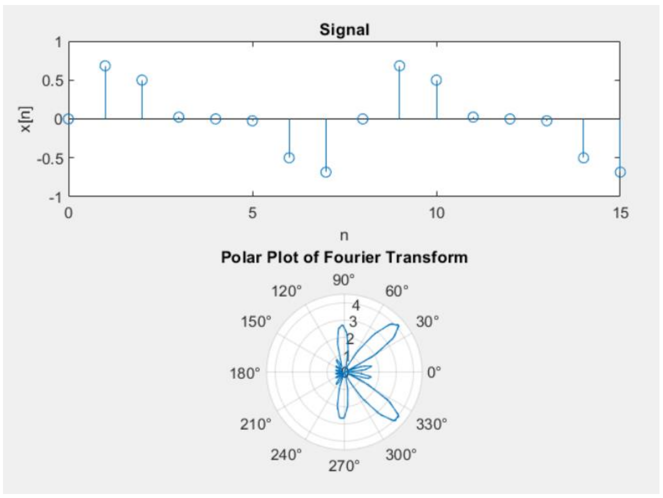
At window sizes 6 and 7, the time-domain signal includes many periods of the sine wave, and the Fourier Transform peaks become very distinct, showing high-frequency resolution. The frequency components are clearly separated, reflecting the improved precision in the frequency domain representation. This further improvement in frequency resolution highlights the trade-off between the number of samples (window size) and the clarity of the frequency domain representation.

In summary, as the window size increases, the time-domain signal captures more periods of the sine wave, leading to a denser and more repetitive signal. This results in a Fourier Transform with sharper and more distinct peaks, indicating improved frequency resolution. Larger window sizes provide better frequency resolution but at the cost of reduced time resolution, capturing the signal's frequency components with greater precision as the window size increases.

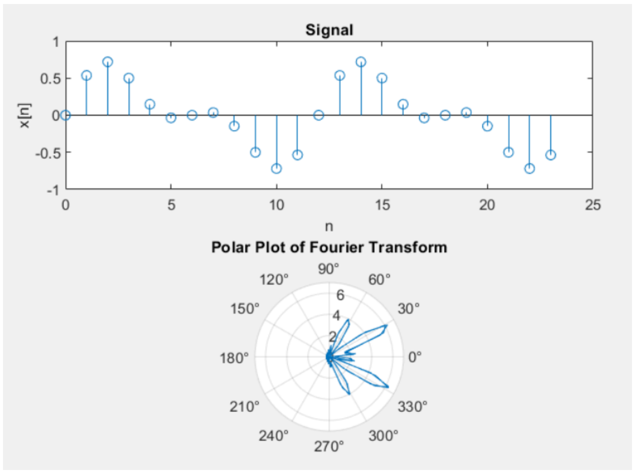
Question 2



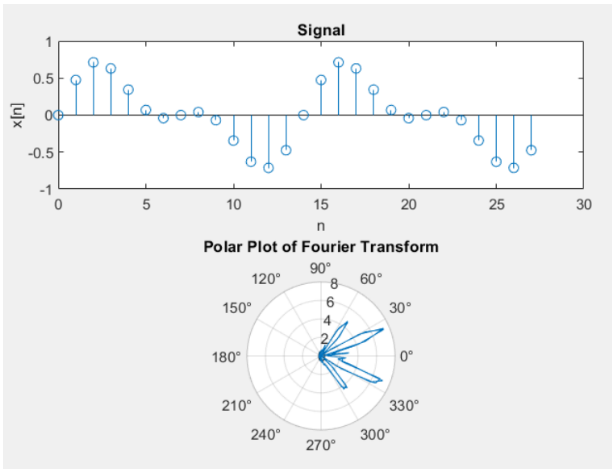
Sampling rate = 2



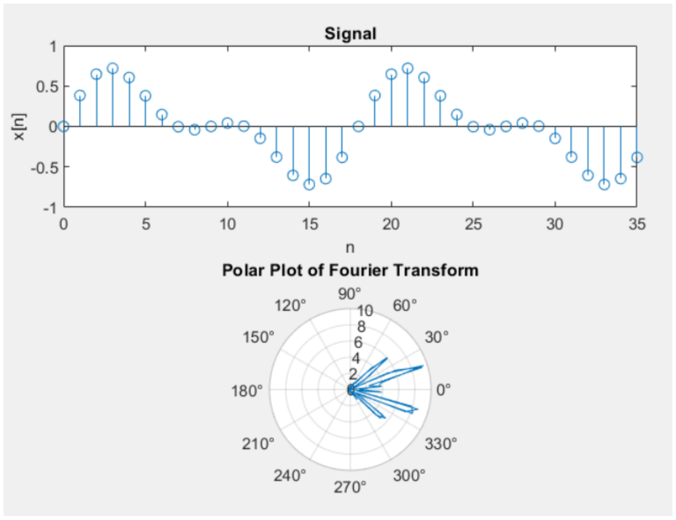
Sampling rate = 4



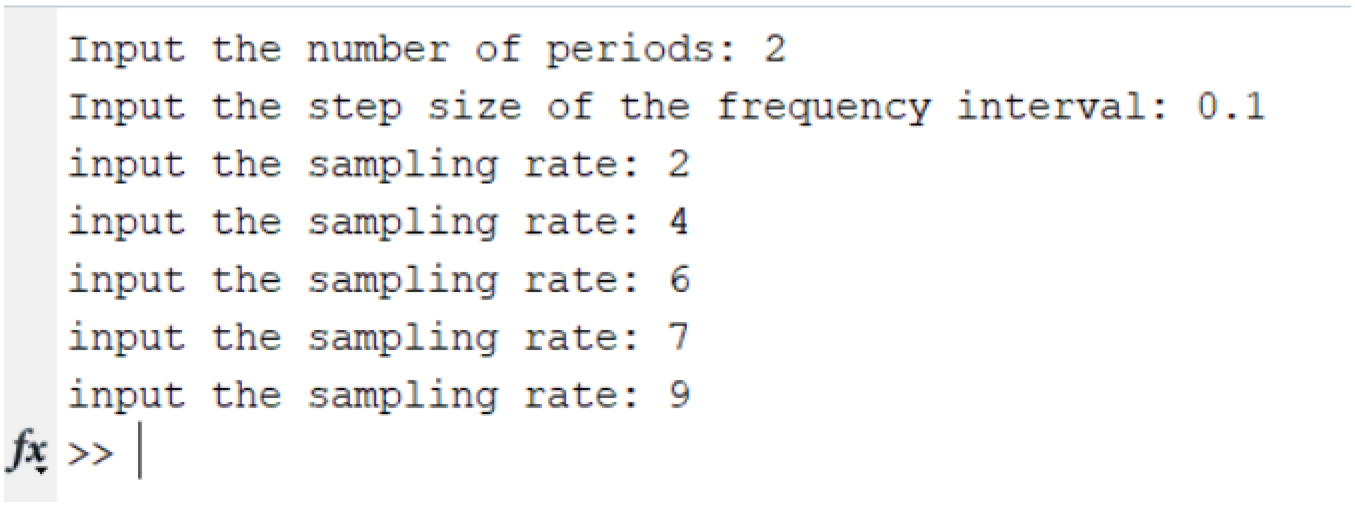
Sampling rate = 6



Sampling rate = 7



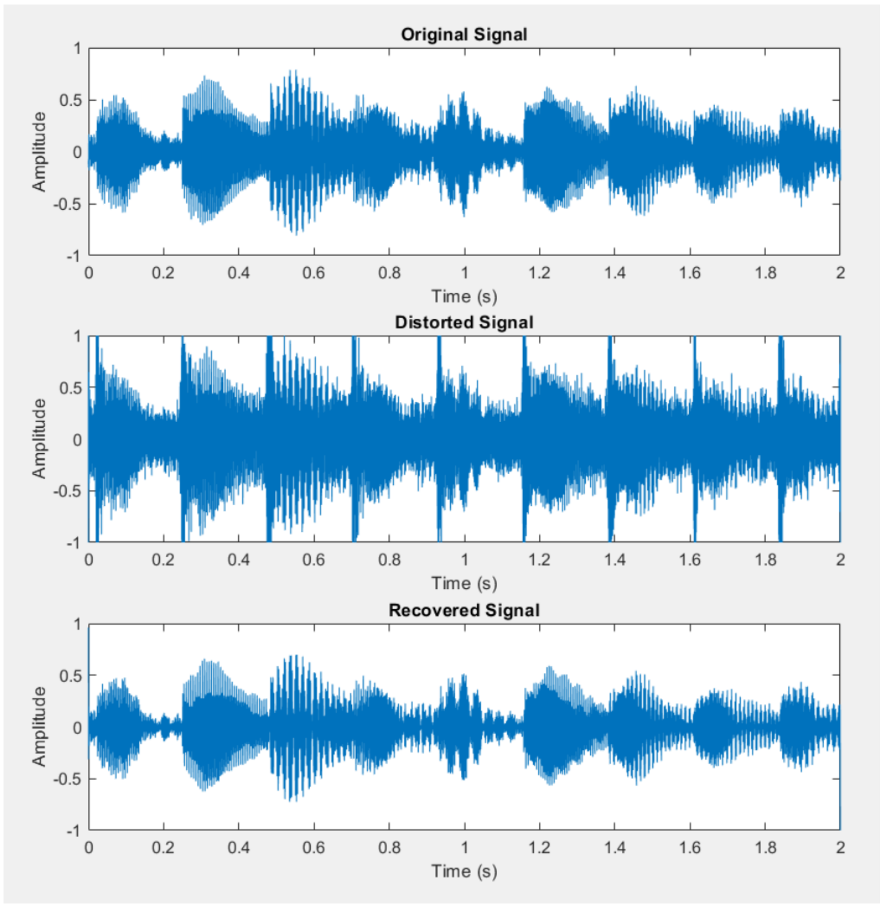
Sampling rate = 9

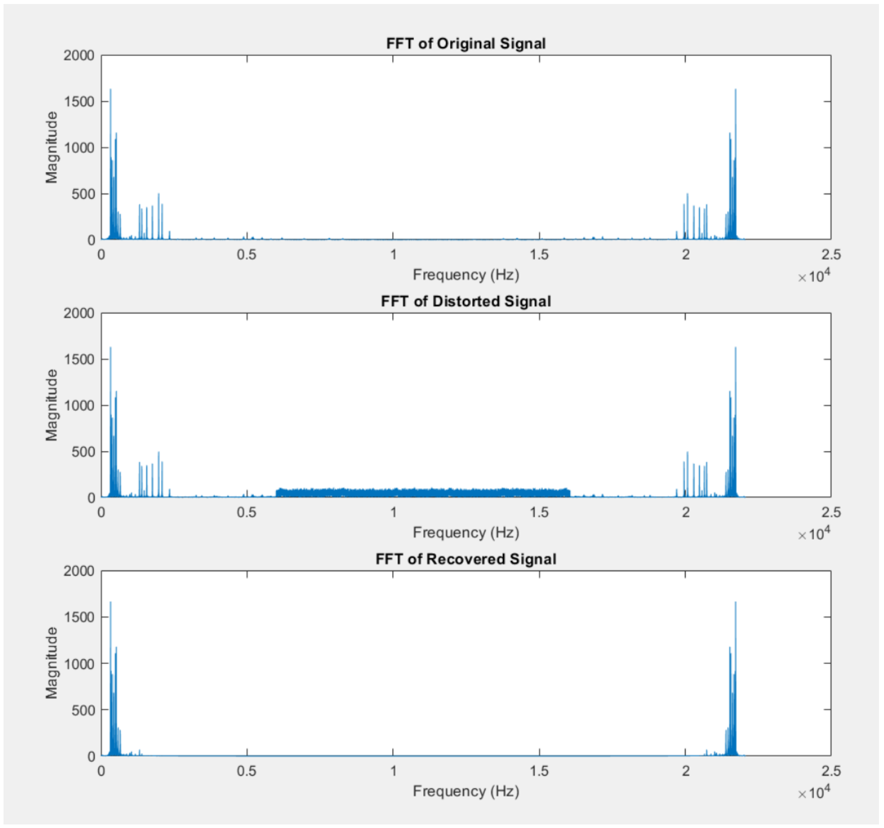


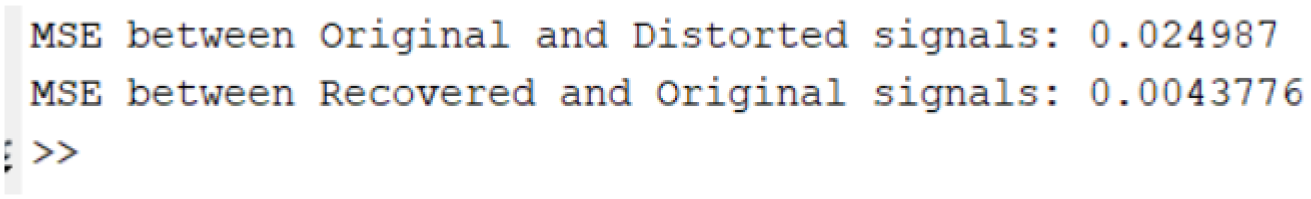
```
Input the number of periods: 2
Input the step size of the frequency interval: 0.1
input the sampling rate: 2
input the sampling rate: 4
input the sampling rate: 6
input the sampling rate: 7
input the sampling rate: 9
fx >>
```

In this set of plots, we observe the effects of varying sampling rates on a composite signal consisting of two sinusoidal components and the resulting Fourier Transform represented in polar form. The signal is generated by combining a 0.5 amplitude sine wave at 1 Hz and a 0.33 amplitude sine wave at 2 Hz. As the sampling rate increases from 2 to 9 Hz, the time-domain representation of the signal becomes progressively clearer. At a sampling rate of 2 Hz, the signal appears sparse, and the polar plot of the Fourier Transform displays broad lobes, indicative of aliasing and inaccurate frequency representation. As the sampling rate increases to 4 and 6 Hz, the time-domain signal becomes denser, and the polar plots show more focused peaks, reflecting reduced aliasing and improved frequency representation. By 9 Hz, the signal is well-defined in the time domain, and the polar plot features sharp, distinct peaks corresponding accurately to the original signal's frequency components, with minimal noise or aliasing effects. Overall, these plots highlight the importance of selecting an appropriate sampling rate to avoid aliasing and achieve a faithful representation of the signal in both time and frequency domains. Higher sampling rates result in more accurate signal representation, reduced distortion, and better precision in detecting frequency components, as clearly demonstrated by the polar plots.

Question 3


Original Signal
Distorted Signal
Recovered Signal
Amplitude
Time (s)


FFT of Original Signal
FFT of Distorted Signal
FFT of Recovered Signal
Magnitude
Frequency (Hz)
$\times 10^4$



```
MSE between Original and Distorted signals: 0.024987
MSE between Recovered and Original signals: 0.0043776
>>
```

The three plots provided show the signals in the time domain (Original, Distorted, and Recovered) and the frequency domain (FFT of Original, Distorted, and Recovered signals). The original signal plot displays a relatively clean waveform, whereas the distorted signal shows increased amplitude and noise, especially in the higher frequency ranges. This is evident in the FFT plot, where the distorted signal contains noticeable noise across the spectrum between approximately 0.6 kHz and 1.6 kHz, indicating the presence of unwanted high-frequency components. In contrast, the recovered signal, which is processed using a low-pass filter, shows a significant reduction in noise. This is especially noticeable in the FFT plot of the recovered signal, where the noise present in the distorted signal is almost eliminated, returning to a spectrum similar to that of the original signal. This indicates that the filtering process was successful in isolating and removing the unwanted noise frequencies.

c)

To recover the original signal from the distorted signal, we designed a system using a low-pass filter, applied in the time domain. The process began with frequency analysis using FFT (Fast Fourier Transform) to identify the noise frequencies present in the distorted signal. This analysis revealed that high-frequency noise components needed to be removed, guiding the choice of filter parameters. For the actual filtering, a time-domain approach was chosen using the 'filtfilt'[1] function in MATLAB, which applies a zero-phase distortion 8th-order Butterworth[2] filter with a cutoff frequency of 1.2 kHz. This approach effectively attenuates unwanted high-frequency noise while preserving the desired signal. The time-domain filtering method was selected because it ensures zero-phase distortion, which is crucial for maintaining the integrity of audio signals where phase information is important. Additionally, time-domain filtering is simpler and more practical for real-time applications, ensuring that the signal's original characteristics are preserved. After applying the filter, the recovered signal was saved as Recovered.wav.

[1] https://www.mathworks.com/help/signal/ref/filtfilt.html
[2] https://stackoverflow.com/questions/61369933/how-to-build-a-bandpass-filter-in-matlab-with-the-butter-function

d)

The Mean Squared Error (MSE) between the recovered signal and the original signal was computed to quantify the effectiveness of the noise reduction. The MSE between the original and distorted signals was found to be 0.024987, whereas the MSE between the recovered and original signals was significantly lower, at 0.0043776. This reduction in MSE clearly demonstrates that the low-pass filtering process effectively improved the signal quality, bringing it closer to the original signal. The decrease in MSE validates the performance of the designed recovery system, confirming that it successfully minimizes the distortions introduced to the original signal by noise.

e)

Yes, by playing and listening to the recovered signal, we can confirm that the system significantly improves the quality of the sound removing the unwanted frequencies.

Conclusion

This lab effectively demonstrated the practical application of signal processing techniques using MATLAB. We explored the impact of sampling rates and window sizes on discrete-time signals, providing insights into how these parameters affect signal representation and frequency resolution. Additionally, the use of polar plots enhanced our ability to visualize amplitude and phase characteristics of signals. The design and implementation of a low-pass filter to recover a distorted audio signal highlighted the importance of filtering in improving signal quality. Overall, this lab enhanced our understanding of both time-domain and frequency-domain methods for signal analysis and processing.

Appendix

Lab3_question1a

```
% Mamadou Diao Kaba 27070179
% Lab 3 question 1 a)
clc;clear;close all;
global n w
x1 = input('Input the number of periods: ');
x2 = input('Input the step size of the frequency interval: ');
w=-x1*pi:x2:x1*pi;

for i = 1:5
    rate = input('input the sampling rate: ');
    N = floor(2 * rate);
    n = 0:(2*N) - 1;
    x=sin(2*pi*n/N);
    x_dtft=func_dtft(x);
    figure(i)
    subplot(2,1,1)
    stem(n,x)
    title('Signal')
    xlabel('n')
    ylabel('x[n]')

    subplot(2,1,2)
    plot(w,abs(x_dtft))
    title('Fourier Transform')
    xlabel('Frequency (rad/sample)')
    ylabel('|X(f)|')
end
```

Lab3_question1b

```
% Mamadou Diao Kaba 27070179
% Lab 3 question 1 b)
clc;clear;close all;
global n w

rate = input('input the sampling rate: ');
N = floor(2 * rate);

for i = 1:5
    window = input('Input the window size: ');
    n = 0:window * N - 1;
    w=-window*pi:0.05:window*pi;
    x=sin(2*pi*n/N);
    x_dtft=func_dtft(x);
    figure(i)
    subplot(2,1,1)
    stem(n,x)
    title('Signal')
    xlabel('n')
    ylabel('x[n]')
    subplot(2,1,2)
    plot(w,abs(x_dtft))
    title('Fourier Transform')
    xlabel('Frequency (rad/sample)')
    ylabel('|X(f)|')
end
```

Lab3_question2

```
% Mamadou Diao Kaba 27070179
% Lab 3 question 2
clc;clear;close all;
global n w
x1 = input('Input the number of periods: ');
x2 = input('Input the step size of the frequency interval: ');
w=-x1*pi:x2:x1*pi;

for i = 1:5
    rate = input('input the sampling rate: ');
    N = floor(2 * rate);
    n = 0:(2*N) - 1;
    x= 0.5*sin(2*pi*n/N) + 0.33*sin(4*pi*n/N);
    x_dtft=func_dtft(x);
    figure(i)
    subplot(2,1,1)
    stem(n,x)
    title('Signal')
    xlabel('n')
    ylabel('x[n]')

    subplot(2,1,2)
    polarplot(w,abs(x_dtft))
    title('Polar Plot of Fourier Transform')
end
```

Lab3_question3

```
% Mamadou Diao Kaba 27070179
% Lab 3 question 3

clc; clear; close all;

% Load the original and distorted signals
[x1, Fs] = audioread('Original.wav');
[x2, Fs] = audioread('Distorted.wav');

% Plot the original and distorted signals in the time domain
L = length(x1);
t = (0:L-1)/Fs;

figure;
subplot(3,1,1)
plot(t, x1);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

```matlab
subplot(3,1,2)
plot(t, x2);
title('Distorted Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Design a low-pass filter to remove high-frequency noise
cutoff_freq = 1.2e3; % Cutoff frequency below the noise frequency
d = designfilt('lowpassiir', 'FilterOrder', 8, ...
               'HalfPowerFrequency', cutoff_freq, ...
               'DesignMethod', 'butter', 'SampleRate', Fs);

% Apply the filter to the distorted signal
x_recovered = filtfilt(d, x2);

% Normalize the recovered signal
x_recovered = x_recovered / max(abs(x_recovered));

% Save the recovered signal
audiowrite('Recovered.wav', x_recovered, Fs);

% Plot the recovered signal in the time domain
subplot(3,1,3)
plot(t, x_recovered);
title('Recovered Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Compute FFT of the signals
X1 = fft(x1);
X2 = fft(x2);
X_recovered = fft(x_recovered);

% Frequency axis for plotting
f = (0:L-1)*(Fs/L);

% Plot the FFTs
figure;
subplot(3,1,1)
plot(f, abs(X1));
title('FFT of Original Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(3,1,2)
plot(f, abs(X2));
title('FFT of Distorted Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(3,1,3)
plot(f, abs(X_recovered));
title('FFT of Recovered Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Compute the MSE
L = length(x1);
MSE_original_distorted = (1/L) * sum((x1 - x2).^2);
```

```matlab
disp(['MSE between Original and Distorted signals: ', num2str(MSE_original_distorted)]);


MSE_recovered_original = (1/L) * sum((x1 - x_recovered).^2);
disp(['MSE between Recovered and Original signals: ', num2str(MSE_recovered_original)]);


% Play the recovered signal
sound(x_recovered, Fs);
```