1. 找一段正常语音片段，记为，长度约5秒，采样率为16kHz。编程验证人耳对声音相位的改变不敏感，步骤如下：

1）将反转得到（相当于相位取反），播放，能否听清其语义？是否与一样？

2）将分割成等长的帧（帧的长度为*l*），将各帧反转后再重新拼接合成。逐渐减小*l*，当*l*取多小时，可以从中听出与相同的语义？通过查阅文献对此现象进行解释。

2、对叠加随机白噪声，信噪比（信号噪声能量比：signal-to-noise ratio, SNR=，单位为分贝dB）。分别取信噪比为10dB、0dB和-10dB。

1）将叠加噪声后的语音信号实施傅里叶变换，在频域设计滤波器，滤除噪声，并进行傅里叶反变换得到降噪后的语音信号。比较原语音、加噪语音和降噪后语音的听感质量，并画出其波形比较；

2）根据1）中的频域滤波，设计相同的时域滤波器，对叠加噪声后的语音信号在时域进行滤波。能否得到跟1）中类似的结果？

要求：

1）给出你的设计思路、实验过程、实验结果以及对实验的分析与总结；并在最后附上你的代码

2）提交word版本，文件命名格式“学号-姓名-编程作业1.docx”

3）作业提交日期：2024年6月27日

**1. To explore the sensitivity of the human body to the reversal of sound signals  
Design Thought**

The aim of this experiment is to verify the insensitivity of human ears to changes in sound phase and to observe the quality changes in speech signals after frame reversal and filtering. The specific steps are as follows:

1. **Read Speech Signal**: Select a speech signal with a sampling rate of 16kHz, approximately 5 seconds in length, and ensure the signal length is 5 seconds.
2. **Phase Inversion of Signal**: Invert the phase of the original speech signal x1(t) to obtain x1(−t), and save it as a WAV file.
3. **Frame Reversal Synthesis**: Divide the original speech signal x1(t)x\_1(t)x1​(t) into frames of different lengths, invert the phase of each frame, and apply median filtering. Gradually reduce the frame length lll to generate multiple reconstructed and filtered speech signals x2(t)
4. **Comparison and Analysis**: Compare the amplitude spectra and analyze the perception of phase changes in human auditory perception and the changes in semantic intelligibility of the reconstructed speech signals.

**Experimental Process**

1. **Phase Inversion of Signal x1(t)x\_1(t)x1​(t)**:
   * Use the flipud function to invert the phase of the original speech signal and obtain x1(−t). Save this as a WAV file named flipped\_audio.wav.
2. **Frame Reversal Synthesis x2(t)x\_2(t)x2​(t)**:
   * Set different frame lengths (e.g., 500, 2000, 5000, 20000 samples), divide the original speech signal into frames, invert the phase of each frame, and apply median filtering (window size = 5).
   * Reassemble each frame to reconstruct the speech signal x2(t) under different conditions and save them as separate files.
3. **Comparison and Analysis of Signals**:
   * Visualize and compare the amplitude spectra of the original signal and the reconstructed and filtered signals.
   * Play back the signals under different conditions to observe auditory differences and semantic intelligibility.

**Results and Analysis**

* **Auditory Comparison of Inverted Signal**:
  + The inverted signal x1(−t) exhibit significant auditory differences compared to the original signal x1(t).This seems to go against our knowledge, doesn't it mean that we are sensitive to the amplitude value of the sound signal and not sensitive to the phase change? In fact, this is because we flip the speech signal for a long time, because it is not linear time invariant, so it does not meet our knowledge. We need to crop it into smaller frames so that it's linear and time invariant.
* **Effectiveness of Frame Reversal Synthesis**:
  + When the frame length is large (e.g., 10000 samples), the reconstructed signal x2 (t) sound distorted, you won't hear the original meaning
  + As the frame length decreases, especially with small frame lengths (e.g., 500 samples), the reconstructed signal x2(t) .You can hear the semantics expressed by the original speech, but the noise will be very large, there is noise, and it is sensitive to noise
* **Changes in Amplitude and Phase Spectra**:
  + The amplitude spectra of the original and reconstructed signals were plotted, demonstrating that filtering improves the semantic content of the signal, especially with larger frame lengths.
  + Changes in the phase spectrum have minimal impact on auditory perception, supporting the observation that phase inversion does not significantly affect speech intelligibility.

**Conclusion**

This experiment effectively demonstrates:

* The insensitivity of human ears to changes in sound phase, where phase-inverted signals are indistinguishable from the original signals.
* The use of frame reversal synthesis to observe signal reconstruction under varying conditions, confirming the impact of frame length on speech quality.
* The filtering of sound plays an extremely important role in the effect of flipping. If the sound is not filtered, it cannot hear the semantics at 10000Hz,5000Hz, etc., and only when it is low to 1000, we can hear the original semantics. But if it is smaller, we can hardly hear it, because we are very sensitive to noise, noise and noise make the original speech confusing. But after using the median filter, we can solve these problems, and even at 10,000 samples per frame, we can hear them. The specific sounds and code are attached to a separate file.

**2. The effect difference between frequency domain filtering and time domain filtering for noise is explored**

**Design Approach**

This experiment aims to explore the effectiveness of denoising techniques on speech signals contaminated with random white noise. The specific steps include:

1. Loading the Original Speech Signal: Load a 16kHz sampled speech signal from a file, ensuring its duration is 5 seconds.
2. Adding Random White Noise: Generate random white noise according to different Signal-to-Noise Ratios (SNR) of 10dB, 0dB, and -10dB. Add this noise to the original speech signal.
3. Frequency Domain Denoising: Perform Fourier Transform on the noisy speech signal and design a frequency domain filter to remove noise components.
4. Time Domain Denoising: Convert the frequency domain filter to its time domain counterpart (impulse response) and apply it to filter the noisy speech signal.
5. Comparative Analysis: Compare the waveforms of the original speech, noisy speech, and denoised speech signals to analyze the perceptual quality.

**Experimental Process**

1. Loading the Speech Signal:
   * Use audioread function to load the pre-recorded speech signal file with a sampling rate of 16kHz. Ensure the signal length is 5 seconds by trimming or padding.
2. Adding Random White Noise:
   * Generate random white noise with energies corresponding to SNRs of 10dB, 0dB, and -10dB. Add this noise to the original speech signal.
3. Frequency Domain Denoising:
   * Apply Fourier Transform to the noisy speech signal and design a basic low-pass filter to attenuate high-frequency noise.
   * Perform Inverse Fourier Transform to obtain the denoised speech signal in the time domain.
4. Time Domain Denoising:
   * Convert the frequency domain filter to its time domain equivalent (impulse response) using Inverse Fourier Transform.
   * Apply this time domain filter to the noisy speech signal using convolution.
   * Normalize the resulting signal to maintain its amplitude within acceptable limits.
5. Results and Analysis:
   * Frequency Domain Denoising Effectiveness: Successfully removes high-frequency noise components while preserving essential speech spectrum, resulting in clearer speech perception.
   * Time Domain Denoising Effectiveness: Utilizes the time domain filter derived from the frequency domain approach to effectively reduce noise, enhancing speech intelligibility.
   * Comparative Analysis: Waveform comparisons of original, noisy, and denoised speech signals show significant noise reduction and improved perceptual quality in both domain-based approaches.

****

Comparison of raw sound, noise sound and filtered sound under -10dB

****

Comparison of raw sound, noise sound and filtered sound under 0dB

****

Comparison of raw sound, noise sound and filtered sound under 10dB

### Conclusion

* **Frequency vs. Time Domain Approach**: Each method has its advantages and considerations depending on the SNR conditions. Frequency domain denoising excels at higher SNRs, while time domain filtering remains effective across various SNR levels.
* **Practical Implications**: Accurate SNR calculation and filter design are crucial for achieving effective speech denoising results in real-world applications.

**Appendix:**

**Exp5-1 code:**

% 读取采样率为16kHz的语音信号

[y, fs] = audioread('output\_audio\_16kHz.wav');

% 确保信号长度为5秒（如果超过5秒则截取前5秒，不足5秒则填充0）

desired\_duration = 5; % 目标信号长度为5秒

num\_samples\_target = desired\_duration \* fs;

y = y(1:min(length(y), num\_samples\_target)); % 截取前5秒或补零

% 初始化滤波器参数

window\_size = 5; % 中值滤波窗口大小

% 1. 反转信号 x\_1(t) 得到 x\_1(-t)

y\_flipped = flipud(y); % flipud 函数用于反转向量

% 保存反转声音为 WAV 文件

output\_filename\_flipped = 'flipped\_audio.wav';

audiowrite(output\_filename\_flipped, y\_flipped, fs);

disp(['Flipped audio saved to: ' output\_filename\_flipped]);

% 2. 将 x\_1(t) 分割成帧，然后反转每一帧，重新拼接成 x\_2(t)

% 设置不同的帧长度 l，观察语义变化

frame\_lengths = [500 2000 5000 20000]; % 帧长度，单位：样本数

for i = 1:length(frame\_lengths)

l = frame\_lengths(i);

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

y\_frames = reshape(y(1:num\_frames\*l), l, num\_frames); % 分割成帧

% 反转每一帧并应用中值滤波

for j = 1:num\_frames

y\_frames(:, j) = medfilt1(y\_frames(:, j), window\_size); % 中值滤波每一帧

end

% 重新拼接成 x\_2(t)

y\_reconstructed = reshape(y\_frames, [], 1);

% 保存 x\_2(t) 为 WAV 文件

output\_filename\_reconstructed = sprintf('reconstructed\_audio\_%d\_filtered.wav', l);

audiowrite(output\_filename\_reconstructed, y\_reconstructed, fs);

disp(['Reconstructed and filtered audio (l = ' num2str(l) ') saved to: ' output\_filename\_reconstructed]);

end

% 输出滤波后的声音

figure;

subplot(2, 1, 1);

plot(y);

title('Original Signal');

xlabel('Time (samples)');

ylabel('Amplitude');

subplot(2, 1, 2);

plot(y\_reconstructed);

title('Reconstructed and Filtered Signal');

xlabel('Time (samples)');

ylabel('Amplitude');

sgtitle('Signal Comparison: Original vs Reconstructed and Filtered');

% 设置图形大小

set(gcf, 'Position', [100, 100, 800, 600]);

**Exp5-2 code:**

% Load the original speech signal (assuming it's already sampled at 16kHz)

[y, fs] = audioread('output\_audio\_16kHz.wav');

% Ensure the signal length is 5 seconds

desired\_duration = 5; % Target signal duration in seconds

num\_samples\_target = desired\_duration \* fs;

y = y(1:min(length(y), num\_samples\_target)); % Trim or pad to 5 seconds

% Define SNRs to test: 10dB, 0dB, -10dB

snrs = [10, 0, -10]; % in dB

% Initialize arrays to store results

y\_noisy = cell(1, length(snrs)); % Noisy speech signals

y\_denoised\_time = cell(1, length(snrs)); % Denoised speech signals in time domain

% Generate white noise

rng('default'); % Set random seed for reproducibility

for i = 1:length(snrs)

snr\_db = snrs(i);

% Calculate noise power

signal\_power = sum(y.^2) / length(y);

noise\_power = signal\_power / (10^(snr\_db/10)); % calculate noise power

% Generate white noise with the calculated noise power

noise = sqrt(noise\_power) \* randn(size(y));

% Add noise to the original speech signal

y\_noisy{i} = y + noise;

% Frequency domain denoising

% Perform FFT

Y\_noisy = fft(y\_noisy{i});

freq\_axis = linspace(0, fs, length(Y\_noisy)); % Frequency axis

% Design frequency filters (simple low-pass filter)

cutoff\_freq = 3000; % Cut-off frequency in Hz

H = zeros(size(Y\_noisy));

H(freq\_axis < cutoff\_freq) = 1; % pass low frequencies

H(freq\_axis > fs - cutoff\_freq) = 1; % pass high frequencies

% Apply the filter in frequency domain

Y\_denoised\_freq = Y\_noisy .\* H;

% Perform inverse FFT to get time domain signal

y\_denoised\_freq = ifft(Y\_denoised\_freq, 'symmetric');

% Time domain denoising (optional for comparison)

% Design corresponding time domain filter (impulse response)

h\_time = ifft(H, 'symmetric');

% Apply time domain filter to noisy speech signal

y\_denoised\_time{i} = conv(y\_noisy{i}, h\_time, 'same');

% Normalize the denoised signals

y\_denoised\_time{i} = y\_denoised\_time{i} / max(abs(y\_denoised\_time{i}));

% Plotting for comparison

figure;

subplot(3, 1, 1);

plot(y);

title('Original Signal');

xlabel('Time (samples)');

ylabel('Amplitude');

subplot(3, 1, 2);

plot(y\_noisy{i});

title(sprintf('Noisy Signal (SNR = %d dB)', snr\_db));

xlabel('Time (samples)');

ylabel('Amplitude');

subplot(3, 1, 3);

plot(y\_denoised\_freq);

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

xlabel('Time (samples)');

ylabel('Amplitude');

sgtitle(sprintf('Speech Signal Denoising Comparison (SNR = %d dB)', snr\_db));

% Adjust figure size for clarity

set(gcf, 'Position', [100, 100, 800, 600]);

% Save denoised signals as WAV files

filename\_denoised\_freq = sprintf('denoised\_signal\_freq\_domain\_%ddb.wav', snr\_db);

audiowrite(filename\_denoised\_freq, y\_denoised\_freq, fs);

filename\_denoised\_time = sprintf('denoised\_signal\_time\_domain\_%ddb.wav', snr\_db);

audiowrite(filename\_denoised\_time, y\_denoised\_time{i}, fs);

% Save noisy signal as WAV file

filename\_noisy = sprintf('noisy\_signal\_%ddb.wav', snr\_db);

audiowrite(filename\_noisy, y\_noisy{i}, fs);

end