# Question:

## The first question

To explore the human ear's insensitivity to phase changes in sound, let's conduct an experiment using a normal speech segment, x1(t), with a duration of approximately 5 seconds and a sampling rate of 16 kHz. Follow these steps:

1. **Reverse the segment x1(t) to obtain x1(−t)** (equivalent to reversing the phase) and play x1(−t). Can you understand its semantics? How does it compare to x1(t)?
2. **Segment x1(t) into equal-length frames (frame length l)**, reverse each frame, and then concatenate them back together to form x2(t). Gradually reduce lll. At what value of lll can you start hearing the same semantics in x2(t) as in x1(t)? Research and explain this phenomenon based on literature.

## The second question

Add random white noise to x1(t), with signal-to-noise ratios (SNRs) defined as SNR=, measured in decibels (dB). Use SNR values of 10 dB, 0 dB, and -10 dB.

1. Apply the Fourier transform to the noisy speech signal. Design a filter in the frequency domain to remove the noise, and then perform an inverse Fourier transform to obtain the denoised speech signal. Compare the auditory quality of the original speech, the noisy speech, and the denoised speech. Plot their waveforms for comparison.
2. Based on the frequency domain filtering in 1), design an equivalent filter in the time domain. Apply this time domain filter to the noisy speech signal. Can you achieve results similar to those in 1)?

# Design

## The first question

**Step 1: Audio Signal Acquisition and Preprocessing**

* Choose an approximately 5-second long audio clip with a sampling rate of 16 kHz.
* If the sampling rate is not 16 kHz, resample the audio to 16 kHz.
* Convert stereo audio to mono by averaging the two channels, ensuring the analysis focuses on a single channel's characteristics.

**Step 2: Full Signal Phase Inversion**

* Reverse the entire audio signal in time to obtain a signal where the phase is completely inverted.
* Play both the original and reversed signals. Evaluate whether the reversed signal retains any understandable content and compare it to the original signal.
* Plot the waveforms of both the original and reversed signals to visually inspect the symmetry introduced by the inversion.

**Step 3: Frame-Based Inversion**

* Divide the audio signal into frames of varying lengths (2000, 1000, 500, 250, 125, and 62 samples). Reverse each frame in time and reassemble the frames to form a new signal.
* Play the reassembled signals for each frame length. Assess at what frame length the reassembled signal begins to sound similar to the original signal, indicating a loss of perceptible phase changes.
* Gradually decrease the frame length to identify the threshold below which the reversed segments' effects become imperceptible.

**Step 4: Time-Frequency Analysis Using STFT**

* Perform Short-Time Fourier Transform on the signals processed with different frame lengths to analyze their time-frequency characteristics.
* Generate STFT spectrograms for each frame length to observe how phase inversion affects the signal's spectral content over time.
* Compare the spectrograms to understand the impact of different frame lengths on the signal’s time-frequency structure.

## The second question

**Step 1: Generate Noisy Audio Signals**

* Load the original audio signal , and convert it to mono if it's stereo.
* Add random white noise to the audio signal to achieve the desired SNR levels of 10 dB, 0 dB, and -10 dB.

**Step 2: Frequency Domain Filtering**

* Perform the Fourier transform on the noisy audio signal.
* Design a low-pass filter in the frequency domain with a cutoff frequency of 3000 Hz.
* Apply the filter and perform the inverse Fourier transform to obtain the denoised audio signal.
* Compare the waveforms and auditory quality of the original, noisy, and denoised signals.

**Step 3: Time Domain Filtering**

* Design a low-pass FIR filter in the time domain with the same cutoff frequency as the frequency domain filter.
* Apply the filter to the noisy audio signal to obtain the denoised audio signal.
* Compare the waveforms and auditory quality of the original, noisy, and denoised signals with those obtained from frequency domain filtering.

# Result

## The first question

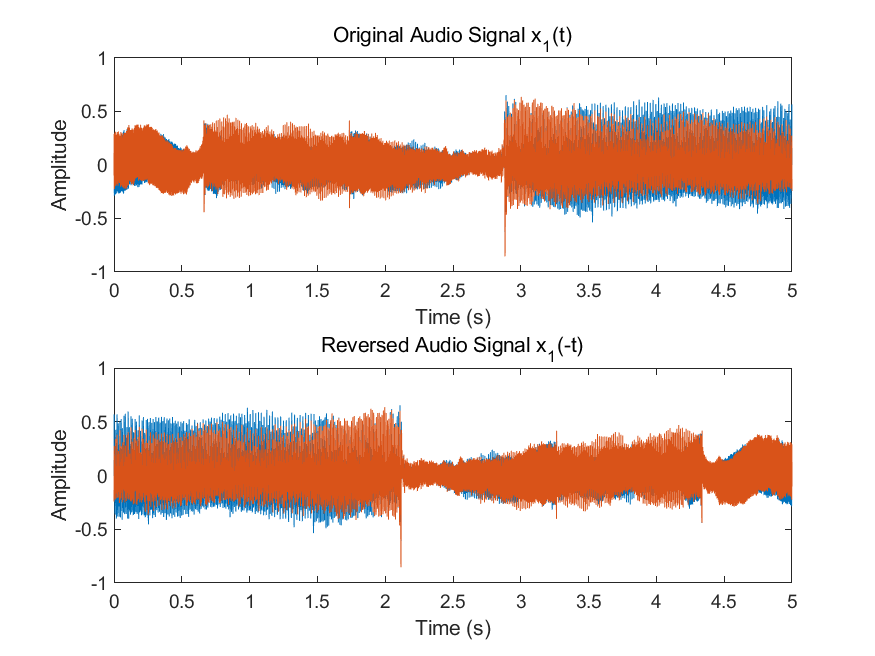


Fig 1 Original and Reversed Audio Signal

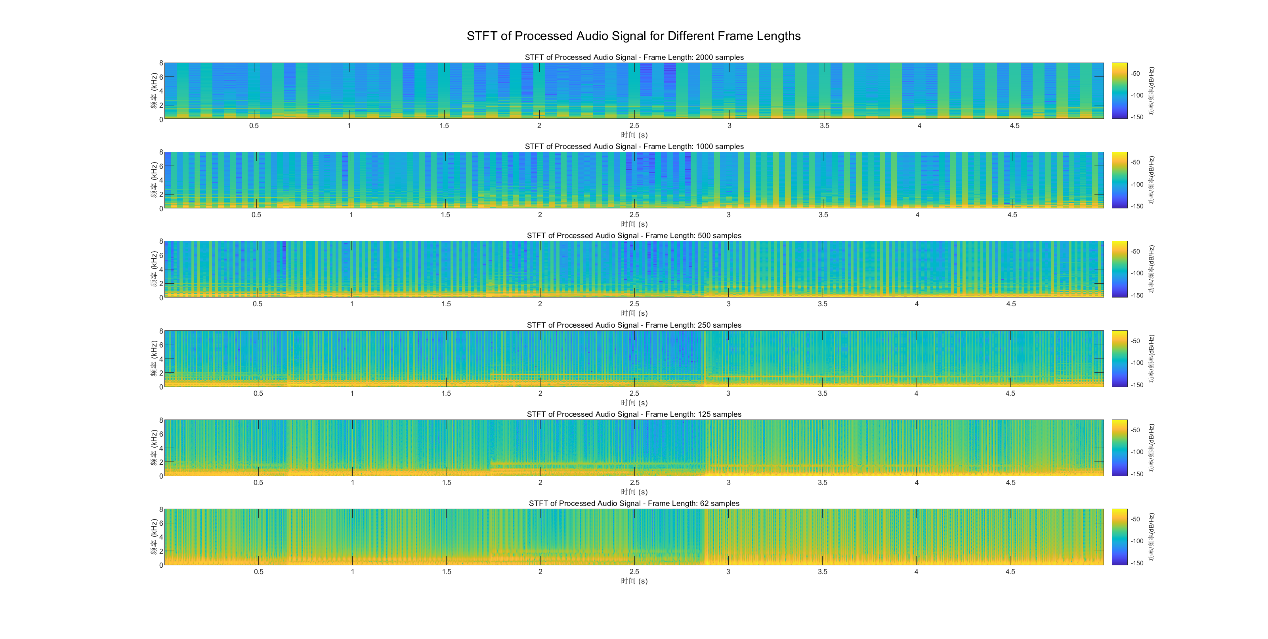


Fig 2 STFT Results

## The second question

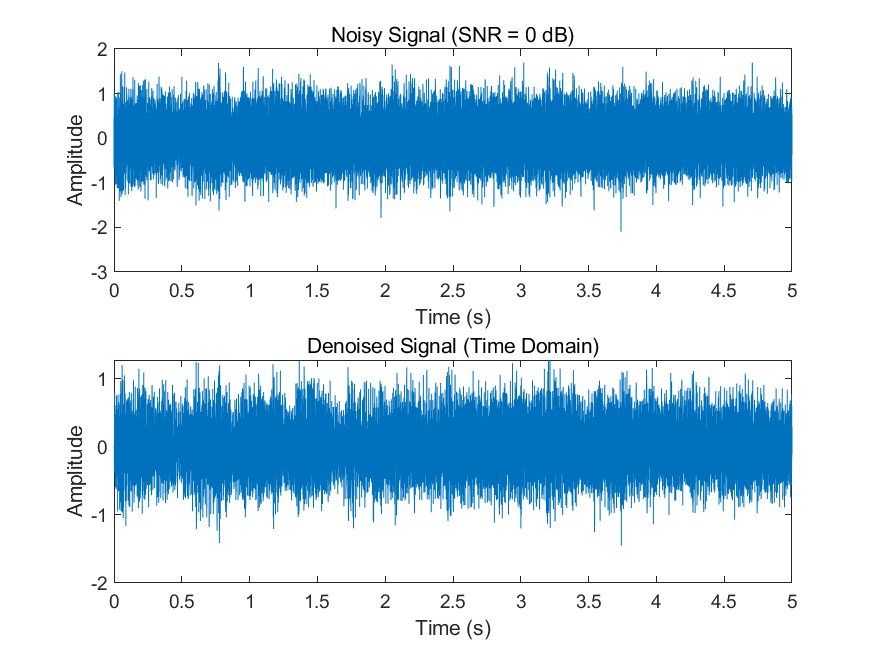


Fig 3 Time Filter(0dB)

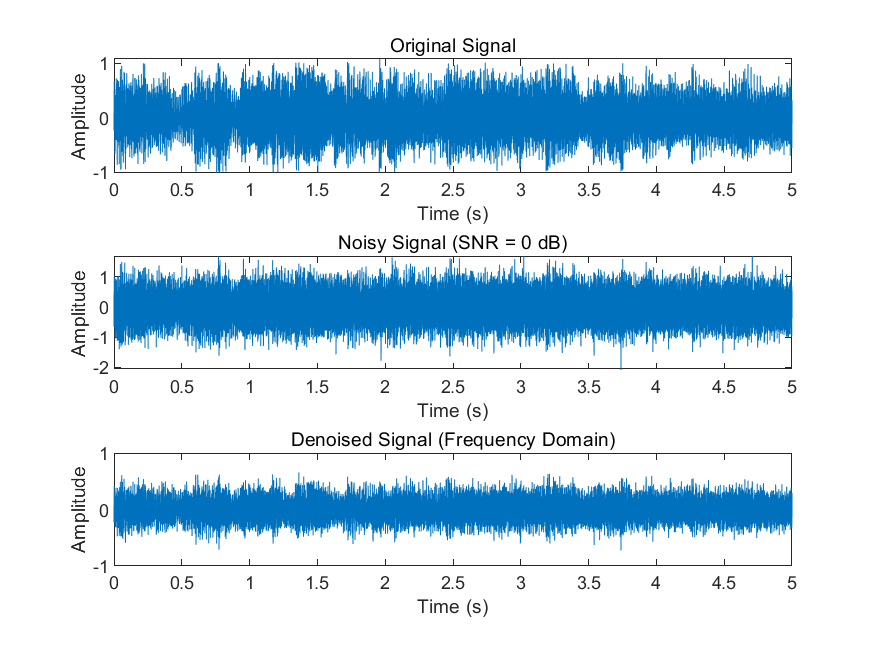


Fig 4 Frequency Filter(0dB)

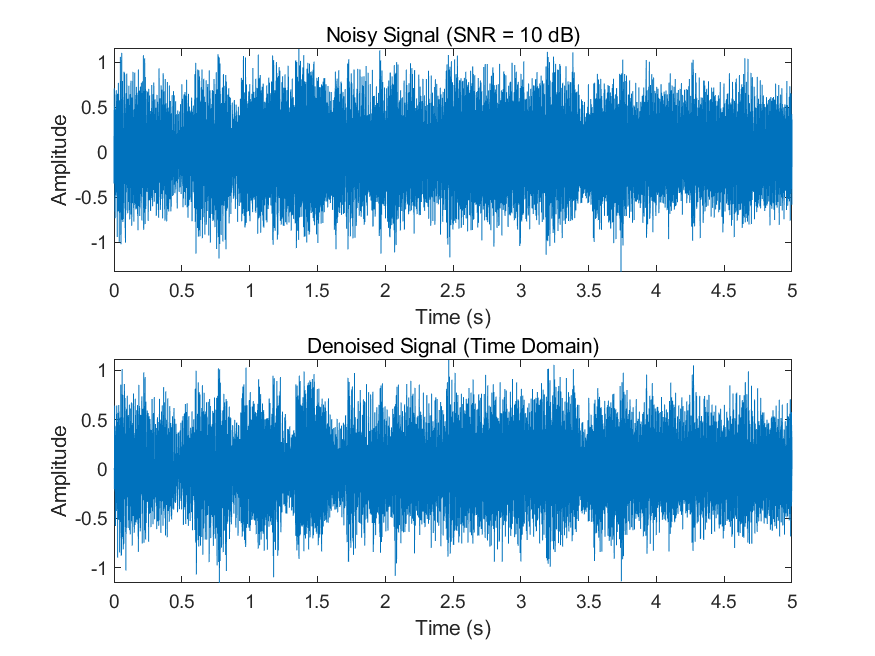


Fig 5 Time Filter(10dB)

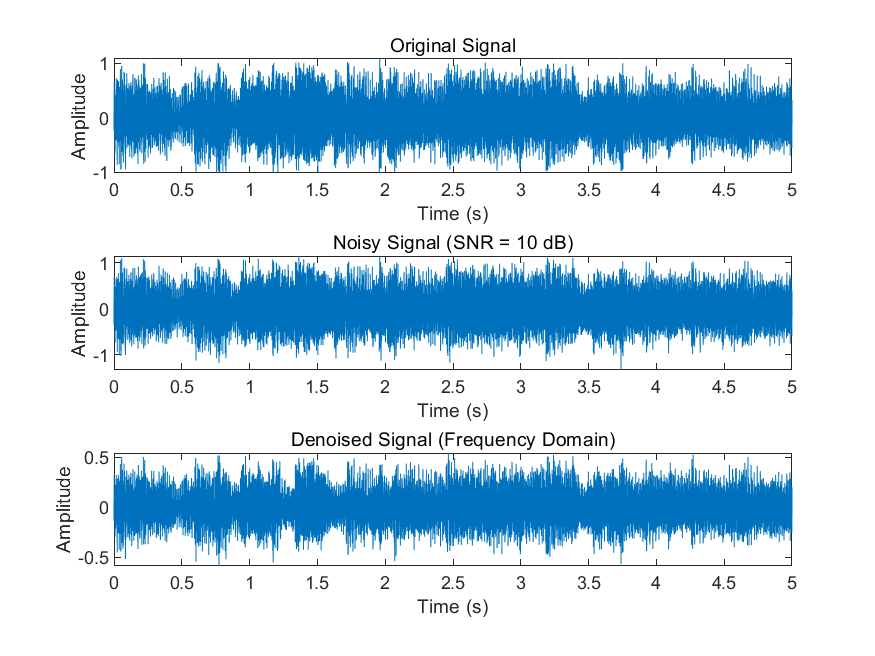


Fig 6 Frequency Filter(10dB)

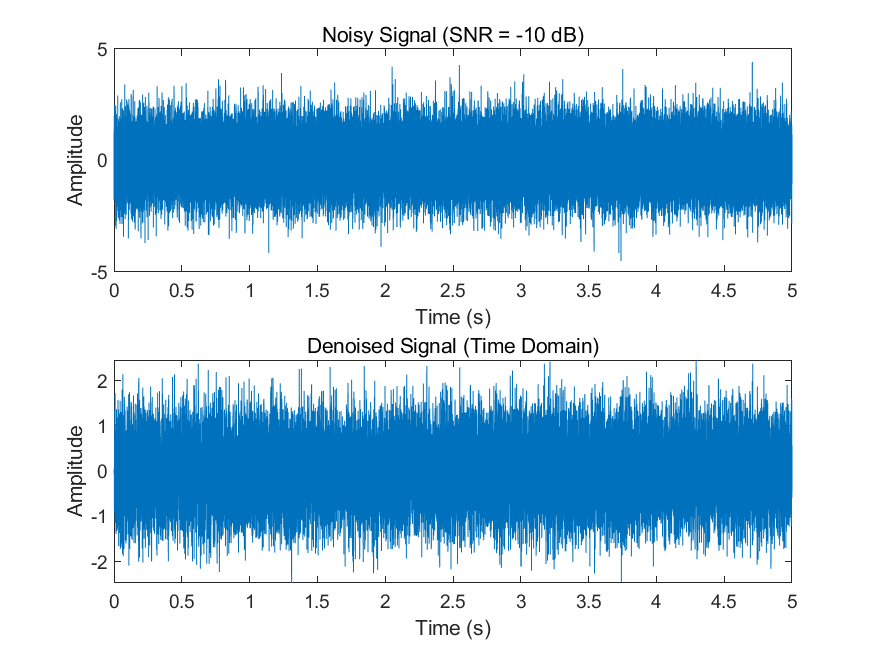


Fig 7 Time Filter(-10dB)

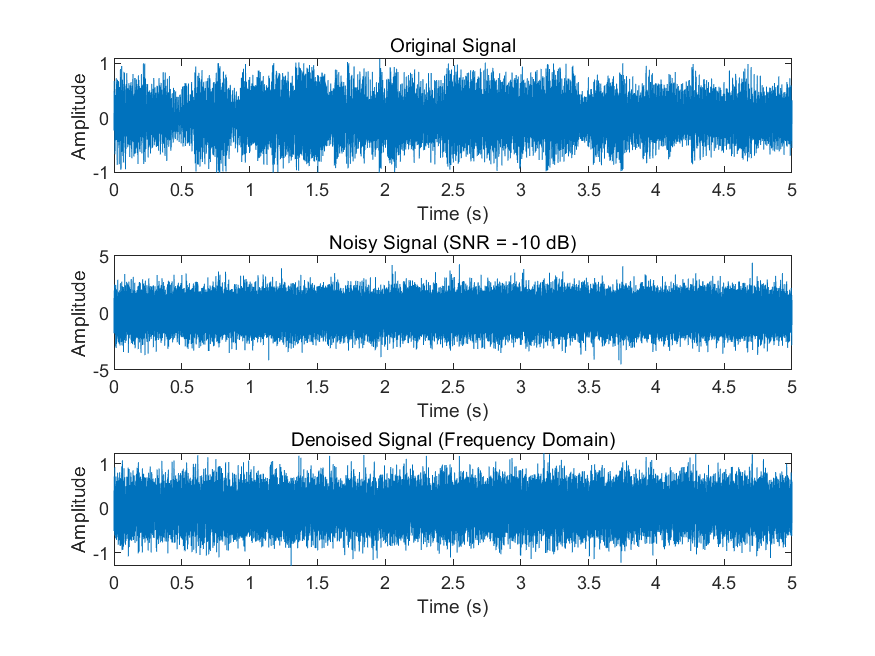


Fig 8 Frequency Filter(-10dB)

# Analysis

## The first question

**Full Signal Phase Inversion**

* **Waveform Analysis**: The waveforms of the original and reversed signals show that the reversed signal is a time-mirrored version of the original. This inversion significantly alters the temporal sequence of the audio.
* **Auditory Perception**: When listening to the reversed signal, the content, especially speech, becomes largely unintelligible. The reversal disrupts the natural flow of phonetic elements, making the reversed speech difficult to comprehend. For non-verbal audio, the melody may be recognizable, but the temporal coherence is lost.

**Frame-Based Inversion**

* **2000, 1000 samples**: The reassembled signals retain the disruptive effect of phase inversion. Even with these frames, the temporal structure is sufficiently altered to affect intelligibility.
* **500, 250 samples**: As the frame length decreases, the reassembled signal begins to resemble the original more closely. While some artifacts are present, the overall content becomes more recognizable.
* **125, 62 samples**: At these lengths, the reassembled signal sounds almost identical to the original. The phase inversion within these short frames does not significantly impact the perceived signal due to the human ear's inability to detect rapid phase changes within such short segments.

**STFT Analysis**

* **Time-Frequency Characteristics**: STFT spectrograms reveal how the signal's spectral content changes over time under different phase manipulations.
  + **Long Frame Lengths**: The spectrograms for longer frames show distinct frequency bands with noticeable phase changes, reflecting the significant impact of inversion over extended periods.
  + **Medium Frame Lengths**: The frequency bands start to blur, indicating a mix of phase effects but still maintaining some level of discernible spectral patterns.
  + **Short Frame Lengths**: The spectrograms exhibit a more uniform and smooth frequency distribution, as the rapid phase changes within short frames blend into a continuous spectral representation, which is less perceptible to the ear.
* **Human Perception**: These observations align with auditory perception, where phase changes within short frames are not easily detected, thus making the signal sound closer to the original despite phase manipulations. I don't hear any difference at all.

## The second question

The results include waveform plots and auditory comparisons for each SNR level. Below is the analysis based on the given plots for each SNR level.

1. **SNR = 10 dB**

**Waveform Comparison:** The original signal is clearly visible. The noisy signal shows significant noise but retains much of the original signal's structure. Both frequency and time domain filters effectively reduce the noise, making the denoised signal similar to the original signal.

**Auditory Comparison:** The denoised signal in both domains retains most of the original audio quality, with reduced background noise.

1. **SNR = 0 dB**

**Waveform Comparison:** The noisy signal's waveform is heavily influenced by noise, making the original signal less distinguishable. Both filters manage to reduce the noise, but some residual noise remains.

**Auditory Comparison:** The denoised signals are significantly clearer than the noisy signal, though some noise artifacts may still be present.

1. **SNR = -10 dB**

**Waveform Comparison:** The noisy signal is dominated by noise, making it difficult to discern the original signal. Both filters reduce noise, but the denoised signals still contain significant noise artifacts.

**Auditory Comparison**: The denoised signals are improved compared to the noisy signal but are far from the clarity of the original signal.

# Conclusion

## The first question

This experiment demonstrates that human perception of phase changes in audio signals is highly dependent on the temporal context of these changes. Full signal inversion disrupts the temporal order significantly, making speech unintelligible. However, when the signal is segmented into very short frames, phase inversion within these frames becomes imperceptible due to the human auditory system's limited time resolution. The STFT analysis provides a detailed view of how phase changes manifest in the time-frequency domain, reinforcing the findings that short-duration phase changes are less impactful on the perceived audio quality.

## The second question

Both frequency and time domain filters are effective in reducing noise, particularly at higher SNR levels (10 dB). The effectiveness decreases as the SNR decreases, with -10 dB being the most challenging scenario. In practical applications, combining both filtering techniques or using adaptive filtering strategies might further improve noise reduction performance, especially in low SNR scenarios. In environments with relatively low background noise (e.g., recording studios or quiet rooms), both frequency and time domain filters can significantly enhance audio signal quality. Simple time domain filters may be preferred for ease of implementation and real-time processing capabilities. More complex filter designs or additional noise reduction techniques (such as adaptive filtering or deep learning methods) may be needed to further improve audio signal quality. In practical applications, combining the strengths of both frequency and time domain filters may provide better noise reduction performance.

# Code appendix

## The first question

**Audio\_Intercept**

% Step 1: Read the audio file

[originalAudio, fs] = audioread('music2.flac'); % 'input\_audio\_file.wav' is the path and name of the audio file

% Step 2: Calculate sample indices for the desired segment

start\_time = 1; % Start time in seconds (2 minutes and 7 seconds)

end\_time = 6; % End time in seconds (2 minutes and 12 seconds)

start\_sample = round(start\_time \* fs); % Calculate the starting sample index

end\_sample = round(end\_time \* fs); % Calculate the ending sample index

% Step 3: Extract the audio segment

audio\_segment = originalAudio(start\_sample:end\_sample, :); % Extract the segment from the original audio

% Step 4: Save the extracted audio segment

audiowrite('segment\_music2.wav', audio\_segment, fs); % Save the segment as a new file 'output\_audio\_segment.wav'

% Display completion message

disp('The audio segment has been successfully saved as segment\_music2.wav');

**Task1**

% Load the audio file with any sample rate

[x1, fs] = audioread('segment\_music2.wav');

% If the sample rate is not 16 kHz, resample it to 16 kHz

if fs ~= 16000

disp(['Resampling from ', num2str(fs), ' Hz to 16000 Hz...']);

x1 = resample(x1, 16000, fs);

fs = 16000; % Update the sample rate to 16 kHz

end

% Continue with the rest of the code

t = (0:length(x1)-1) / fs; % Time vector for plotting

% Reverse the audio signal

x1\_reversed = flipud(x1);

% Play the original and reversed audio

disp('Playing original audio...');

sound(x1, fs);

pause(length(x1) / fs + 2);

disp('Playing reversed audio...');

sound(x1\_reversed, fs);

pause(length(x1\_reversed) / fs + 2);

% Plot the original and reversed audio for visualization

figure;

subplot(2, 1, 1);

plot(t, x1);

title('Original Audio Signal x\_1(t)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(2, 1, 2);

plot(t, x1\_reversed);

title('Reversed Audio Signal x\_1(-t)');

xlabel('Time (s)');

ylabel('Amplitude');

% Divide into frames, reverse each frame, and reassemble

frame\_lengths = [2000, 1000, 500, 250, 125, 62];

for i = 1:length(frame\_lengths)

l = frame\_lengths(i);

num\_frames = floor(length(x1) / l);

x2 = zeros(size(x1));

for k = 1:num\_frames

frame\_start = (k - 1) \* l + 1;

frame\_end = k \* l;

frame = x1(frame\_start:frame\_end);

frame\_reversed = flipud(frame);

x2(frame\_start:frame\_end) = frame\_reversed;

end

disp(['Playing audio with frame length of ', num2str(l), ' samples...']);

sound(x2, fs);

pause(length(x2) / fs + 2);

end

**STFT**

% Load the audio file with any sample rate

[x1, fs] = audioread('segment\_music2.wav');

% If the sample rate is not 16 kHz, resample it to 16 kHz

if fs ~= 16000

disp(['Resampling from ', num2str(fs), ' Hz to 16000 Hz...']);

x1 = resample(x1, 16000, fs);

fs = 16000; % Update the sample rate to 16 kHz

end

% If the audio is stereo, convert it to mono by averaging the two channels

if size(x1, 2) > 1

disp('Converting stereo to mono by averaging the channels...');

x1 = mean(x1, 2); % Convert to mono by averaging the two channels

end

% Define frame lengths to test

frame\_lengths = [2000, 1000, 500, 250, 125, 62];

% Plotting the STFT results for each frame length

figure;

for i = 1:length(frame\_lengths)

l = frame\_lengths(i);

num\_frames = floor(length(x1) / l);

x2 = zeros(size(x1));

for k = 1:num\_frames

frame\_start = (k - 1) \* l + 1;

frame\_end = k \* l;

frame = x1(frame\_start:frame\_end);

% Reverse the frame

frame\_reversed = flipud(frame);

x2(frame\_start:frame\_end) = frame\_reversed;

end

% If the processed audio is stereo, convert it to mono

if size(x2, 2) > 1

x2 = mean(x2, 2);

end

% Plot the STFT of the processed signal

subplot(length(frame\_lengths), 1, i);

spectrogram(x2, hamming(l), [], [], fs, 'yaxis');

title(['STFT of Processed Audio Signal - Frame Length: ', num2str(l), ' samples']);

end

% Adjust figure properties

sgtitle('STFT of Processed Audio Signal for Different Frame Lengths');

## The second question

% Load the audio file

[x1, fs] = audioread('segment\_music.wav');

% Resample to 16 kHz

if fs ~= 16000

x1 = resample(x1, 16000, fs);

fs = 16000;

end

% Convert to mono if stereo

if size(x1, 2) > 1

x1 = mean(x1, 2);

end

% Generate white noise

len = length(x1);

noise = randn(len, 1);

% Define SNR values

SNRs = [10, 0, -10];

% Initialize cell arrays to store results

noisy\_signals = cell(length(SNRs), 1);

denoised\_signals\_freq = cell(length(SNRs), 1);

denoised\_signals\_time = cell(length(SNRs), 1);

% Loop over SNR values

for i = 1:length(SNRs)

SNR = SNRs(i);

noise\_power = var(x1) / (10^(SNR / 10));

noise\_scaled = sqrt(noise\_power) \* noise;

noisy\_signal = x1 + noise\_scaled;

noisy\_signals{i} = noisy\_signal;

% Fourier Transform of noisy signal

Y = fft(noisy\_signal);

f = (0:length(Y)-1) \* fs / length(Y); % Frequency vector

% Design a simple low-pass filter in the frequency domain

cutoff\_freq = 3000; % Cutoff frequency in Hz

H = double(f <= cutoff\_freq); % Frequency domain filter

% Apply the filter

Y\_filtered = Y .\* H';

denoised\_signal\_freq = ifft(Y\_filtered);

denoised\_signals\_freq{i} = real(denoised\_signal\_freq); % Get real part

% Design a similar low-pass filter in the time domain

order = 100; % Filter order

b = fir1(order, cutoff\_freq/(fs/2)); % Low-pass FIR filter

% Apply the time domain filter

denoised\_signal\_time = filter(b, 1, noisy\_signal);

denoised\_signals\_time{i} = denoised\_signal\_time;

% Plot waveforms

t = (0:length(x1)-1) / fs;

figure;

subplot(3, 1, 1);

plot(t, x1);

title('Original Signal');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

plot(t, noisy\_signal);

title(['Noisy Signal (SNR = ', num2str(SNR), ' dB)']);

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 3);

plot(t, denoised\_signals\_freq{i});

title('Denoised Signal (Frequency Domain)');

xlabel('Time (s)');

ylabel('Amplitude');

% Plot time-domain denoising result

figure;

subplot(2, 1, 1);

plot(t, noisy\_signal);

title(['Noisy Signal (SNR = ', num2str(SNR), ' dB)']);

xlabel('Time (s)');

ylabel('Amplitude');

subplot(2, 1, 2);

plot(t, denoised\_signals\_time{i});

title('Denoised Signal (Time Domain)');

xlabel('Time (s)');

ylabel('Amplitude');

end

% Play the original audio

disp('Playing original audio...');

sound(x1, fs);

pause(length(x1) / fs + 2); % Pause to allow full playback

% Play the noisy audio signals

for i = 1:length(SNRs)

disp(['Playing noisy audio with SNR = ', num2str(SNRs(i)), ' dB...']);

sound(noisy\_signals{i}, fs);

pause(length(noisy\_signals{i}) / fs + 2); % Pause to allow full playback

end

% Play the frequency domain denoised audio signals

for i = 1:length(SNRs)

disp(['Playing frequency domain denoised audio with SNR = ', num2str(SNRs(i)), ' dB...']);

sound(denoised\_signals\_freq{i}, fs);

pause(length(denoised\_signals\_freq{i}) / fs + 2); % Pause to allow full playback

end

% Play the time domain denoised audio signals

for i = 1:length(SNRs)

disp(['Playing time domain denoised audio with SNR = ', num2str(SNRs(i)), ' dB...']);

sound(denoised\_signals\_time{i}, fs);

pause(length(denoised\_signals\_time{i}) / fs + 2); % Pause to allow full playback

end