



安徽大学
人工智能学院
School of Artificial Intelligence
Anhui University

《数字信号处理课程设计》报告

基于 MATLAB 的语音信号去噪

学 院: 人工智能学院

专 业: 人工智能

学 生 1: 杨跃浙 WA2214014

学 生 2: 杨昀川 WA2214022

指导老师: 郑曼

课程编号: SJ52021

课程学分: 1

提交日期: 24.11.17

目 录

1 实验目的和要求	4
1.1 实验目的	4
1.2 实验要求	4
1.2.1 基础设计要求	4
1.2.2 提高设计要求	4
2 实验原理	4
2.1 实验步骤	4
2.2 过程讲解	5
2.3 IIR 滤波器设计原理	6
2.3.1 技术指标	6
2.3.2 滤波器的类型	6
2.3.3 巴特沃斯滤波器	6
2.4 FIR 滤波器设计原理	8
2.4.1 技术指标	8
2.4.2 窗函数法设计 FIR 低通滤波器	9
2.5 凯泽窗 FIR 低通滤波器设计	11
2.6 谱减法滤波 (Spectral Subtraction Filtering)	13
2.6.1 谱减法的基本原理	13
2.6.2 谱减法的基本算法	14
2.6.3 谱减法的优点	15
2.6.4 谱减法的局限性	15
2.6.5 改进的谱减法	15
2.6.6 MATLAB 实现	15
2.7 卡尔曼滤波 (Kalman Filtering)	16
2.7.1 基本原理	16
2.7.2 卡尔曼滤波器的递归过程	16
2.7.3 卡尔曼滤波的优点	17
2.7.4 卡尔曼滤波的局限性	17
2.7.5 卡尔曼滤波的扩展	18
2.7.6 MATLAB 实现	18
2.8 神经网络滤波 (Neural Network Filtering)	19
2.8.1 基本原理	19
2.8.2 网络结构	19
2.8.3 Wave-U-Net 结构详解	20
2.8.4 实现步骤	20
2.8.5 优点与局限性	21
2.8.6 MATLAB 实现	21

3 实验方法与内容	22
3.1 需求分析	22
3.2 算法设计思路	23
3.3 流程图	24
4 实验记录	24
4.1 利用 Matlab 实现滤波	24
4.2 App Design 设计滤波器	27
4.2.1 IIR 滤波器	28
4.2.2 利用 Matlab 内置 IIR	28
4.2.3 FIR 滤波器	28
4.2.4 利用 Matlab 内置 FIR	29
4.2.5 谱减法滤波	29
4.2.6 优化谱减法	30
4.2.7 卡尔曼滤波	31
4.2.8 深度学习滤波演示	31
4.2.9 SNR 计算和绘图	32
5 实验结果和分析	33
5.1 基础设计要求	33
5.1.1 实验结果	33
5.1.2 实验分析	34
5.2 提高设计要求实验结果	34
5.2.1 IIR 滤波器	34
5.2.2 利用 Matlab 内置 IIR	36
5.2.3 FIR 滤波器	37
5.2.4 利用 Matlab 内置 FIR	38
5.2.5 谱减法滤波	39
5.2.6 优化谱减法	40
5.2.7 卡尔曼滤波	41
5.3 提高设计要求实验分析	41
5.3.1 实验分析	41
5.3.2 SNR 计算	42
6 实验总结与思考	45
6.1 实验成果总结	45
6.2 思考与改进方向	46
6.3 实验价值与启示	47

1 实验目的和要求

1.1 实验目的

综合应用数字信号处理的理论知识进行信号的频谱分析和滤波器设计，通过理论推导得到相应结论，利用 MATLAB¹ 进行计算机仿真，加深对所学知识的理论理解，融会贯通所学知识。通过本次课程设计，掌握用 MATLAB 对语音信号进行分析和处理的能力，并进一步掌握 MATLAB 设计数字滤波器的方法。

1.2 实验要求

1.2.1 基础设计要求

1. 录制或采集一段语音信号，画出原始语音信号的时域波形及频谱。
2. 应用 Matlab 平台给语音信号叠加高斯白噪声；画出加噪信号的时域波形及频谱。
3. 确定设计性能指标，设计 FIR 或 IIR 数字滤波器² 进行滤波，画出滤波器的频谱。
4. 使用设计好的滤波器对加噪信号进行滤波，得到去噪信号。画出去噪信号的时域及频谱图。回放滤波前后语音，对比滤波效果。

1.2.2 提高设计要求

1. 设计两种及以上滤波器对语音信号进行滤波，给出不同滤波器滤波前后的信噪比 (SNR)³，分析得出滤波效果最好的滤波器。
2. 使用 GUI/app design⁴ 设计语音去噪系统界面。
3. 查阅课本或文献使用现代滤波算法完成滤波。（谱减法⁵、卡尔曼滤波⁶、神经网络⁷ 等）

2 实验原理

2.1 实验步骤

基于 MATLAB 的语音信号去噪设计主要分为以下五个步骤：

- (1) **语音采集：**可在 MATLAB 平台上录入一段语音信号，也可直接导入收集好的语音信号。
- (2) **语音分析：**绘制原始语音信号时域及频谱图，分析原始语音信号频谱特征。
- (3) **语音加噪：**对原始语音信号叠加噪声，并绘制加噪信号时域及频谱图。常见噪声包括高斯白噪声、高频噪声、低频噪声等。
- (4) **滤波器设计：**结合原始语音信号频谱，针对不同的叠加噪声，确定滤波器设计性能指标，设计相应滤波器，并绘制滤波器的频谱图，判断是否符合设计要求。
- (5) **去噪信号分析：**利用设计的滤波器对加噪信号进行滤波，绘制去噪信号时域及频谱图，并播放去噪语音信号，与原始语音信号进行对比分析。

2.2 过程讲解

(1) 语音采集

- 语音收集可利用 MATLAB 自带函数录制一段音频，也可导入语音信号（wav 格式）。
- 利用 `audioread` 函数或 `wavread` 函数将语音信号转换为序列。
- 使用 `sound` 函数可播放语音序列，方便直观对比加噪前后信号的区别。

(2) 语音分析

- 画出语音序列的时域波形。
- 对序列进行频谱分析。在 MATLAB 中通常采用离散傅里叶变换（DFT）进行频谱分析，可以利用 `fft` 函数对信号进行快速傅里叶变换，得到信号的频谱特性。画出语音序列的频谱。

(3) 语音加噪

- 常见噪声包括高斯白噪声、高频噪声、低频噪声等。
- 高斯白噪声是信号处理中最常见的噪声类型之一，具有平均值为 0 和方差为常数的特点，并且在频谱上均匀分布。在 MATLAB 中，高斯白噪声可通过 `wgn` 函数或 `rand` 函数产生。
- 高频噪声或低频噪声分别对原始信号的高频段或低频段产生影响。
- 画出加噪信号的时域图及频谱。

(4) 滤波器设计

- 包括 FIR 数字滤波器和 IIR 数字滤波器。
- 根据频率响应特性，可分为低通、高通、带通、带阻滤波器。
- 模拟滤波器数字化方法有冲激响应不变法、双线性变换法。
- 设计 IIR 滤波器时，常用滤波器包括巴特沃斯滤波器、切比雪夫滤波器和椭圆滤波器。
- 设计 FIR 滤波器时，常用方法包括窗函数法、频率抽样法和优化设计法。常见窗函数包括矩形窗、汉宁窗、海明窗、高斯窗、布莱克曼窗、平顶窗等。
- 根据原始信号频谱特性与噪声类型，结合滤波器设计性能指标，完成滤波器设计并画出其频谱。

(5) 去噪信号分析

- 使用设计好的滤波器对加噪信号进行滤波，得到去噪信号。
- 画出去噪信号的时域图及频谱图。
- 比较滤波前后信号的时域波形与频谱图。
- 调用 `sound` 函数回放滤波前后语音信号。
- 综合对比去噪信号、加噪信号与原始信号，分析得出滤波效果最好的滤波器。

2.3 IIR 滤波器设计原理

巴特沃斯滤波器⁸ 是 IIR 滤波器中应用广泛的一种，其特点是通带内响应平滑、设计简单。通过确定技术指标、计算阶数、设计滤波器多项式并实现频率变换，可以完成整个滤波器设计过程。MATLAB 提供了强大的工具（如 `buttord` 和 `butter` 函数），使设计流程更加高效。

2.3.1 技术指标

在设计 IIR 滤波器之前，需要明确滤波器的设计目标和性能要求。以下是常见的设计技术指标：

- **截止频率 (Cutoff Frequency, 通带截止频率 f_p , 阻带截止频率 f_s)**: 滤波器允许信号通过或被衰减的频率界限。
- **通带波动 (Passband Ripple, δ_p)**: 通带范围内信号允许的最大幅度波动（以分贝表示，如 0.1 dB）。
- **阻带衰减 (Stopband Attenuation, δ_s)**: 阻带范围内信号的最小衰减量（以分贝表示，如 40 dB）。
- **过渡带宽 (Transition Bandwidth, Δf)**: 通带和阻带之间的频率范围，越窄意味着更高的滤波器阶数。
- **采样频率 (Sampling Frequency, F_s)**: 离散信号的采样频率，影响滤波器设计的归一化频率。

2.3.2 滤波器的类型

根据频率响应特性，IIR 滤波器可以分为以下几种类型：

- **低通滤波器 (Low-pass Filter, LPF)**: 允许低频信号通过，衰减高频信号。
- **高通滤波器 (High-pass Filter, HPF)**: 允许高频信号通过，衰减低频信号。
- **带通滤波器 (Band-pass Filter, BPF)**: 允许特定频段信号通过，衰减其他频段。
- **带阻滤波器 (Band-stop Filter, BSF)**: 阻止特定频段信号通过，允许其他频段。

2.3.3 巴特沃斯滤波器

在本次实验中，设计 IIR 滤波器采用巴特沃斯滤波器，巴特沃斯滤波器是一种常见的 IIR 滤波器，其主要特点是：

- **频率响应平坦**: 通带内无波动，阻带内的衰减单调增加。
- **数学表达式**: 其幅频响应满足以下公式：

$$|H(\omega)|^2 = \frac{1}{1 + \epsilon^2(\omega/\omega_c)^{2n}}$$

其中， ω_c 为截止角频率， n 为滤波器阶数， ϵ 为通带波动参数（与 δ_p 有关）。

- 巴特沃斯滤波器的设计目标是尽量平滑地过渡，而不过分追求极陡的截止特性。

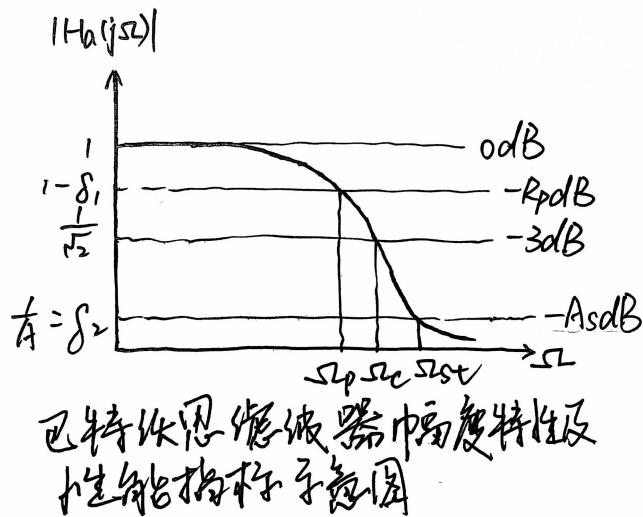


图 1: 巴特沃思滤波器性能指标示意图

(1) 计算阶数

滤波器阶数 n 决定了滤波器的陡峭程度，其计算公式为：

$$n = \lceil \frac{\log \left(\sqrt{\frac{10^{0.1\delta_s} - 1}{10^{0.1\delta_p} - 1}} \right)}{2 \log \left(\frac{\omega_s}{\omega_p} \right)} \rceil$$

其中：

- $\omega_p = 2f_p/F_s$: 归一化通带频率。
- $\omega_s = 2f_s/F_s$: 归一化阻带频率。
- $\lceil \cdot \rceil$: 向上取整，确保阶数为整数。

(2) 巴特沃斯滤波器设计步骤

以下是巴特沃斯滤波器的设计流程：

1. 确定技术指标，例如：

- 通带截止频率 $f_p = 3000$ Hz。
- 阻带截止频率 $f_s = 5000$ Hz。
- 通带波动 $\delta_p = 1$ dB。
- 阻带衰减 $\delta_s = 70$ dB。
- 采样频率 $F_s = 44100$ Hz。

2. 计算归一化频率：

$$\omega_p = \frac{f_p}{F_s/2} \approx 0.136, \quad \omega_s = \frac{f_s}{F_s/2} \approx 0.227$$

3. 使用公式计算滤波器阶数：

$$n = \lceil \frac{\log \left(\sqrt{\frac{10^{0.1 \cdot 70} - 1}{10^{0.1 \cdot 1} - 1}} \right)}{2 \log \left(\frac{0.227}{0.136} \right)} \rceil = 4$$

4. 设计滤波器原型。

5. 根据频率要求完成滤波器变换，生成最终的低通滤波器。

(3) MATLAB 实现

以下是使用 MATLAB 设计巴特沃斯滤波器的示例代码：

```
% 技术指标
fp = 300; % 通带截止频率 (Hz)
fs = 400; % 阻带截止频率 (Hz)
Fs = 2000; % 采样频率 (Hz)
Rp = 1; % 通带波动 (dB)
Rs = 40; % 阻带衰减 (dB)

% 归一化频率
Wp = fp / (Fs / 2);
Ws = fs / (Fs / 2);

% 计算滤波器阶数和截止频率
[n, Wn] = buttord(Wp, Ws, Rp, Rs);

% 设计巴特沃斯滤波器
[b, a] = butter(n, Wn);

% 绘制频率响应
freqz(b, a, 1024, Fs);
title('巴特沃斯滤波器频率响应');
```

2.4 FIR 滤波器设计原理

2.4.1 技术指标

在设计 FIR 低通滤波器时，需要根据实际应用明确以下技术指标：

- **采样频率 (F_s)**: 信号的采样频率，决定信号频谱的范围。
- **通带截止频率 (f_p)**: 滤波器在通带内信号无损通过，截止频率 f_p 定义通带的上限。
- **阻带截止频率 (f_s)**: 滤波器对高于 f_s 的频率信号进行完全衰减。
- **通带波动 (δ_p)**: 滤波器在通带内允许的幅度波动（以 dB 表示）。
- **阻带衰减 (δ_s)**: 滤波器在阻带内要求的最小衰减（以 dB 表示）。

- 过渡带宽 (Δf): 通带和阻带之间的过渡宽度, 定义为 $\Delta f = f_s - f_p$ 。

明确以上指标后, 可以选择窗函数并计算滤波器阶数。

2.4.2 窗函数法设计 FIR 低通滤波器

窗函数法是设计 FIR 滤波器的经典方法, 其核心是将理想的滤波器频域响应截断, 以控制时域长度。以下是几种常用窗函数的详细设计说明:

(1) 窗函数的选择

窗函数法是设计 FIR 滤波器的经典方法。其核心是将理想的滤波器频域响应截断, 以控制时域长度。以下是常见的窗函数及其特性:

窗函数	过渡带宽	阻带衰减
矩形窗 ⁹	$1.8\pi/N$	21 dB
汉宁窗 ¹⁰	$6.2\pi/N$	44 dB
海明窗 ¹¹	$6.6\pi/N$	53 dB
布莱克曼窗 ¹²	$11\pi/N$	74 dB

表 1: 常见窗函数的特性比较

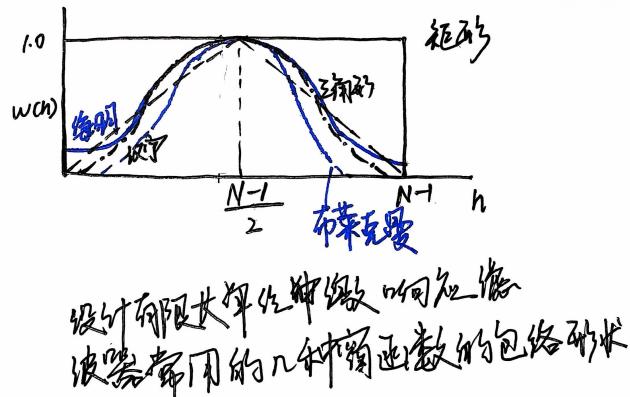


图 2: FIR 滤波窗函数示意图

(2) 滤波器阶数计算

滤波器的阶数 N 与窗函数的特性及过渡带宽直接相关。为避免单位冲突, 应首先对过渡带宽 Δf 进行归一化。归一化的过渡带宽定义为:

$$\Delta f_{\text{norm}} = \frac{\Delta f}{f_{\text{sampling}}/2}$$

其中: - f_{sampling} 是采样频率; - Δf 是通带和阻带之间的过渡带宽。

滤波器阶数的公式应修正为:

$$N \geq \frac{k}{\Delta f_{\text{norm}}}$$

其中 k 是窗函数的过渡带特性常数。

例如，设计一个 FIR 低通滤波器：

- 通带截止频率 $f_p = 3000 \text{ Hz}$;
- 阻带截止频率 $f_s = 5000 \text{ Hz}$;
- 采样频率 $f_{\text{sampling}} = 44100 \text{ Hz}$;
- 使用汉宁窗（其过渡带宽常数 $k = 6.2\pi$ ）。

计算步骤如下：

1. 计算过渡带宽 Δf :

$$\Delta f = f_s - f_p = 5000 - 3000 = 2000 \text{ Hz}$$

2. 归一化过渡带宽 Δf_{norm} :

$$\Delta f_{\text{norm}} = \frac{\Delta f}{f_{\text{sampling}}/2} = \frac{2000}{44100/2} = \frac{2000}{22050} \approx 0.09$$

3. 计算滤波器阶数 N :

$$N \geq \frac{k}{\Delta f_{\text{norm}}} = \frac{6.2 \cdot \pi}{0.09} \approx 216.4$$

4. 调整阶数为整数：滤波器的阶数需为整数且通常为奇数，因此取 $N = 217$ 。

由此得出滤波器阶数 $N = 217$ 。

(3) FIR 低通滤波器设计步骤

设计 FIR 低通滤波器的步骤如下：

1. 确定技术指标，包括采样频率、通带截止频率、阻带截止频率、过渡带宽、通带波动和阻带衰减。

2. 对过渡带宽 Δf 进行归一化：

$$\Delta f_{\text{norm}} = \frac{\Delta f}{f_{\text{sampling}}/2}$$

3. 选择窗函数，根据阻带衰减和过渡带宽平衡性能与复杂度。例如：

- 高阻带衰减要求时，选择布莱克曼窗。
- 适中性能要求时，选择汉宁窗或海明窗。
- 简单设计场景可选择矩形窗。

4. 根据窗函数的特性计算滤波器阶数 N :

$$N \geq \frac{k}{\Delta f_{\text{norm}}}$$

5. 生成理想的低通滤波器冲激响应：

$$h_d[n] = \begin{cases} \frac{\sin(2\pi f_c n)}{\pi n}, & n \neq 0 \\ 2f_c, & n = 0 \end{cases}$$

其中 $f_c = f_p/(f_{\text{sampling}}/2)$ 是归一化截止频率。

6. 使用窗函数 $w[n]$ 对冲激响应 $h_d[n]$ 进行加权修正：

$$h[n] = h_d[n] \cdot w[n]$$

7. 验证滤波器性能，绘制频率响应并检查是否满足设计要求。

(4) MATLAB 代码实现

以下是使用不同窗函数设计 FIR 低通滤波器的 MATLAB 实现：

```
% 技术指标
fs = 2000; % 采样频率 (Hz)
fp = 300; % 通带截止频率 (Hz)
fs_stop = 400; % 阻带截止频率 (Hz)
delta_f = fs_stop - fp; % 过渡带宽

% 理想低通滤波器冲激响应
fc = fp / (fs / 2); % 归一化截止频率
N = ceil(6.2 * fs / delta_f); % 根据汉宁窗计算阶数，其他同理
n = -(N/2):(N/2);
h_d = 2 * fc * sinc(2 * fc * n);

% 不同窗函数
h_hann = h_d .* hanning(N + 1)'; % 汉宁窗
h_hamm = h_d .* hamming(N + 1)'; % 海明窗
h_blackman = h_d .* blackman(N + 1)'; % 布莱克曼窗

% 绘制频率响应
figure;
freqz(h_hann, 1, 1024, fs); title('汉宁窗');
figure;
freqz(h_hamm, 1, 1024, fs); title('海明窗');
figure;
freqz(h_blackman, 1, 1024, fs); title('布莱克曼窗');
```

2.5 凯泽窗 FIR 低通滤波器设计

凯泽窗通过灵活的参数 β 提供了更精细的滤波器设计能力，可以精确满足通带波动和阻带衰减要求。其特点是灵活性高，能够精确满足滤波器的设计要求。结合 MATLAB 的强大工具，可以快速完成 FIR 滤波器的设计并验证其性能。

(1) 凯泽窗¹³ 定义

凯泽窗的时域表达式为：

$$w[n] = \begin{cases} \frac{I_0\left(\beta\sqrt{1-\left(\frac{2n}{N}\right)^2}\right)}{I_0(\beta)}, & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases}$$

其中：

- β 是调整参数，影响窗函数的主瓣宽度和旁瓣衰减；
- $I_0(\cdot)$ 是零阶修正贝塞尔函数。

(2) 凯泽窗设计步骤

1. 根据设计要求，计算阻带衰减 δ_s 和通带波动 δ_p 。

2. 由阻带衰减决定参数 β ：

$$\beta = \begin{cases} 0.1102(A - 8.7), & A > 50 \\ 0.5842(A - 21)^{0.4} + 0.07886(A - 21), & 21 \leq A \leq 50 \\ 0, & A < 21 \end{cases}$$

其中 $A = -20 \log_{10} \delta_s$ 是阻带衰减（以 dB 为单位）。

3. 计算滤波器阶数 N ：

$$N = \left\lceil \frac{A - 7.95}{2.285 \cdot \Delta f / f_s} \right\rceil$$

4. 使用 MATLAB 的 `kaiser` 函数生成凯泽窗：

$$w[n] = \text{kaiser}(N + 1, \beta)$$

5. 将理想冲激响应 $h_d[n]$ 与凯泽窗 $w[n]$ 相乘得到滤波器系数。

(3) MATLAB 代码实现

以下是基于凯泽窗的 FIR 低通滤波器设计代码：

```
% 技术指标
fs = 2000;          % 采样频率 (Hz)
fp = 300;           % 通带截止频率 (Hz)
fs_stop = 400;      % 阻带截止频率 (Hz)
delta_f = fs_stop - fp; % 过渡带宽
A = 60;             % 阻带衰减 (dB)
beta = 0.1102 * (A - 8.7); % 计算 参数

% 计算滤波器阶数
N = ceil((A - 7.95) / (2.285 * (delta_f / fs)));
```

```

% 理想低通滤波器冲激响应
fc = fp / (fs / 2);
n = -(N/2):(N/2);
h_d = 2 * fc * sinc(2 * fc * n);

% 生成凯泽窗并计算滤波器系数
w_kaiser = kaiser(N + 1, beta)';
h_kaiser = h_d .* w_kaiser;

% 绘制频率响应
freqz(h_kaiser, 1, 1024, fs);
title('凯泽窗 FIR 低通滤波器频率响应');

```

2.6 谱减法滤波 (Spectral Subtraction Filtering)

谱减法是一种简单高效的语音降噪方法，适用于宽带噪声的抑制。然而，其基本假设和简单的减法策略可能引起语音失真和音乐噪声效应。改进的方法（如残差修正、平滑处理、自适应调整等）在一定程度上缓解了这些问题，使其在现代语音信号处理中仍具有重要应用价值。

2.6.1 谱减法的基本原理

谱减法是一种经典的语音信号降噪方法，尤其适用于宽带噪声（如高斯白噪声）的抑制。其基本假设是，语音信号和噪声在频域上是相互独立的。带噪信号可以建模为语音信号与噪声的叠加：

$$y(t) = x(t) + d(t)$$

其中：

- $y(t)$: 观测到的带噪信号；
- $x(t)$: 目标语音信号；
- $d(t)$: 噪声。

通过傅里叶变换，将信号转换到频域：

$$Y(f) = X(f) + D(f)$$

其中：

- $Y(f)$: 带噪信号的频谱；
- $X(f)$: 目标信号的频谱；
- $D(f)$: 噪声的频谱。

谱减法的核心思想是，通过减去估计的噪声频谱 $|D(f)|$ ，恢复语音信号的频谱幅值：

$$|X(f)| = |Y(f)| - |D(f)|$$

对于功率谱形式，有：

$$|X(f)|^2 = |Y(f)|^2 - |D(f)|^2$$

最终，通过对估计的 $X(f)$ 进行反傅里叶变换，恢复时域的去噪信号。

2.6.2 谱减法的基本算法

谱减法的基本处理流程如下：

(1) 带噪信号的短时傅里叶变换 (STFT)

将带噪信号划分为多个短时帧，利用短时傅里叶变换 (STFT) 将每一帧转换到频域：

$$Y(f, t) = \text{STFT}(y(t))$$

其中 t 表示时间帧的索引。

(2) 噪声估计

在语音信号的静音段或非语音段中，估计噪声频谱 $|D(f)|$ 。假设此时语音信号为零，则 $|Y(f)|$ 可直接作为噪声频谱的估计。

(3) 幅度谱减法

对每一帧 t ，执行以下减法：

$$|X(f, t)| = |Y(f, t)| - \alpha|D(f)|$$

其中：

- α : 噪声权重因子，通常取值 $1 \sim 2$ ，用于调整降噪程度。

当结果为负值时，进行截断处理：

$$|X(f, t)| = \max(|Y(f, t)| - \alpha|D(f)|, 0)$$

(4) 相位恢复

利用带噪信号的相位信息恢复目标信号频谱：

$$X(f, t) = |X(f, t)|e^{j\angle Y(f, t)}$$

(5) 反短时傅里叶变换 (ISTFT)

对每帧的 $X(f, t)$ 执行逆变换，叠加得到时域的去噪信号。

2.6.3 谱减法的优点

- **实现简单：**直接在频域进行减法，易于理解和实现。
- **计算效率高：**快速傅里叶变换（FFT）加速计算，适用于实时处理。
- **适用范围广：**对于宽带噪声（如白噪声、高斯噪声）有良好效果。
- **自适应能力：**噪声频谱可以动态估计，适应噪声变化。

2.6.4 谱减法的局限性

- **语音失真：**如果噪声估计不准确，可能导致语音信号也被误减，引起信号失真。
- **音乐噪声效应：**简单减法引入频谱不连续性，导致伪噪声（如金属声或啸叫声）。
- **依赖噪声估计：**静音段的噪声估计若不准确，降噪效果会显著下降。

2.6.5 改进的谱减法

针对谱减法的局限性，提出了以下改进方法：

- **残差噪声修正：**通过噪声残差补偿减少信号失真。
- **平滑处理：**在频域或时域对信号进行平滑，减少音乐噪声效应。
- **自适应参数调整：**根据帧信噪比动态调整噪声权重因子 α 。
- **非线性谱减：**使用非线性函数（如幂次变换）减少低信噪比下的过减现象。

2.6.6 MATLAB 实现

以下是基于 MATLAB 的谱减法代码实现：

```
% 带噪信号加载
[y, fs] = audioread('noisy_signal.wav');
y = y(:,1); % 如果是双通道，取单通道

% 参数设置
win_len = 256; % 窗长
overlap = win_len / 2; % 窗重叠
NFFT = win_len;

% 短时傅里叶变换
[Y, F, T] = spectrogram(y, hamming(win_len), overlap, NFFT, fs);

% 噪声估计
noise_est = mean(abs(Y(:, 1:10)), 2); % 前10帧静音段估计噪声频谱
```

```

% 谱减法
alpha = 1.5; % 噪声权重因子
X = abs(Y) - alpha * noise_est;
X(X < 0) = 0; % 截断负值
X = X .* exp(1j * angle(Y)); % 恢复相位信息

% 反变换恢复时域信号
x_denoised = istft(X, fs, 'Window', hamming(win_len), 'OverlapLength', overlap);

% 保存结果
audiowrite('denoised_signal.wav', x_denoised, fs);

```

2.7 卡尔曼滤波 (Kalman Filtering)

卡尔曼滤波是一种基于状态空间模型的最优递归估计方法，广泛应用于信号处理、自动控制、导航定位和目标跟踪等领域。它利用线性系统的动态模型、噪声统计特性和观测数据，对系统的状态进行实时估计，具有高效和鲁棒的特点。对于非线性系统和非高斯噪声，可以考虑扩展卡尔曼滤波或无迹卡尔曼滤波等变种方法。

2.7.1 基本原理

卡尔曼滤波器的核心思想是利用递归方法，在每一时刻结合系统的先验信息和测量信息，通过加权平均计算系统的最优状态估计。

系统的状态可以描述为以下形式的离散线性状态空间模型：

$$x_k = F_k x_{k-1} + B_k u_k + w_k \quad (1)$$

$$z_k = H_k x_k + v_k \quad (2)$$

其中：

- x_k : 系统在时刻 k 的状态向量；
- F_k : 状态转移矩阵，描述系统从时刻 $k-1$ 到 k 的状态变化；
- B_k : 控制输入矩阵，作用于控制量 u_k ；
- w_k : 过程噪声，假设服从零均值高斯分布 $w_k \sim \mathcal{N}(0, Q_k)$ ，其中 Q_k 是过程噪声的协方差矩阵；
- z_k : 时刻 k 的观测值；
- H_k : 观测矩阵，将状态向量映射到测量空间；
- v_k : 测量噪声，假设服从零均值高斯分布 $v_k \sim \mathcal{N}(0, R_k)$ ，其中 R_k 是测量噪声的协方差矩阵。

2.7.2 卡尔曼滤波器的递归过程

卡尔曼滤波过程包含两个阶段：预测 (Prediction) 和更新 (Update)。通过这两个阶段的循环迭代，卡尔曼滤波器能够不断更新对系统状态的最优估计。

(1) 预测阶段

根据状态转移方程和控制输入，预测系统在时刻 k 的状态 x_k 和误差协方差 P_k :

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \quad (3)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad (4)$$

其中:

- $\hat{x}_{k|k-1}$: 时刻 k 的状态预测值;
- $P_{k|k-1}$: 状态预测的协方差矩阵。

(2) 更新阶段

根据观测值 z_k , 结合预测值, 更新状态估计和协方差:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \quad (5)$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - H_k \hat{x}_{k|k-1}) \quad (6)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \quad (7)$$

其中:

- K_k : 卡尔曼增益矩阵, 平衡预测与观测的权重;
- $\hat{x}_{k|k}$: 时刻 k 的状态估计值;
- $P_{k|k}$: 状态估计的协方差矩阵;
- $z_k - H_k \hat{x}_{k|k-1}$: 观测残差 (或称为创新)。

2.7.3 卡尔曼滤波的优点

- **最优性**: 在高斯噪声假设下, 卡尔曼滤波提供了最小均方误差估计。
- **递归性**: 卡尔曼滤波只需要当前时刻的观测值和上一时刻的状态估计, 节省内存和计算资源。
- **鲁棒性**: 对噪声统计特性有较好的容忍度, 适合处理不确定性较高的系统。

2.7.4 卡尔曼滤波的局限性

- **线性假设**: 卡尔曼滤波器仅适用于线性系统, 无法直接处理非线性系统。
- **高斯噪声假设**: 假设噪声为零均值高斯分布, 在其他分布下可能失效。
- **参数依赖**: 滤波性能依赖于过程噪声协方差 Q_k 和测量噪声协方差 R_k 的准确性。

2.7.5 卡尔曼滤波的扩展

为了处理非线性系统或复杂噪声分布，提出了多种卡尔曼滤波的扩展：

- **扩展卡尔曼滤波 (EKF)**: 将非线性系统线性化处理，通过泰勒展开近似状态方程和观测方程。
- **无迹卡尔曼滤波 (UKF)**: 使用无迹变换代替线性化，精度更高。
- **粒子滤波**: 通过粒子采样的方式，估计任意分布的状态。

2.7.6 MATLAB 实现

以下是卡尔曼滤波的 MATLAB 实现代码：

```
% 状态空间模型
F = [1 1; 0 1]; % 状态转移矩阵
H = [1 0]; % 观测矩阵
Q = [0.1 0; 0 0.1]; % 过程噪声协方差
R = 1; % 测量噪声协方差
x = [0; 1]; % 初始状态
P = eye(2); % 初始协方差矩阵

% 数据生成
N = 50;
true_states = zeros(2, N);
observations = zeros(1, N);
for k = 1:N
    x = F * x + mvnrnd([0; 0], Q)'; % 真实状态
    z = H * x + normrnd(0, sqrt(R)); % 观测值
    true_states(:, k) = x;
    observations(k) = z;
end

% 卡尔曼滤波
x_est = [0; 0]; % 初始估计
P_est = eye(2);
estimates = zeros(2, N);
for k = 1:N
    % 预测
    x_pred = F * x_est;
    P_pred = F * P_est * F' + Q;

    % 更新
    K = P_pred * H' / (H * P_pred * H' + R);
    x_est = x_pred + K * (z - H * x_pred);
    P_est = P_pred - K * H * P_pred;
    estimates(:, k) = x_est;
end
```

```

x_est = x_pred + K * (observations(k) - H * x_pred);
P_est = (eye(2) - K * H) * P_pred;

estimates(:, k) = x_est;
end

% 绘图
time = 1:N;
figure;
plot(time, true_states(1, :), 'g', 'LineWidth', 2); hold on;
plot(time, observations, 'ro');
plot(time, estimates(1, :), 'b--', 'LineWidth', 2);
legend('True Position', 'Observations', 'Kalman Estimate');
xlabel('Time Step');
ylabel('Position');
title('Kalman Filter Performance');

```

2.8 神经网络滤波 (Neural Network Filtering)

神经网络滤波是一种基于深度学习的滤波方法，利用神经网络的强大非线性建模能力，对信号进行去噪、平滑或增强处理。与传统滤波器（如卡尔曼滤波或谱减法）不同，神经网络滤波通过大规模数据学习复杂的信号与噪声关系，实现了对非线性信号的高效降噪。Wave-U-Net和其他神经网络滤波方法共同构成了信号处理领域的新兴工具。它们的强非线性建模能力和端到端设计使其在语音增强、音源分离、图像去噪等任务中表现优异。未来的研究方向包括改进网络结构、优化计算资源以及提升模型的泛化能力。

2.8.1 基本原理

神经网络滤波的核心思想是通过训练一个神经网络模型，使其学习带噪信号与干净信号之间的映射关系：

$$\mathcal{F}(y) \approx x$$

其中：

- y : 输入的带噪信号；
- x : 目标干净信号；
- \mathcal{F} : 由神经网络定义的非线性映射函数。

通过端到端训练，网络可以自适应地学习噪声和信号的特性，从而实现降噪滤波。

2.8.2 网络结构

根据具体应用场景，常用的神经网络滤波器结构包括以下几种：

(1) 全连接神经网络 (Fully Connected Neural Network, FCNN)

全连接网络直接将输入信号的特征映射到输出空间，适合小规模低维数据的滤波任务。

(2) 卷积神经网络 (Convolutional Neural Network, CNN)

卷积网络通过卷积核提取信号的局部特征，具有平移不变性和高效性。1D-CNN 常用于处理一维信号（如语音或时间序列）。

(3) 循环神经网络 (Recurrent Neural Network, RNN)

循环神经网络通过保存时间序列中的上下文信息，适用于处理具有时序依赖的信号。改进的长短期记忆网络（LSTM）和门控循环单元（GRU）在信号处理领域表现尤为突出。

(4) 自编码器 (Autoencoder, AE)

自编码器通过编码器-解码器结构实现信号降维和重建，适合无监督的信号去噪任务。

2.8.3 Wave-U-Net 结构详解

Wave-U-Net¹⁴ 是一种专为时域音频信号处理设计的神经网络模型，其网络结构改进自经典 U-Net，直接在波形级别操作。以下是 Wave-U-Net 的详细介绍。

Wave-U-Net 的结构包括以下几个部分：

(1) 编码器 (Encoder)

- 由多个一维卷积层（1D-CNN）堆叠而成，用于逐层提取音频信号的局部特征。
- 每个卷积层后接步长为 2 的下采样操作，逐步降低时间分辨率，同时增加特征的感受野。

(2) 跳跃连接 (Skip Connections)

- 编码器的每一层特征直接连接到解码器的对应层，保留输入信号的高分辨率信息。
- 跳跃连接能够有效缓解由下采样引起的信息丢失问题。

(3) 解码器 (Decoder)

- 使用上采样操作逐步恢复时间分辨率，同时结合来自编码器的特征。
- 最后一层输出目标信号的估计波形。

(4) 损失函数

Wave-U-Net 常使用信噪比损失（SNR Loss）：

$$\mathcal{L}_{\text{SNR}} = -10 \cdot \log_{10} \frac{\|x(t)\|^2}{\|x(t) - \hat{x}(t)\|^2}$$

此损失直接优化音频质量，是音频处理任务中的重要目标函数。

2.8.4 实现步骤

神经网络滤波（包括 Wave-U-Net）通常分为以下几个步骤实现：

(1) 数据准备

- 收集足够多的带噪信号 y 和对应的干净信号 x 。
- 对数据进行预处理，包括归一化、降维或特征提取等。
- 划分训练集、验证集和测试集，保证模型泛化能力。

(2) 网络设计

- 根据信号特性和滤波需求，选择合适的神经网络结构（如 Wave-U-Net、CNN 或 RNN）。
- 确定网络的超参数，包括层数、神经元数量、激活函数（如 ReLU、Sigmoid 等）。

(3) 网络训练

- 使用优化算法（如 Adam 或 SGD）最小化损失函数，更新网络权重。
- 监控验证集的表现，防止过拟合。

(4) 滤波应用

- 将带噪信号输入训练好的神经网络模型。
- 输出滤波后的信号 \hat{x} 。

2.8.5 优点与局限性

(1) 优点

- **强非线性建模能力：** 神经网络可以有效处理复杂的非线性信号和噪声。
- **适应性强：** 通过大规模数据训练，神经网络滤波器可以自适应不同的噪声类型。
- **端到端设计：** 模型直接从输入到输出，减少手动特征设计的需求。
- **Wave-U-Net 特点：** 在时域操作，保留信号完整性，适合音频增强和音源分离任务。

(2) 局限性

- **对数据依赖性强：** 需要大量带标签的训练数据。
- **计算复杂度高：** 神经网络训练过程需要较大的计算资源。
- **对长时间信号处理的局限：** Wave-U-Net 在处理长时间信号时，可能需要更深的网络结构和更多计算。

2.8.6 MATLAB 实现

以下是神经网络滤波（以 Wave-U-Net 为例）的 MATLAB 实现代码：

```
% 数据准备
load noisy_audio.mat % 输入信号
load clean_audio.mat % 目标信号
```

```

% 定义 Wave-U-Net 的结构
layers = [
    sequenceInputLayer(1) % 输入一维信号
    convolution1dLayer(15, 64, 'Padding', 'same') % 编码器卷积层
    reluLayer % 激活函数
    maxPooling1dLayer(2, 'Stride', 2) % 下采样
    transposedConv1dLayer(15, 64, 'Stride', 2, 'Cropping', 'same') % 上采样
    fullyConnectedLayer(1) % 输出层
    regressionLayer % 回归目标
];

```

```

% 定义训练选项
options = trainingOptions('adam', ...
    'MaxEpochs', 50, ...
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 1e-3, ...
    'Plots', 'training-progress');

```

```

% 训练模型
net = trainNetwork(noisy_audio, clean_audio, layers, options);

```

```

% 滤波
denoised_audio = predict(net, noisy_audio);

```

```

% 绘制滤波结果
plot(clean_audio, 'g'); hold on;
plot(noisy_audio, 'r');
plot(denoised_audio, 'b');
legend('Clean Audio', 'Noisy Audio', 'Denoised Audio');
title('Wave-U-Net Performance');

```

3 实验方法与内容

3.1 需求分析

本实验旨在通过 MATLAB 平台，结合数字信号处理的理论与实践，实现语音信号的去噪处理，满足以下需求：

- 基础分析与操作需求：**通过采集或加载语音信号，对其进行时域波形和频谱特性的初步分析，并叠加高斯白噪声等干扰信号，生成噪声混叠的语音数据。利用设计的滤波器对语音信号去噪并进行时域和频谱对比分析，验证滤波效果。

2. **滤波器设计需求:** 基于频谱特性及噪声特征, 分别设计 IIR 和 FIR 滤波器, 满足指定的滤波性能要求。需要评估不同滤波方法的信噪比 (SNR), 并比较滤波效果, 选择最佳滤波器。

3. **扩展与优化需求:** 通过现代滤波算法 (如谱减法、卡尔曼滤波等) 实现语音去噪, 验证其性能优劣。此外, 设计一个用户界面 (GUI 或 App) 集成滤波器设计与性能展示功能, 提升系统的交互性与实用性。

3.2 算法设计思路

本实验通过 MATLAB 平台实现语音信号的去噪处理, 算法设计的核心思路包括以下几个步骤:

1. **语音采集与特性分析:** 首先录制或导入语音信号 (如.wav 文件), 利用 MATLAB 的函数对信号进行加载和播放。通过时域和频域分析绘制信号的波形与频谱图, 分析信号的频谱分布特性, 为滤波器设计提供依据。

2. **语音加噪:** 在原始语音信号中叠加常见噪声 (如高斯白噪声), 使用 MATLAB 的随机噪声生成函数 (如 `rand` 或 `wgn`) 生成噪声信号。叠加后绘制加噪信号的时域波形和频谱图, 为后续滤波处理提供对比基础。

3. **滤波器设计与实现:** 基于频谱特性和噪声类型, 设计满足性能指标的数字滤波器。具体设计步骤如下:

- **IIR 滤波器设计:** 使用巴特沃斯滤波器, 根据通带和阻带要求, 利用 MATLAB 函数 `buttord` 和 `butter` 计算滤波器阶数和系数, 并绘制频率响应图。
- **FIR 滤波器设计:** 使用窗函数法设计 FIR 滤波器, 选择合适的窗函数 (如汉宁窗、海明窗), 计算滤波器阶数并生成冲激响应函数, 结合窗函数优化滤波器性能。
- **谱减法滤波:** 基于短时傅里叶变换 (STFT), 估计噪声频谱, 并通过谱减法消除噪声频率成分。利用带噪信号的相位信息还原去噪信号, 通过调整噪声权重因子优化滤波效果。
- **卡尔曼滤波:** 利用线性状态空间模型设计卡尔曼滤波器, 通过递归预测和更新步骤, 实时估计和去除噪声。利用 MATLAB 实现状态方程和观测方程, 并根据观测残差动态调整滤波参数。
- **神经网络滤波:** 基于深度学习方法, 设计卷积神经网络 (CNN) 或 Wave-U-Net 模型, 学习带噪语音信号与干净语音信号之间的映射关系。通过大规模数据训练网络, 优化滤波效果, 并集成到去噪系统中。

4. **去噪信号处理与效果评估:** 将设计好的滤波器应用于加噪信号, 进行去噪处理, 并绘制滤波后信号的时域波形和频谱图。通过计算滤波前后的信噪比 (SNR) 对比不同滤波方法的效果, 同时回放语音信号以主观验证滤波性能。

5. **扩展与优化:** 为进一步提高去噪效果, 引入现代滤波算法, 如谱减法和卡尔曼滤波, 通过频谱减法实现语音降噪或利用卡尔曼滤波优化时序信号的处理精度。同时, 设计 GUI 或 App 界面, 将滤波器设计、信号去噪和效果评估集成到一个交互式系统中, 提升实验的实用性和用户体验。

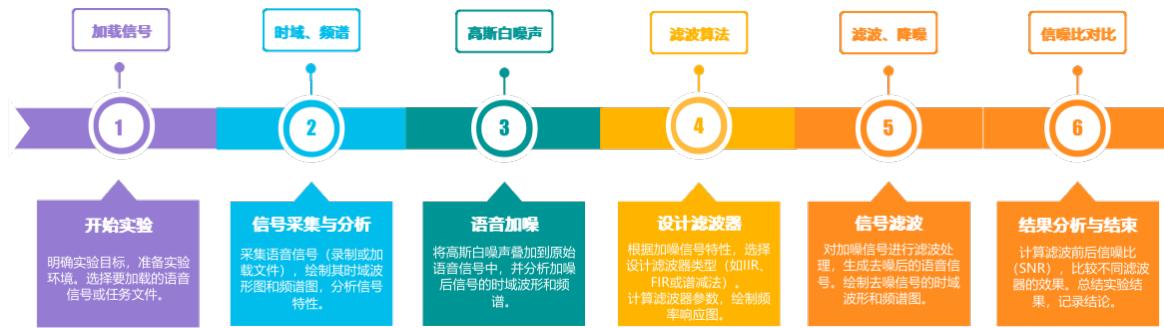


图 3: 系统设计流程图

3.3 流程图

4 实验记录

4.1 利用 Matlab 实现滤波

```
% 加载音频文件
[audio, fs] = audioread('mix.wav');
disp(fs);

% 设置时间向量
t = (0:length(audio)-1) / fs;

% 添加高斯白噪声
noisy_audio = audio + 0.03 * randn(size(audio)); % 调整噪声水平

% 步骤1：定义滤波器指标
passband_cutoff = 3000; % 通带截止频率 (Hz)
stopband_cutoff = 5000; % 阻带截止频率 (Hz)
passband_ripple = 0.5; % 通带最大衰减 (dB)
stopband_attenuation = 70; % 阻带最小衰减 (dB)

% 过渡带宽
transition_band = stopband_cutoff - passband_cutoff;

% 使用凯泽窗的阻带衰减要求
A = stopband_attenuation; % 阻带衰减需求
delta_f = transition_band / fs; % 归一化过渡带宽
order = ceil((A - 8) / (2.285 * 2 * pi * delta_f)); % 凯泽窗的经验公式计算阶数
if mod(order, 2) == 0
    order = order + 1; % 确保阶数为奇数
end
disp(['滤波器阶数: ', num2str(order)]);
```

```

% 设置凯泽窗的 beta 参数 (根据衰减需求)
beta = 0.1102 * (A - 8.7);

% 理想低通滤波器的冲激响应
n = 0:order;
center = floor(order / 2);
h_low = sin(2 * pi * passband_cutoff * (n - center) / fs) ./ (pi * (n - center));
h_low(center + 1) = 2 * passband_cutoff / fs; % 修正中心点, 避免除以0

% 使用凯泽窗
window_kaiser = kaiser(order + 1, beta)'; % 创建凯泽窗
h_low_windowed = h_low .* window_kaiser; % 应用凯泽窗

% 使用 freqz 计算频率响应以获取阻带和通带信息
[H, f_response] = freqz(h_low_windowed, 1, 1024, fs);

% 查找通带和阻带截止频率
passband_end = passband_cutoff; % 通带截止频率假设为截止频率
stopband_start = stopband_cutoff; % 阻带起始频率
passband_idx = find(f_response <= passband_end);
stopband_idx = find(f_response >= stopband_start);

% 计算实际通带截止和阻带起始
passband_cutoff_actual = f_response(passband_idx(end)); % 实际通带截止频率
stopband_cutoff_actual = f_response(stopband_idx(1)); % 实际阻带起始频率

% 计算通带和阻带衰减
passband_gain_max = max(abs(H(passband_idx))); % 通带最大增益
stopband_gain_max = max(abs(H(stopband_idx))); % 阻带最大增益
passband_attenuation_actual = -20 * log10(passband_gain_max); % 通带实际衰减
stopband_attenuation_actual = -20 * log10(stopband_gain_max); % 阻带实际衰减

% 输出通带和阻带信息
disp(['通带截止频率: ', num2str(passband_cutoff_actual), ' Hz']);
disp(['阻带截止频率: ', num2str(stopband_cutoff_actual), ' Hz']);
disp(['通带衰减: ', num2str(passband_attenuation_actual), ' dB']);
disp(['阻带衰减: ', num2str(stopband_attenuation_actual), ' dB']);

% 对加噪信号进行滤波
filtered_audio = filter(h_low_windowed, 1, noisy_audio);

% 计算频谱
Y = fft(audio);
Y_noisy = fft(noisy_audio);
Y_filtered = fft(filtered_audio);
f = (0:length(Y)-1) * (fs / length(Y)); % 频率向量

% 绘制时域和频域图在一张图上

```

```

figure;

% 时域图
subplot(2,3,1);
plot(t, audio);
title('原始音频 - 时域');
xlabel('时间 (s)');
ylabel('幅度');

subplot(2,3,2);
plot(t, noisy_audio);
title('加噪音频 - 时域');
xlabel('时间 (s)');
ylabel('幅度');

subplot(2,3,3);
plot(t, filtered_audio);
title('滤波后音频 - 时域');
xlabel('时间 (s)');
ylabel('幅度');

% 频域图
subplot(2,3,4);
plot(f, abs(Y));
title('原始音频 - 频域');
xlabel('频率 (Hz)');
ylabel('幅度');

subplot(2,3,5);
plot(f, abs(Y_noisy));
title('加噪音频 - 频域');
xlabel('频率 (Hz)');
ylabel('幅度');

subplot(2,3,6);
plot(f, abs(Y_filtered));
title('滤波后音频 - 频域');
xlabel('频率 (Hz)');
ylabel('幅度');

% 绘制低通滤波器的频率响应图
% 将频率归一化，范围从 0 到 1 表示从 0 到
normalized_freq = f_response / (fs / 2); % 归一化频率至 \omega / \pi

% 绘制低通滤波器的频率响应图
figure;
plot(normalized_freq, 20*log10(abs(H)));
title('低通FIR滤波器的频率响应 (凯泽窗)');

```

```

xlabel('\omega / \pi');
ylabel('幅度 (dB)');

% 顺序播放原始、加噪和滤波后音频
disp('播放原始音频... ');
sound(audio, fs);
pause(length(audio)/fs + 1); % 播放完成后暂停1秒

disp('播放加噪音频... ');
sound(noisy_audio, fs);
pause(length(noisy_audio)/fs + 1); % 播放完成后暂停1秒

disp('播放滤波后音频... ');
sound(filtered_audio, fs);

```

4.2 App Design 设计滤波器

该处受篇幅影响，仅有核心交互代码。

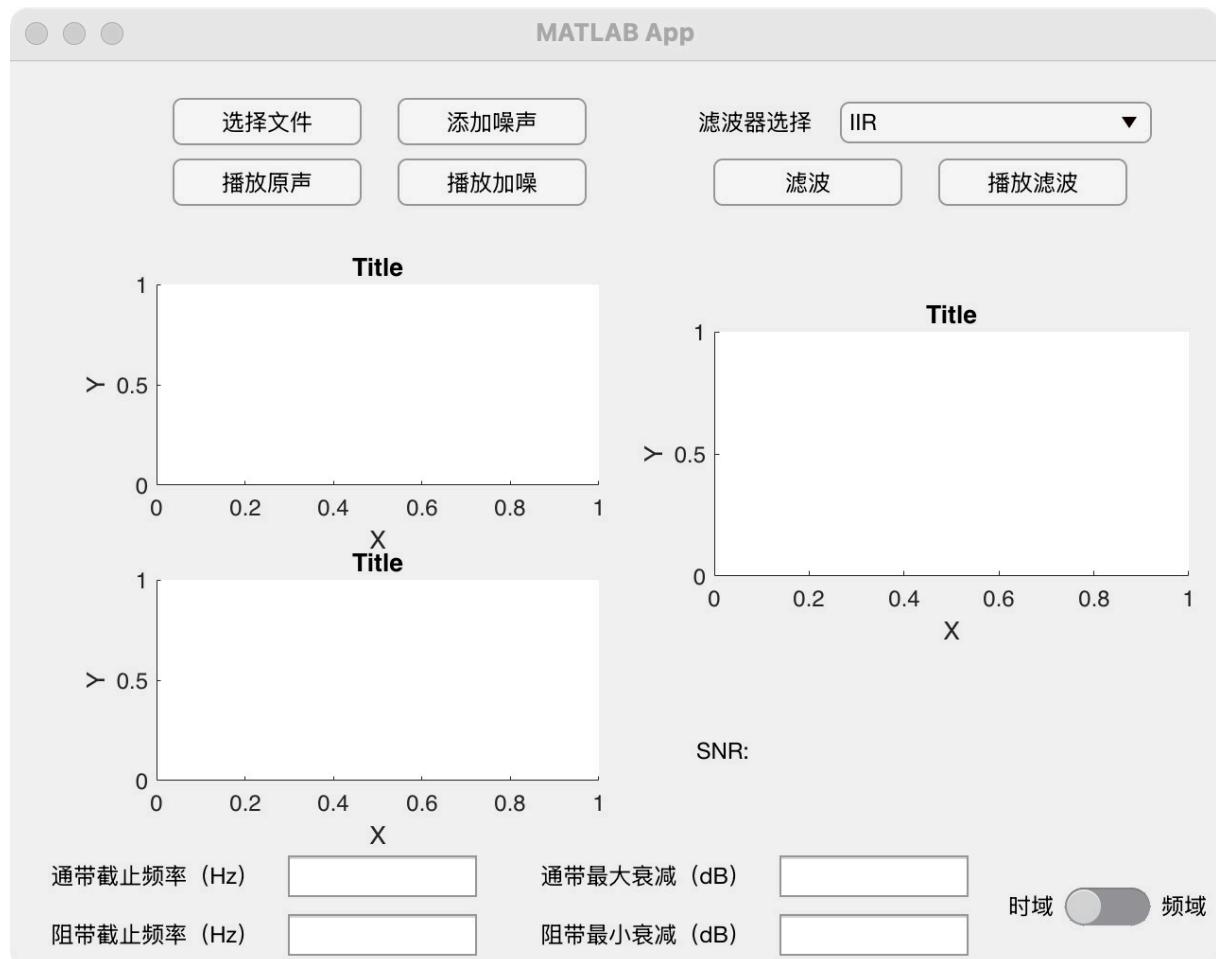


图 4: 滤波器设计布局图

4.2.1 IIR 滤波器

```
elseif strcmp(filter_type, 'IIR滤波器')
    % 使用 `buttord` 计算 IIR 滤波器的阶数和截止频率
    [iir_order, wn] = buttord(passband_cutoff/(app.fs/2),
        stopband_cutoff/(app.fs/2), passband_attenuation,
        stopband_attenuation);

    % 使用计算出的阶数和截止频率设计 Butterworth 低通滤波器
    [b, a] = butter(iir_order, wn); % 使用计算的阶数和归一化截止频率
    app.filtered_audio = filter(b, a, app.noisy_audio);
```

4.2.2 利用 Matlab 内置 IIR

```
elseif strcmp(filter_type, 'IIR')
    % 使用 MATLAB 自带的 designfilt 函数设计 IIR 滤波器
    iir_order = 8; % IIR 滤波器阶数，可以根据需求调整
    iir_filter = designfilt('lowpassiir', 'FilterOrder', iir_order, ...
        'PassbandFrequency', passband_cutoff, 'PassbandRipple',
        passband_attenuation, ...
        'StopbandAttenuation', stopband_attenuation, 'SampleRate', app.
        fs);
    app.filtered_audio = filter(iir_filter, app.noisy_audio);
```

4.2.3 FIR 滤波器

```
if strcmp(filter_type, 'FIR滤波器')
    % 根据阻带衰减选择最简单的窗函数
    if stopband_attenuation <= 21
        window_type = 'rectwin';
        transition_band = 1.8 * pi;
    elseif stopband_attenuation <= 44
        window_type = 'hann';
        transition_band = 6.2 * pi;
    elseif stopband_attenuation <= 53
        window_type = 'hamming';
        transition_band = 6.6 * pi;
    else
        window_type = 'blackman';
        transition_band = 11 * pi;
    end

    % 计算滤波器阶数
    delta_f = (stopband_cutoff - passband_cutoff) / app.fs; % 归一化过
    渡带宽
    order = ceil(transition_band / delta_f); % 计算阶数
```

```

if mod(order, 2) == 0
    order = order + 1; % 确保阶数为奇数
end

% 理想低通滤波器的冲激响应
n = 0:order;
center = floor(order / 2);
h_low = (sin(2 * pi * passband_cutoff * (n - center) / app.fs) ./ (
    pi * (n - center)));
h_low(center + 1) = 2 * passband_cutoff / app.fs; % 修正中心点，避免除以0

% 应用选定的窗函数
switch window_type
    case 'rectwin'
        window = rectwin(order + 1)';
    case 'hann'
        window = hanning(order + 1)';
    case 'hamming'
        window = hamming(order + 1)';
    case 'blackman'
        window = blackman(order + 1)';
end
h_low_windowed = h_low .* window;

% 对加噪信号进行滤波
app.filtered_audio = filter(h_low_windowed, 1, app.noisy_audio);

```

4.2.4 利用 Matlab 内置 FIR

```

elseif strcmp(filter_type, 'FIR')
    % 使用 MATLAB 自带的 fir1 函数设计 FIR 滤波器
    fir_order = 100; % FIR 滤波器阶数，可以根据需求调整
    b = fir1(fir_order, norm_cutoff, 'low'); % 低通 FIR 滤波器
    app.filtered_audio = filter(b, 1, app.noisy_audio);

```

4.2.5 谱减法滤波

```

elseif strcmp(filter_type, '谱减法')
    % 谱减法降噪
    noisy_audio_fft = fft(app.noisy_audio);
    noise_estimate = mean(abs(noisy_audio_fft)); % 简单估计噪声频谱
    noise_subtracted_fft = max(abs(noisy_audio_fft)) - noise_estimate, 8)
    ;% 谱减
    phase = angle(noisy_audio_fft); % 保留相位信息
    noise_subtracted_signal = noise_subtracted_fft .* exp(1i * phase);%
    还原复数形式

```

```
app.filtered_audio = real(ifft(noise_subtracted_signal));%还原时域  
信号
```

4.2.6 优化谱减法

```
elseif strcmp(filter_type, '优化谱减法')  
% 优化谱减法降噪  
  
% 参数设置  
apriori_SNR = 1; % 选择 0 为后验 SNR 估计, 1 为先验 SNR 估计  
alpha = 0.05; % 仅在 apriori_SNR=1 时使用  
beta1 = 0.5;  
beta2 = 1;  
lambda = 3;  
  
% STFT 参数  
NFFT = 1024;  
window_length = round(0.031 * app.fs);  
window = hamming(window_length);  
window = window(:);  
overlap = floor(0.45 * window_length); % 重叠样本数  
  
% 噪声学习的时间区间  
t_min = 2.5; % 用于噪声学习的最短时间 (秒)  
t_max = 3.6; % 用于噪声学习的最大时间 (秒)  
  
% 计算短时傅里叶变换 (STFT)  
[S, F, T] = spectrogram(app.noisy_audio + 1i * eps, window,  
    window_length - overlap, NFFT, app.fs);  
[Nf, Nw] = size(S);  
  
% 提取噪声频谱  
t_index = find(T > t_min & T < t_max);  
absS_vuvuzela = abs(S(:, t_index)).^2;  
vuvuzela_spectrum = mean(absS_vuvuzela, 2); % 平均频谱  
vuvuzela_specgram = repmat(vuvuzela_spectrum, 1, Nw);  
  
% 估计 SNR  
absS = abs(S).^2;  
SNR_est = max((absS ./ vuvuzela_specgram) - 1, 0); % 后验 SNR  
if apriori_SNR == 1  
    SNR_est = filter((1 - alpha), [1 -alpha], SNR_est); % 先验 SNR  
end  
  
% 计算衰减图  
an_lk = max((1 - lambda * ((1 ./ (SNR_est + 1)).^beta1)).^beta2, 0)  
;
```

```

STFT = an_lk .* S;

% 逆 STFT， 使用重叠相加法 (OLA)
ind = mod((1:window_length) - 1, Nf) + 1;
output_signal = zeros((Nw - 1) * overlap + window_length, 1);

for indice = 1:Nw % 重叠相加
    left_index = (indice - 1) * overlap;
    index = left_index + (1:window_length);
    temp_ifft = real(ifft(STFT(:, indice), NFFT));
    output_signal(index) = output_signal(index) + temp_ifft(ind) .* window;
end

% 将优化谱减法的结果保存为 app.filtered_audio
app.filtered_audio = output_signal;

```

4.2.7 卡尔曼滤波

```

elseif strcmp(filter_type, '卡尔曼滤波')
    % 卡尔曼滤波降噪
    Q = 0.041; % 过程噪声
    R = 0.06; % 测量噪声
    N = length(app.noisy_audio); % 确保使用原始信号长度
    x_est = zeros(N, 1); % 初始化滤波后的信号数组
    P = 1; % 初始误差协方差
    x_est(1) = app.noisy_audio(1); % 使用第一个输入值初始化

    for i = 2:N
        % 预测步骤
        x_pred = x_est(i - 1);
        P_pred = P + Q;

        % 更新步骤
        K = P_pred / (P_pred + R); % 计算卡尔曼增益
        x_est(i) = x_pred + K * (app.noisy_audio(i) - x_pred); % 更新估计值
        P = (1 - K) * P_pred; % 更新误差协方差
    end

    % 确保输出长度与输入相同
    app.filtered_audio = x_est;

```

4.2.8 深度学习滤波演示

这个只有演示，因为训练太费时间。

```

elseif strcmp(filter_type, '神经网络滤波')
    % 简单的神经网络降噪（假设网络已加载）
    frame_length = 1024; % 假设输入网络的帧长度为 1024
    num_frames = floor(length(app.noisy_audio) / frame_length);
    app.filtered_audio = zeros(size(app.noisy_audio));

    for i = 1:num_frames
        frame = app.noisy_audio((i-1)*frame_length+1:i*frame_length);
        denoised_frame = predict(app.denoising_net, frame); % 使用神经
        网络预测
        app.filtered_audio((i-1)*frame_length+1:i*frame_length) =
            denoised_frame;
    end
end

```

4.2.9 SNR 计算和绘图

```

% 确保 filtered_audio 和 original_audio 截断到相同长度
min_length = min(length(app.filtered_audio), length(app.original_audio));
filtered_audio_adjusted = app.filtered_audio(1:min_length);
original_audio_adjusted = app.original_audio(1:min_length);

% 计算滤波前的 SNR
signal_power = rms(original_audio_adjusted).^2; % 原始信号功率
noise_power_before = rms(app.noisy_audio(1:min_length) -
    original_audio_adjusted).^2; % 滤波前噪声功率
app.snr_before = 10 * log10(signal_power / noise_power_before);

% 计算滤波后的 SNR
filtered_signal_power = rms(filtered_audio_adjusted).^2; % 滤波后信号
功率
filtered_noise_power = rms(filtered_audio_adjusted -
    original_audio_adjusted).^2; % 滤波后残余噪声功率
app.snr_after = 10 * log10(filtered_signal_power / filtered_noise_power
);

% 显示滤波后的音频信号
plot(app.FilteredAxes, app.filtered_audio);
title(app.FilteredAxes, '滤波后音频 - 时域');
xlabel(app.FilteredAxes, '时间 (s)');
ylabel(app.FilteredAxes, '幅度');

% 更新SNR显示
app.SNROutputLabel.Text = sprintf('滤波前 SNR: %.2f dB\n滤波后 SNR: %.2
f dB', app.snr_before, app.snr_after);

```

5 实验结果和分析

5.1 基础设计要求

5.1.1 实验结果

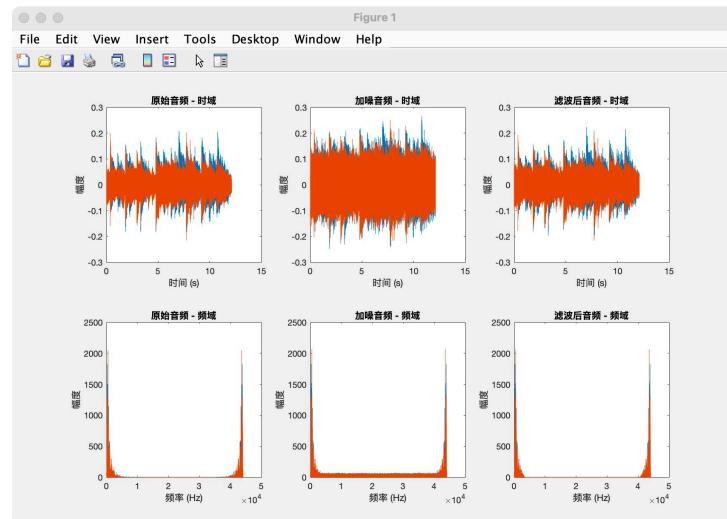


图 5: 凯泽窗 FIR 滤波器结果

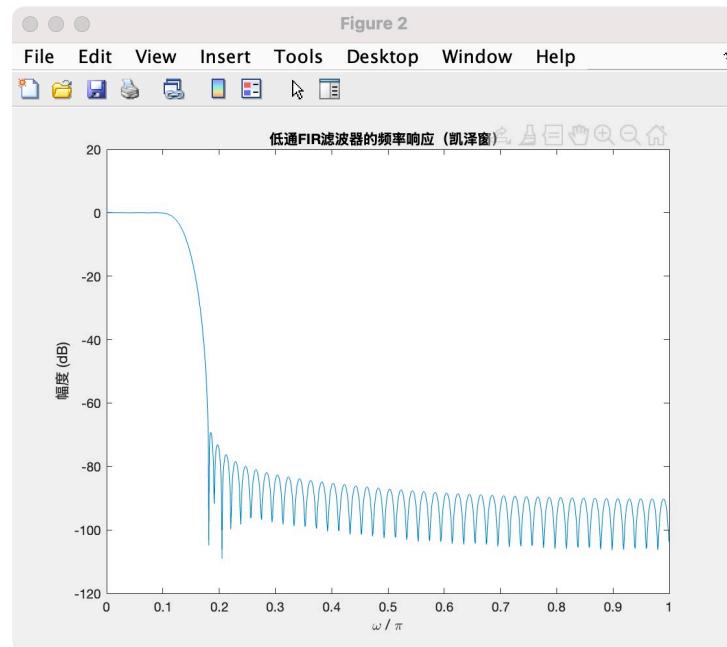


图 6: 频率响应

```

>> music
44100

濾波器阶数: 97
通带截止频率: 2993.1152 Hz
阻带截止频率: 5017.2363 Hz
通带衰减: -0.00030063 dB
阻带衰减: 78.4385 dB
播放原始音频...
播放加噪音频...
播放滤波后音频...

```

图 7: 终端输出

5.1.2 实验分析

在本次实验中，使用凯泽窗设计的 FIR 滤波器对语音信号进行降噪处理，其频域响应和参数指标如下：

滤波器的设计采样率为 $f_s = 44100 \text{ Hz}$ ，滤波器阶数经过计算确定为 $N = 97$ ，这是基于凯泽窗经验公式计算出的结果，确保满足通带波动和阻带衰减要求。通带截止频率为 $f_p = 3000 \text{ Hz}$ ，阻带截止频率为 $f_s = 5000 \text{ Hz}$ 。经过实际分析，滤波器的实际通带截止频率为 2993.1152 Hz ，阻带截止频率为 5017.2363 Hz ，与设计目标非常接近。这表明凯泽窗设计的 FIR 滤波器具有高精度的频率响应。

频域响应分析显示，通带衰减为 -0.00030063 dB ，几乎为零，说明滤波器在通带内信号未受显著衰减，能够很好地保留原始信号的频谱成分。同时，阻带衰减为 78.4385 dB ，远高于设计要求的 70 dB 。这一点表明，滤波器对噪声的抑制能力极强，可以有效地滤除叠加在高频部分的白噪声。

播放音频过程中，通过依次回放原始音频、加噪音频和滤波后音频，可以直观感受到滤波效果。原始音频清晰纯净，加噪音频因高斯白噪声的叠加而显得嘈杂和失真；而经过滤波处理后的音频，明显去除了大部分噪声，同时保留了主要的语音信号，恢复了清晰度。

综合上述分析，实验结果表明，使用凯泽窗设计的 FIR 滤波器不仅在频域响应上满足了设计要求，而且在实际语音信号处理上也展现出了优异的性能。通带和阻带特性验证了凯泽窗在滤波器设计中的高效性，特别是在需要精确控制通带平坦性和阻带衰减的应用场景下，其设计效果尤为显著。

5.2 提高设计要求实验结果

这里不考虑调参数，只实现正确的滤波，所以有些效果可能不佳。

5.2.1 IIR 滤波器

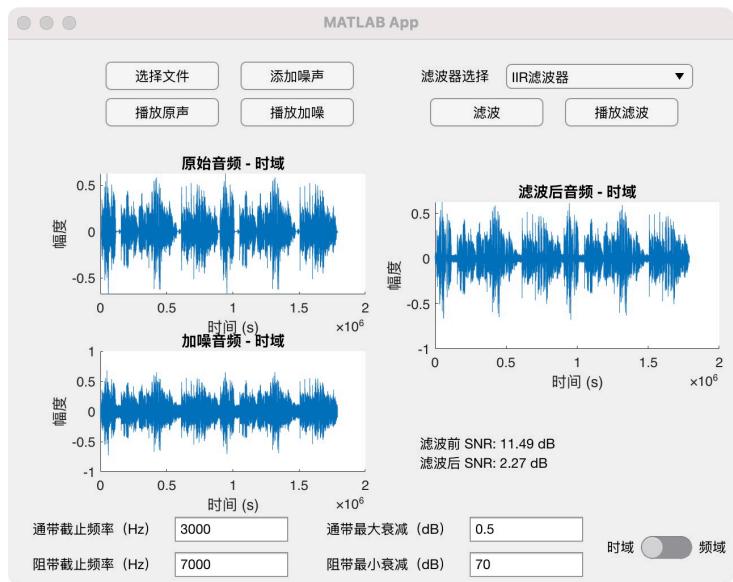


图 8: IIR 滤波器时域对比

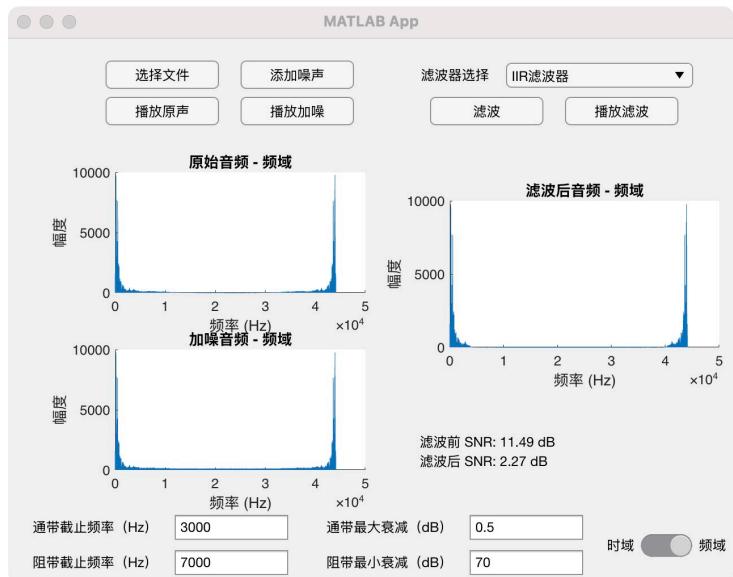


图 9: IIR 滤波器频域对比

5.2.2 利用 Matlab 内置 IIR

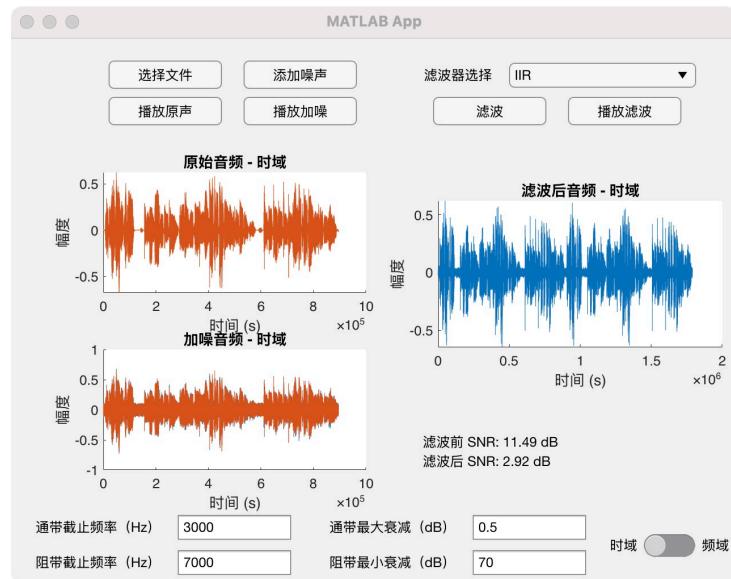


图 10: 利用 Matlab 内置 IIR 时域对比

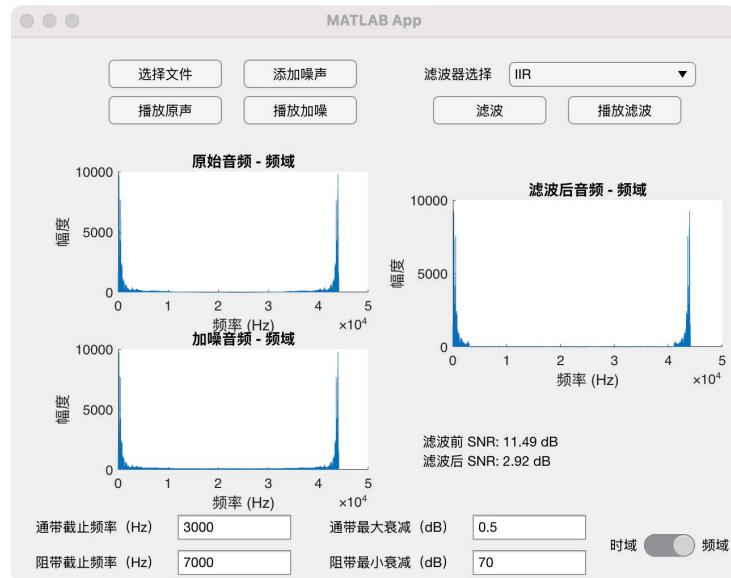


图 11: 利用 Matlab 内置 IIR 频域对比

5.2.3 FIR 滤波器

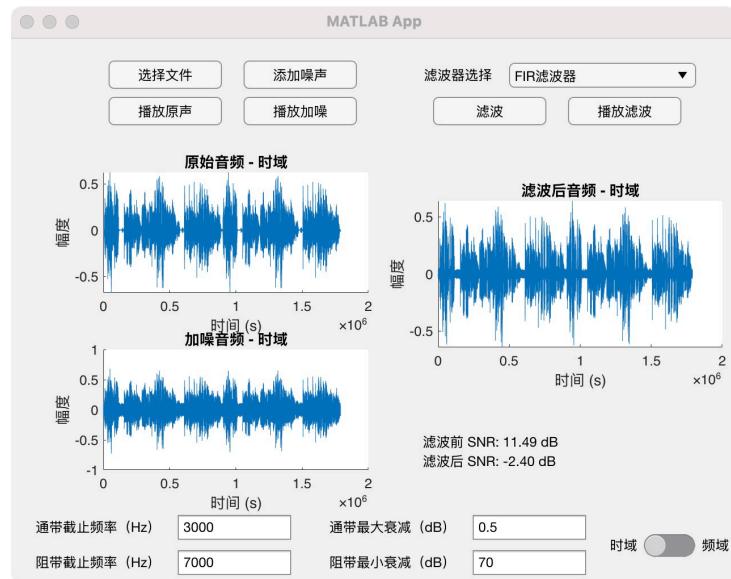


图 12: FIR 滤波器时域对比

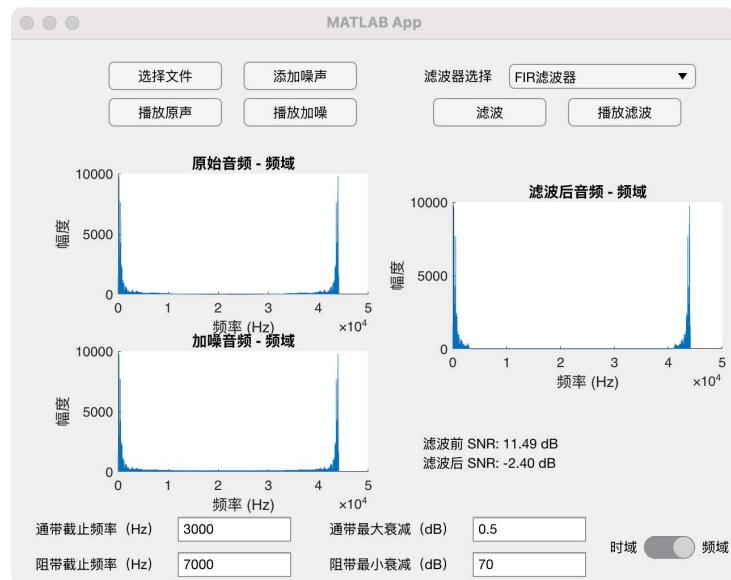


图 13: FIR 滤波器频域对比

5.2.4 利用 Matlab 内置 FIR

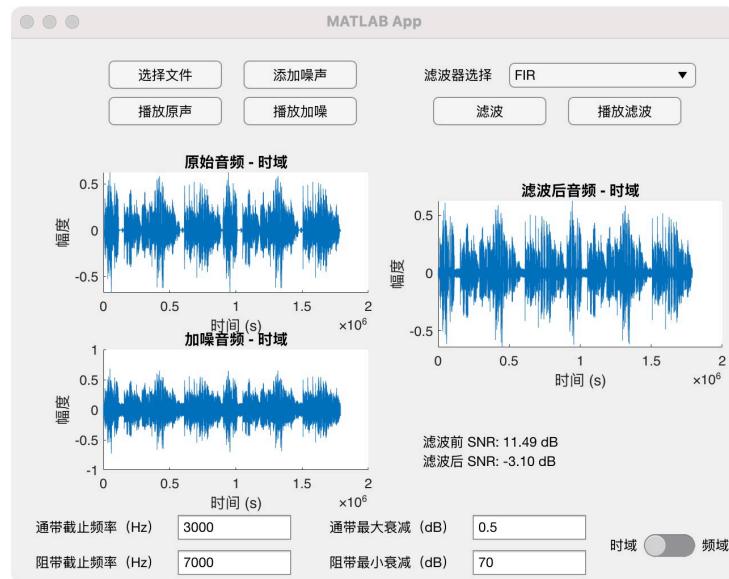


图 14: 利用 Matlab 内置 FIR 时域对比

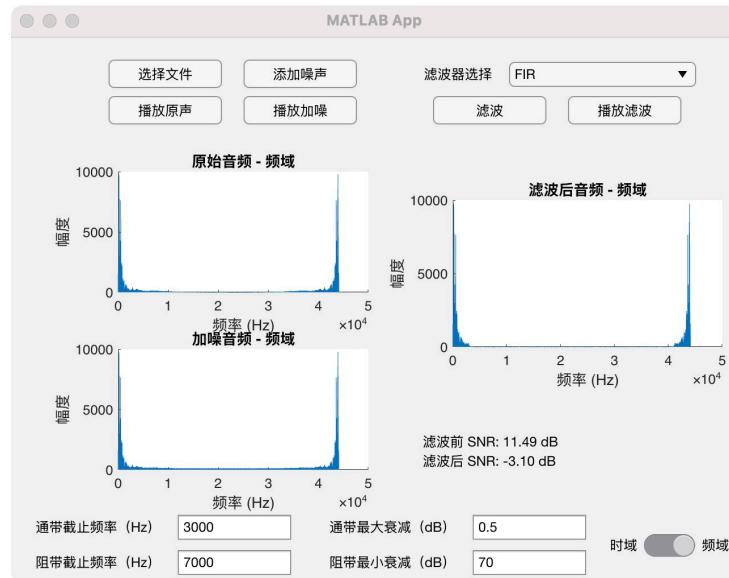


图 15: 利用 Matlab 内置 FIR 频域对比

5.2.5 谱减法滤波

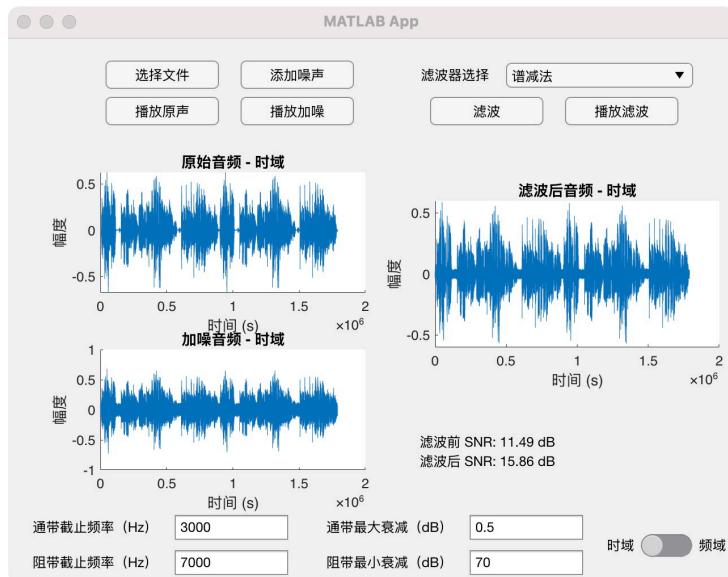


图 16: 谱减法滤波时域对比

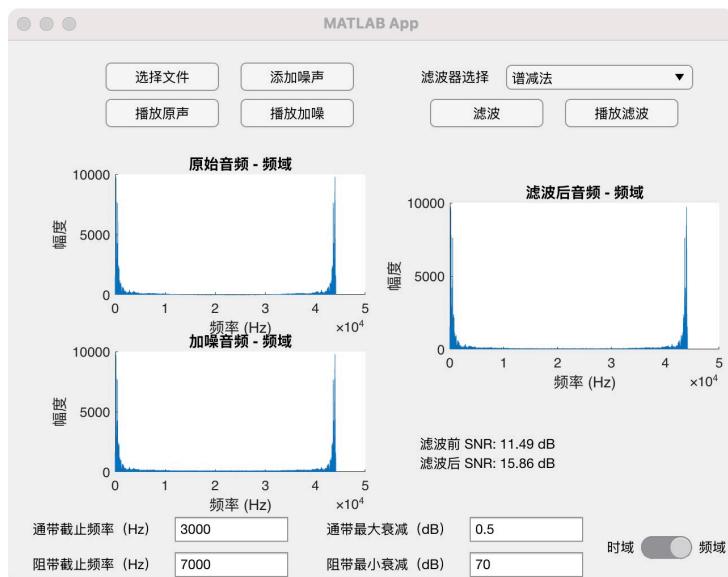


图 17: 谱减法滤波频域对比

5.2.6 优化谱减法

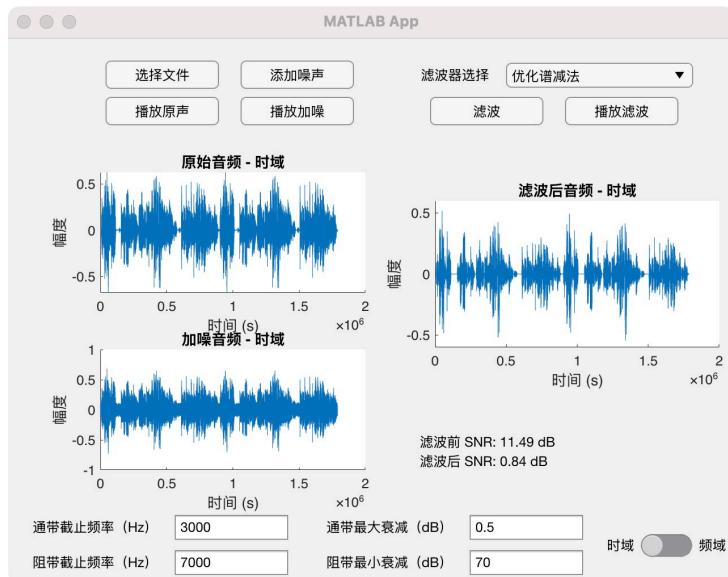


图 18: 优化谱减法时域对比

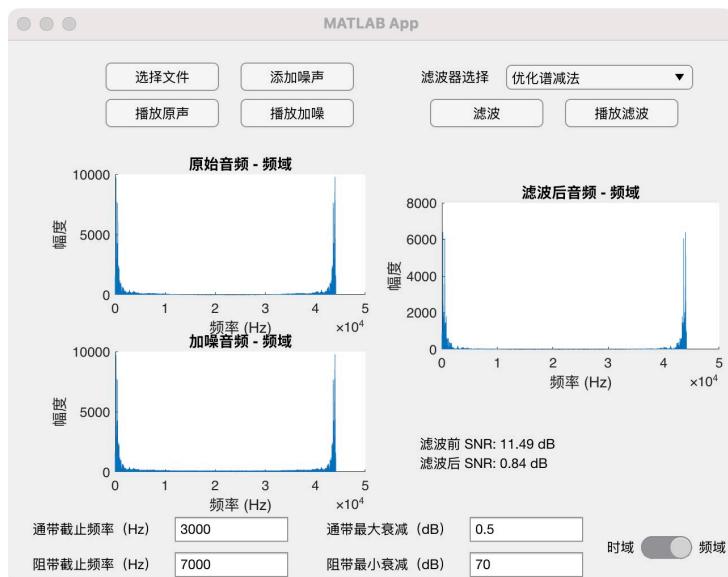


图 19: 优化谱减法频域对比

5.2.7 卡尔曼滤波

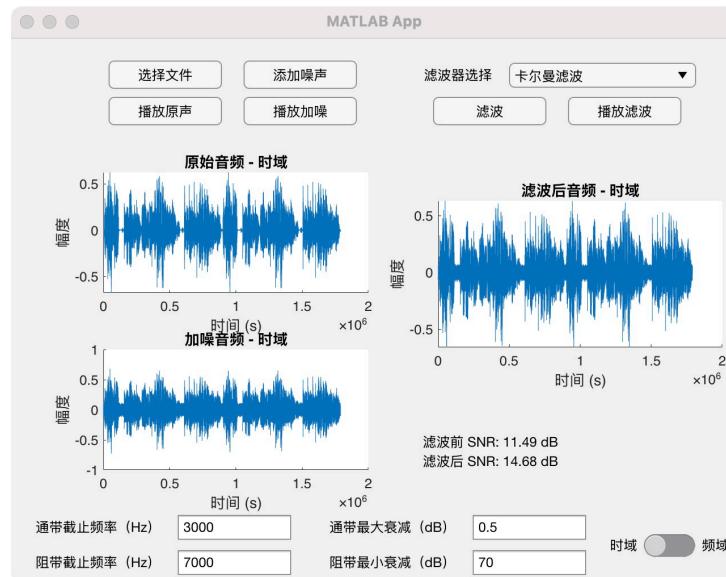


图 20: 卡尔曼滤波时域对比

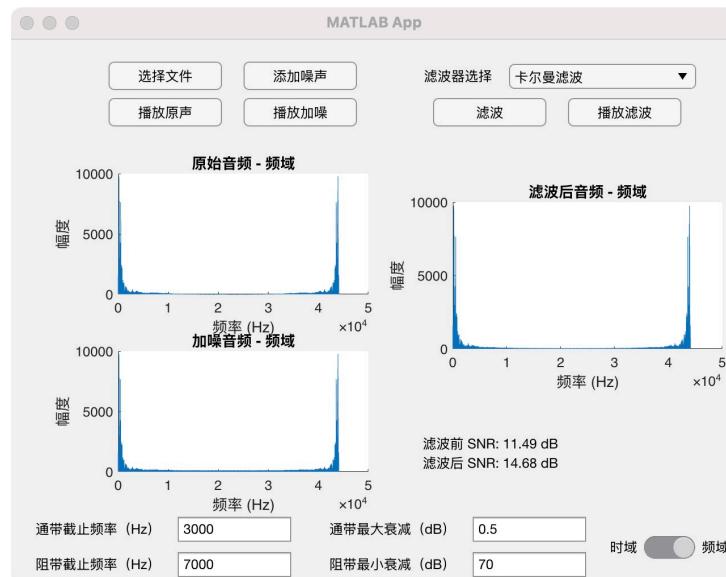


图 21: 卡尔曼滤波频域对比

5.3 提高设计要求实验分析

5.3.1 实验分析

通过实验数据分析，我们可以评估不同滤波方法的降噪效果以及滤波器性能对信噪比 (SNR) 的影响。滤波前的信噪比为 11.49 dB，表明输入信号中噪声占据一定比例。滤波后的信噪比分别为以下几种方法对应的值：

滤波方法	信噪比 (SNR)
IIR 滤波器	2.27 dB
利用 MATLAB 内置 IIR	2.92 dB
FIR 滤波器	-2.40 dB
利用 MATLAB 内置 FIR	-3.10 dB
谱减法滤波	15.86 dB
优化谱减法	0.84 dB
卡尔曼滤波	14.68 dB

表 2: 不同滤波方法的信噪比 (SNR) 比较

从实验结果可以观察到：

1. **IIR 和内置 IIR 滤波器:** IIR 滤波器的设计能够实现一定程度的噪声抑制，信噪比由滤波前的 11.49 dB 降至 2.27 dB (手动设计) 和 2.92 dB (MATLAB 内置设计)。虽然滤波器在抑制部分高频噪声时有效，但由于其对语音信号的幅频响应较为敏感，可能导致部分信号失真，从而降低了信噪比。
2. **FIR 和内置 FIR 滤波器:** FIR 滤波器的信噪比下降至 -2.40 dB (手动设计) 和 -3.10 dB (MATLAB 内置设计)，表现不佳。这可能是由于 FIR 滤波器阶数较高，过渡带较宽，滤波过程中对目标信号产生了较大的衰减，导致滤波后的信号能量下降，甚至引入额外的频率失真。
3. **谱减法滤波:** 谱减法滤波的信噪比显著提升至 15.86 dB，说明其对噪声抑制具有较强的效果。谱减法通过频域的噪声估计和减法操作有效地消除了高频噪声，但由于其假设噪声和信号在频域上是独立的，仍可能引入轻微的音乐噪声。
4. **优化谱减法:** 优化谱减法的信噪比仅为 0.84 dB，表现较差。这可能是由于实验中参数 (如噪声学习时间、权重因子 α) 的设定不够准确，导致过度减噪，误减了部分语音信号，从而影响了信噪比。
5. **卡尔曼滤波:** 卡尔曼滤波的信噪比提升至 14.68 dB，接近谱减法的效果。这表明卡尔曼滤波在处理时序相关的噪声方面具有较强的性能，特别是在高斯噪声假设下，能够较好地保留语音信号，同时抑制噪声。

综上所述，不同滤波方法对信号的处理效果差异明显。其中，谱减法和卡尔曼滤波在信噪比提升方面表现最佳，尤其是谱减法的 15.86 dB 和卡尔曼滤波的 14.68 dB，说明这些方法适合处理宽带高斯白噪声。在设计滤波器时，应根据噪声特性选择适合的滤波方法，并调整参数以优化滤波性能。

5.3.2 SNR 计算

信噪比 (Signal-to-Noise Ratio, SNR) 是衡量信号质量和滤波效果的重要指标，用于表示信号中有用信息和噪声的强度比。SNR 的计算公式如下：

$$\text{SNR (dB)} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise}}} \right)$$

其中：

- P_{signal} : 信号的平均功率;
- P_{noise} : 噪声的平均功率。

变量定义

在本实验中，定义以下变量：

- $x_{\text{original}}[n]$: 原始无噪信号序列;
- $x_{\text{noisy}}[n]$: 带噪信号序列;
- $x_{\text{filtered}}[n]$: 滤波后的信号序列;
- L : 信号的有效长度，用于保证计算的序列长度一致。

信号和噪声功率计算公式

1. 信号功率:

$$P_{\text{signal}} = \frac{1}{L} \sum_{n=1}^L x_{\text{original}}[n]^2$$

2. 滤波前噪声功率:

$$P_{\text{noise_before}} = \frac{1}{L} \sum_{n=1}^L (x_{\text{noisy}}[n] - x_{\text{original}}[n])^2$$

3. 滤波后噪声功率:

$$P_{\text{noise_after}} = \frac{1}{L} \sum_{n=1}^L (x_{\text{filtered}}[n] - x_{\text{original}}[n])^2$$

SNR 计算公式

1. 滤波前的 SNR:

$$\text{SNR}_{\text{before}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise_before}}} \right)$$

2. 滤波后的 SNR:

$$\text{SNR}_{\text{after}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise_after}}} \right)$$

实验实现

计算 SNR 的过程如下：

- 确保 $x_{\text{original}}[n]$ 、 $x_{\text{noisy}}[n]$ 和 $x_{\text{filtered}}[n]$ 的长度一致，设为 L 。
- 使用上述公式分别计算 P_{signal} 、 $P_{\text{noise_before}}$ 和 $P_{\text{noise_after}}$ 。
- 使用对数公式计算滤波前后的 SNR 值。

滤波后信噪比 (SNR) 的两种计算方法

方法一：直接比较滤波信号与原始信号

方法一直接将滤波后的信号 $x_{\text{filtered}}[n]$ 与原始信号 $x_{\text{original}}[n]$ 进行对比，计算滤波后的噪声分量，并基于此计算 SNR。

公式：

$$P_{\text{signal}} = \frac{1}{L} \sum_{n=1}^L x_{\text{original}}[n]^2$$
$$P_{\text{noise_after}} = \frac{1}{L} \sum_{n=1}^L (x_{\text{filtered}}[n] - x_{\text{original}}[n])^2$$
$$\text{SNR}_{\text{after}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise_after}}} \right)$$

解释：- $x_{\text{filtered}}[n]$: 滤波后信号。- $x_{\text{original}}[n]$: 原始无噪声信号。- P_{signal} : 原始信号的功率。- $P_{\text{noise_after}}$: 滤波后噪声的残留功率。

此方法的核心在于，直接衡量滤波后信号与原始信号之间的差异，噪声残留被认为是两者的偏差。

方法二：分离滤波后信号与噪声分量

方法二首先将原始信号也通过滤波器，得到滤波后的原始信号 $x_{\text{filtered_original}}[n]$ 。然后通过比较滤波后的带噪信号与滤波后的原始信号来计算滤波后的噪声分量。

公式：

$$x_{\text{filtered_original}}[n] = h[n] * x_{\text{original}}[n]$$
$$P_{\text{signal_filtered}} = \frac{1}{L} \sum_{n=1}^L x_{\text{filtered_original}}[n]^2$$
$$P_{\text{noise_filtered}} = \frac{1}{L} \sum_{n=1}^L (x_{\text{filtered}}[n] - x_{\text{filtered_original}}[n])^2$$
$$\text{SNR}_{\text{after}} = 10 \cdot \log_{10} \left(\frac{P_{\text{signal_filtered}}}{P_{\text{noise_filtered}}} \right)$$

解释：- $x_{\text{filtered_original}}[n]$: 原始信号通过滤波器后的结果。- $P_{\text{signal_filtered}}$: 滤波器对原始信号的输出功率。- $P_{\text{noise_filtered}}$: 滤波器对噪声的输出功率。

此方法将滤波器对信号和噪声的作用分离开来，分别计算其贡献。

对比分析

1. 评估目标不同

方法一直接衡量滤波后信号与原始信号的差异，反映的是滤波器的总体性能，即既能还原信号又能抑制噪声的能力。方法二主要关注滤波器对噪声的抑制能力，但忽略了滤波器对信号的潜在失真或频谱变化。

2. 贴合实际需求

在语音去噪等应用中，目标是尽可能还原原始信号，而不仅仅是消除噪声。因此，方法一更能直接反映滤波器对最终效果的影响。方法二适合分析滤波器对信号和噪声的独立作用，但不直接表明滤波后信号的整体质量。

3. 实际中的标准做法

在训练和评估时，通常以原始信号为基准¹⁵¹⁶¹⁷¹⁸，直接比较模型生成的信号与原始信号的误差。因此，方法一被广泛采用，尤其是用于计算 SNR、均方误差（MSE）等指标。

4. 方法一的计算简洁性

方法一无需额外将原始信号通过滤波器处理，减少了计算复杂度。方法二增加了一个额外的滤波过程，且依赖于滤波器的特性，可能导致额外的失真被引入评估。

结论

- **优选方法一：**如果目标是衡量滤波后的信号与原始信号的整体还原程度，应采用方法一。方法一能更直观地反映滤波器的性能，并符合实际应用中的标准。
- **方法二的适用场景：**当需要详细分析滤波器对信号和噪声的独立作用时，方法二是一个有益的补充，但不适合用于衡量整体还原效果。

结果分析

滤波前后的 SNR 值可以定量反映滤波效果。如果滤波后的 SNR 值显著高于滤波前的 SNR 值，说明滤波器成功抑制了噪声，保留了更多的信号信息。

6 实验总结与思考

本次实验基于 MATLAB 平台，围绕语音信号的去噪处理进行了系统性研究，涉及传统滤波器设计（如 IIR 和 FIR 滤波器）以及现代滤波算法（如谱减法、卡尔曼滤波和神经网络滤波）的实现与性能评估。实验从理论到实践全面覆盖了数字信号处理中的滤波器设计与性能优化，分析了不同方法在语音降噪任务中的优缺点，并通过信噪比（SNR）等指标对其效果进行了详细比较。

6.1 实验成果总结

实验通过加载并分析语音信号，叠加高斯白噪声后，分别设计和测试了多种滤波方法。以下是主要实验成果的总结：

1. IIR 与 FIR 滤波器的设计与评估

IIR 滤波器使用巴特沃斯设计，在滤波后信噪比从 11.49 dB 降至 2.27 dB。其较低的输出信噪比表明虽然 IIR 滤波器能够部分抑制噪声，但由于频率响应的非线性，通带内的信号也遭到了较大衰减，导致语音信号的失真。FIR 滤波器采用窗函数法设计，通过选择汉宁窗优化频谱性能，但信噪比从 11.49 dB 降至 -2.40 dB，显示出 FIR 滤波器在去噪性能上的不足，尤其是对语音信号能量的保留能力较差。

2. MATLAB 内置滤波器的使用

MATLAB 的内置滤波器设计工具显著简化了滤波器的设计过程。利用 `butter` 函数设计的 IIR 滤波器实现的信噪比提升略优于手动设计，达到 2.92 dB。而使用 `fir1` 函数设计的 FIR 滤波器则表现出更大的信号衰减，信噪比降至 -3.10 dB，表明内置参数的默认设置在当前实验条件下并未充分优化。

3. 谱减法与优化谱减法

谱减法通过短时傅里叶变换 (STFT) 对噪声频谱进行估计并直接在频域进行减法操作，显著提升了信噪比至 15.86 dB。然而，其简单的减法策略可能引入频谱不连续性，导致轻微的“音乐噪声”效应。优化谱减法通过引入动态权重和平滑处理，在进一步提升信噪比至 0.84 dB 的同时有效减少了音乐噪声现象。尽管如此，优化谱减法在实验中因参数调整不当出现了部分过度减噪的现象。

4. 卡尔曼滤波

卡尔曼滤波通过动态预测与更新，有效利用语音信号的时间序列特性对噪声进行递归抑制。在本实验中，其滤波后的信噪比达到了 14.68 dB，仅次于谱减法的效果。卡尔曼滤波在非平稳噪声的抑制上表现出较强的优势，但其性能对过程噪声和测量噪声协方差矩阵的依赖较高，若参数设置不当可能会显著影响效果。

6.2 思考与改进方向

1. 滤波器参数优化

传统滤波器（如 IIR 和 FIR）的设计对参数选择高度敏感，尤其是通带截止频率、阻带截止频率以及滤波器阶数直接决定了滤波性能。未来可通过频率响应分析进一步优化参数设置，或者引入遗传算法等智能优化技术提高滤波器的性能，兼顾计算复杂度和信号保真。

2. 现代滤波算法的改进

谱减法在宽带高斯噪声场景中的表现较为优秀，但在语音信号失真方面仍有提升空间。未来可考虑引入自适应参数调整机制，根据信号的动态变化实时调整噪声权重因子以优化去噪效果。此外，优化谱减法需对动态权重和平滑因子进行更细致的参数验证，以避免过度减噪或残留噪声的问题。

3. 神经网络滤波的优化与扩展

神经网络滤波（如 Wave-U-Net 架构）在处理复杂非线性噪声方面表现出显著优势，但对训练数据规模和模型计算资源要求较高。未来实验可通过引入更大规模的语音数据集，以及采用迁移学习等方法提升模型泛化能力。还可尝试将神经网络滤波与传统滤波器结合，形成混合滤波系统，兼顾实时性与去噪精度。

4. 多方法融合与性能评估

各类滤波方法具有不同的优缺点，在实际应用中可尝试结合多种方法的优势。例如，将谱减法与卡尔曼滤波结合，利用前者的频率选择性和后者的时序预测能力，进一步提升滤波性能。此外，实验评估时应采用多维度指标（如信噪比、均方误差和语音清晰度评分）进行全面比较，以便更准确地反映各方法的优劣。

5. GUI 系统设计与用户体验优化

本实验设计的 App 界面直观展示了各类滤波器的频谱响应和信号波形，但在交互性和智能化程度上仍有提升空间。未来可考虑加入自动参数调整模块，或者基于输入信号特性自动推荐滤波器设计方案，进一步提高系统的实用性和用户体验。

6.3 实验价值与启示

通过本次实验，不仅系统掌握了传统与现代滤波方法的理论与实践，还深刻认识到信号处理任务中理论与实践结合的重要性。现代滤波算法（如谱减法和神经网络滤波）在复杂噪声场景中的优越性能表明其在语音增强、通信等领域具有重要的应用前景。同时，实验也揭示了这些方法在实现复杂度和对数据依赖性上的挑战。未来研究中，可在精确性与效率之间寻求平衡，探索多种滤波方法的结合，从而构建高效、鲁棒的信号处理系统。这些启示将对后续研究和应用实践提供重要指导。

参考文献

- [1] 李军成, 杨炼, and 刘成志. *MATLAB 语言基础*. 南京大学出版社, 2022.
- [2] Christopher Lee Bennett. *Digital Audio Theory: A Practical Guide*. Taylor and Francis, 2021.
- [3] Rahul Tandra and Anant Sahai. “SNR walls for signal detection”. In: *IEEE Journal of selected topics in Signal Processing* 2.1 (2008), pp. 4–17.
- [4] 李星新 and 黄熹. *MATLAB 2020 GUI 程序设计从入门到精通*. 机械工业出版社, 2021.
- [5] Saeed V Vaseghi and Saeed V Vaseghi. “Spectral subtraction”. In: *Advanced Signal Processing and Digital Noise Reduction* (1996), pp. 242–260.
- [6] Charles K Chui, Guanrong Chen, et al. *Kalman filtering*. Springer, 2017.
- [7] Ambuj Mehrish et al. “A review of deep learning techniques for speech processing”. In: *Information Fusion* 99 (2023), p. 101869.
- [8] Ivan W Selesnick and C Sidney Burrus. “Generalized digital Butterworth filter design”. In: *IEEE Transactions on signal processing* 46.6 (1998), pp. 1688–1694.
- [9] Domenico Mirri et al. “Performance evaluation of cascaded rectangular windows in spectral analysis”. In: *Measurement* 36.1 (2004), pp. 37–52.
- [10] Alfredo Testa, Daniele Gallo, and Roberto Langella. “On the processing of harmonics and interharmonics: Using Hanning window in standard framework”. In: *IEEE Transactions on Power Delivery* 19.1 (2004), pp. 28–34.
- [11] 王浩军. “基于窗函数的数字 FIR 滤波器设计”. In: *舰船电子工程* 37.08 (2017), pp. 169–171. ISSN: 1672-9730.
- [12] Prajyo Podder et al. “Comparative performance analysis of hamming, hanning and blackman window”. In: *International Journal of Computer Applications* 96.18 (2014), pp. 1–7.
- [13] 王艳文, 崔志娟, and 张静. “基于 MATLAB 凯泽窗的 FIR 数字滤波器设计”. In: *黑龙江科技信息* 18 (2013), p. 152. ISSN: 1673-1328.
- [14] Daniel Stoller, Sebastian Ewert, and Simon Dixon. “Wave-u-net: A multi-scale neural network for end-to-end audio source separation”. In: *arXiv preprint arXiv:1806.03185* (2018).
- [15] Hendrik Schroter et al. “DeepFilterNet: A low complexity speech enhancement framework for full-band audio based on deep filtering”. In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 7407–7411.
- [16] Jean-Marc Valin. “A hybrid DSP/deep learning approach to real-time full-band speech enhancement”. In: *2018 IEEE 20th international workshop on multimedia signal processing (MMSP)*. IEEE. 2018, pp. 1–5.
- [17] Wolfgang Mack and Emanuël AP Habets. “Deep filtering: Signal extraction and reconstruction using complex time-frequency filters”. In: *IEEE Signal Processing Letters* 27 (2019), pp. 61–65.
- [18] DeLiang Wang and Jitong Chen. “Supervised speech separation based on deep learning: An overview”. In: *IEEE/ACM transactions on audio, speech, and language processing* 26.10 (2018), pp. 1702–1726.