DIGITAL SIGNAL PROCESSING LAB

SALMANUL FARIS PV - B22ECB60

ASSIGNMENT – 2

1. List some basic applications that utilize Nyquist sampling?

Audio Sampling: Digital audio recording and playback, such as CDs, use Nyquist sampling to capture and reproduce sound accurately. For example, CDs sample audio at 44.1 kHz, which is sufficient to reproduce the full range of human hearing.

Digital Video: Video signals are sampled and digitized for high-quality video streaming and compression. Nyquist sampling ensures video signals are captured accurately without aliasing.

Communication Systems: Analog signals are sampled and converted to digital form for transmission in communication systems. This allows accurate reconstruction of the signal at the receiver end, essential for telecommunication.

Medical Imaging: Technologies like MRI and CT scans use Nyquist sampling principles to capture and reconstruct detailed images of the body, ensuring accurate medical diagnostics.

Radar and Sonar Systems: These systems sample signals to detect and analyze objects. Accurate sampling, based on Nyquist principles, is crucial for effective object detection and distance measurement.

Digital Music Synthesis: In music production, digital synthesizers and samplers use Nyquist sampling to convert

analog sounds into digital formats for manipulation and playback.

Image Processing: Digital cameras use sampling to capture light and create images. High sampling rates prevent aliasing and ensure detailed, accurate digital photographs.

Speech Recognition: Systems that convert spoken words into text rely on accurate sampling of audio signals, based on Nyquist principles, to effectively recognize and process speech.

Data Acquisition Systems: Measurement instruments sample analog signals (e.g., temperature, pressure) and convert them into digital form, using Nyquist sampling to ensure precise data capture.

Digital Signal Processing (DSP): DSP applications use sampled signals to apply filters and perform various signal processing tasks. Accurate sampling ensures filters are applied effectively without introducing errors.

2. List some real time applications of linear convolution.

Audio Processing: Linear convolution is used in real-time audio effects like reverb, echo, and equalization. It helps in shaping sound by applying filters to audio signals in real time.

Speech Processing: In speech recognition and enhancement systems, linear convolution is used to filter and process speech signals, improving clarity and reducing noise in real-time communications.

Image Processing: Real-time image filtering applications, such as edge detection, blurring, and sharpening, use linear convolution to modify image characteristics on-the-fly.

Radar Systems: Linear convolution is applied in radar signal processing for filtering and detecting objects. It helps in improving the accuracy of object detection and tracking in real time.

Sonar Systems: Similar to radar, sonar systems use linear convolution for real-time signal processing to detect and identify underwater objects and features.

Communication Systems: In digital communication, linear convolution is used for channel equalization and noise reduction, enhancing the quality and reliability of data transmission in real time.

Video Processing: Real-time video filters and effects, such as motion blur and noise reduction, use linear convolution to process video frames and enhance visual quality.

Medical Monitoring: Real-time data filtering in medical devices, such as ECG and EEG monitors, uses linear

convolution to process physiological signals for accurate health monitoring and diagnostics.

Control Systems: Linear convolution is employed in real-time control systems for filtering sensor data and improving system responses by applying digital filters.

Consumer Electronics: Devices like smartphones and smart speakers use linear convolution for real-time audio enhancements and noise cancellation in communication and multimedia applications.

Virtual Reality (VR): In VR systems, linear convolution is used for real-time spatial audio processing and environmental sound effects to create immersive experiences.

3. Write a MATLAB/SCILAB Code to verify sampling theorem using an audio/speech signal.

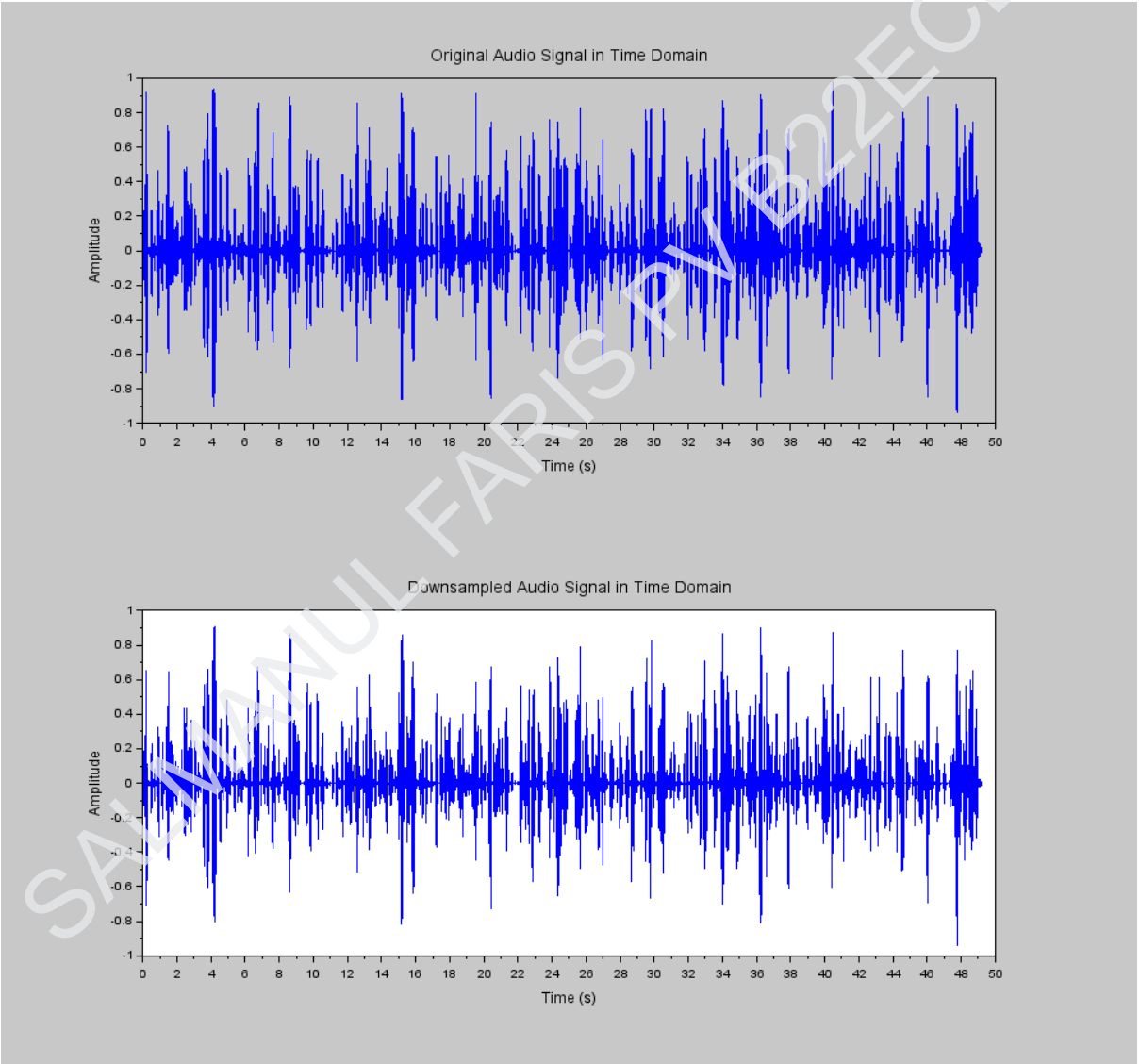
```
// Step 1: Load the audio file
// Replace 'path_to_your_audio_file.wav' with the actual path
to your downloaded .wav file
[audio_signal, sample_rate] =
wavread("C:\Users\imsal\Desktop\dsp lab\audio.wav");

// Display basic information about the signal
disp("Sample Rate: " + string(sample_rate) + " Hz");
disp("Audio Signal Length: " + string(length(audio_signal)));

// Step 2: Perform the Fourier Transform to analyze the
frequency content
n = length(audio_signal);
audio_spectrum = abs(fft(audio_signal)); // Get the magnitude
of the FFT
frequencies = (0:n-1) * (sample_rate/n); // Calculate
corresponding frequency values

// Step 3: Plot the frequency spectrum of the audio signal
clf; // Clear the figure window
//figure();
subplot(2, 1, 1);
plot(frequencies, audio_spectrum);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Spectrum of the Original Audio Signal');

// Step 4: Find the maximum frequency component (fm)
// SCILAB doesn't support the ~ syntax, so we need to
separate the max function
[max_value, max_index] = max(audio_spectrum(1:n/2)); // Only
consider the first half of the spectrum
fm = frequencies(max_index);
```

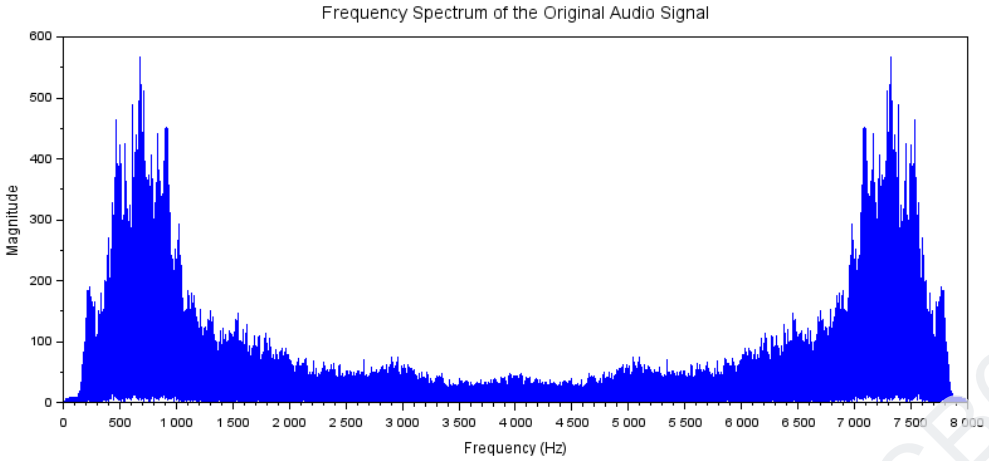
```
disp("Maximum Frequency Component (fm): " + string(fm) + "
Hz");

// Step 5: Calculate the Nyquist Rate
nyquist_rate = 2 * fm;
disp("Nyquist Rate: " + string(nyquist_rate) + " Hz");

// Step 6: Check if the original sampling rate meets the
Nyquist criterion
if sample_rate >= nyquist_rate then
    disp("The original sampling rate meets the Nyquist
criterion.");
else
    disp("The original sampling rate is below the Nyquist
rate. Aliasing may occur.");
end

// Step 7: Downsample the signal to test the sampling theorem
// Choose a downsampling factor that brings the new sample
rate below the Nyquist rate
//downsample_factor = round(sample_rate / nyquist_rate) + 1;
// Just below Nyquist
// Choose a more aggressive downsampling factor to get below
the Nyquist rate
downsample_factor = 10; // Example of an aggressive
downsampling factor
downsampled_signal = audio_signal(1:downsample_factor:$); //
Downsample the signal
new_sample_rate = sample_rate / downsample_factor;

disp("New Sample Rate after Downsampling: " +
string(new_sample_rate) + " Hz");

// Step 8: Listen to the downsampled signal (optional, if
sound is supported)
// Uncomment the line below to play the downsampled signal
// sound(downsampled_signal, new_sample_rate);
```

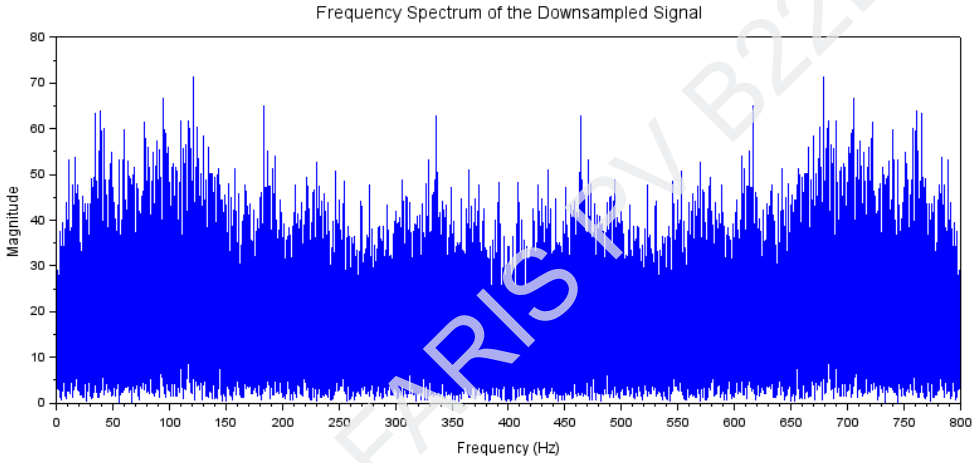
```
// Step 9: Plot the frequency spectrum of the downsampled
signal
downsampled_spectrum = abs(fft(downsampled_signal)); // FFT
of downsampled signal
frequencies_new = (0:length(downsampled_signal)-1) *
(new_sample_rate/length(downsampled_signal));

//clf; // Clear the figure window
//figure();
subplot(2, 1, 2);
plot(frequencies_new, downsampled_spectrum);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Spectrum of the Downsampled Signal');

// Step 10: Plot the original signal in the time domain
t_original = (0:n-1) / sample_rate; // Time vector for
original signal
figure();
subplot(2, 1, 1); // Create a subplot (2 rows, 1 column, 1st
plot)
plot(t_original, audio_signal);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Audio Signal in Time Domain');

// Step 11: Plot the downsampled signal in the time domain
t_downsampled = (0:length(downsampled_signal)-1) /
new_sample_rate; // Time vector for downsampled signal
subplot(2, 1, 2); // Create a subplot (2 rows, 1 column, 2nd
plot)
plot(t_downsampled, downsampled_signal);
xlabel('Time (s)');
ylabel('Amplitude');
title('Downsampled Audio Signal in Time Domain');
```

o/p:

"Sample Rate: 8000 Hz"
"Audio Signal Length: 393348"
"Maximum Frequency Component (fm): 674.82229 Hz"
"Nyquist Rate: 1349.6446 Hz"
"The original sampling rate meets the Nyquist criterion."
"New Sample Rate after Downsampling: 800 Hz"







4. Write a MATLAB/SCILAB code that performs linear convolution of a sine signal with a low-pass filter

```
// Parameters
Fs = 1000;                 // Sampling frequency (Hz)
t = 0:1/Fs:1-1/Fs;         // Time vector
f = 4;                    // Frequency of sine wave (Hz)

// Generate sine signal
x = sin(2*%pi*f*t);

// Design a low-pass filter using a simple moving average
filter
N = 10;                    // Order of the filter (number of
taps)
h = ones(1, N)/N;          // Impulse response of the low-pass
filter

// Perform linear convolution
y = conv(x, h);

// Time vector for the convolved signal
t_conv = (0:length(y)-1)/Fs;

// Plotting the signals
clf;
subplot(3,1,1);
plot(t, x);
title('Original Sine Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,2);
plot((0:N-1)/Fs, h);
title('Low-Pass Filter Impulse Response');
xlabel('Time (s)');
```

```
ylabel('Amplitude');

subplot(3,1,3);
plot(t_conv, y);
title('Convolved Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```