
北京理工大學

信号与信息处理课程设计实验报告

# 信号产生、分析与处理软件系统设计

学　　院：信息与电子学院

专　　业：电子信息工程（徐特立英才班）

班　　级：[redacted]

姓　　名：[redacted]

学　　号：[redacted]

# 目录

# 实验八 心电信号心率检测方法

## 一、 总体方案设计

### 1. 需求分析

本项目面向心电信号心率监测的需求，利用 MATLAB 中 APP 设计实现心电信号频谱分析、滤波处理以及通过时域法和频域法检测心率等一系列功能。项目的全部功能均可通过图形窗口界面进行可视化交互操作，不需要再通过编写代码的方式实现。具体的需求分析阐述如下：

- 数据加载：支持加载.mat 格式的心电信号数据，自动处理数据格式并生成时间轴。
- 滤波器设计：支持高低通组合 IIR 巴特沃斯带通滤波器设计，通带、阻带截止频率和通带波纹、阻带衰减等参数可由用户设置。利用设计的滤波器对原始心电信号进行处理，实现噪声抑制。
- 心率检测：支持时域法和频域法两种心率计算方式，计算心率采用的时间窗长度和步长可由用户设置。
- 结果可视化展示：通过 GUI 界面直观展示所设计滤波器的幅频响应和相频响应、滤波前后心电信号的波形和频谱图以及时域法和频域法求得的心率变化曲线。

### 2. 设计目标

- 软件系统设计：开发一款可视化的 MATLAB App 工具，实现心电信号的频谱分析、IIR 带通滤波器设计以及时域法和频域法检测心率并进行结果对比。
- 用户友好交互：设计的心电检测系统提供友好的用户界面，降低操作复杂度，支持参数自定义配置。在心率检测运行过程提供进度条展示程序处理进度，并提供状态提示和错误告警。
- 数据处理可靠性：优化 FFT 计算点数和 R 峰检测逻辑，提升结果准确性。
- 结果图表绘制：全面展示所设计滤波器的幅频响应和相频响应、滤波前后心电信号的波形和频谱图以及时域法和频域法求得的心率变化等图表。

### 3. 设计方案

#### 3.1. 系统总体结构

本系统基于 MATLAB App Designer 开发，采用模块化分层架构，整体分为 GUI 交互层、功能模块层和数据存储层这三部分。每一层均实现了相应的系统功能，具体如下所示：

- GUI 交互层：提供用户操作入口和结果展示窗口，包括标签页、按钮、输入框、坐标轴等组件。

- 功能模块层：心电检测系统的核心算法实现层，用于封装滤波器设计、心电信号处理和心率检测等算法。
- 数据存储层：存储原始和滤波处理后的心电数据、所设计的滤波器参数，心率检测结果等，为 GUI 交互层和功能模块层提供数据支持。

### 3.2. 系统功能模块规划与核心功能实现

系统的功能模块层分为数据加载与预处理模块、信号频谱分析模块、滤波器设计模块、心率检测模块、结果可视化模块和状态与告警模块。各模块独立封装、功能明确，具体实现功能阐述如下：

#### 3.2.1. 数据加载与预处理模块

该模块负责加载用户选中的.mat 心率信号文件，并进行数据格式的校验与转换，生成心率信号对应的时间向量

#### 3.2.2. 滤波器设计模块

该模块负责根据用户输入的通带波纹、阻带衰减以及通带、阻带截止频率等参数，通过高通滤波器和低通滤波器组合设计带通滤波器，并通过结果可视化模块展示滤波器的频率响应。滤波器设计完成后，对原始心电信号进行滤波处理，为后续的心率检测做好准备。

#### 3.2.3. 信号频谱分析模块

该模块负责对原始和滤波处理后的心电信号进行 FFT 频谱分析，并通过结果可视化模块展示对应的频谱图。

#### 3.2.4. 心率检测模块

该模块根据滤波后的心电信号通过时域法和频域法计算心率，计算心率的平均值和方差，并通过结果绘制模块绘制心率变化曲线。

#### 3.2.5. 结果可视化模块

该模块负责绘制所设计滤波器的幅频响应和相频响应、滤波前后心电信号的波形和频谱图以及时域法和频域法求得的心率变化曲线。

#### 3.2.6. 状态与告警模块

该模块负责提示用户当前操作状态，展示状态灯和心率计算进度条，并对数据缺失、参数异常等状态进行错误告警，提升用户的使用体验。

### 3.3. 各模块之间的关系

各功能模块相互协作、数据互通，大致按照操作顺序递进关联。数据加载模块加载的原始心电信号作为滤波器设计模块、频谱分析模块的输入数据，分别进行滤波处理

和频谱分析。滤波器设计模块处理后的心电信号再次作为频谱分析模块的输入数据，对处理后的心电信号进行频谱分析并与处理前的心电信号进行对比。与此同时，滤波器设计模块处理后的心电信号可同时作为心率检测模块的输入信号，利用时域法和频域法检测心率。以上这些模块的数据最后又会输入到结果可视化模块，信号波形、频谱、心率结果等均通过可视化模块展示。

**4. 开发环境**

- 操作系统：Windows11 64 位操作系统。
- 开发软件：MATLAB R2025b。
- MATLAB 工具包：App Designer（用于 GUI 设计与代码生成），Signal Processing Toolbox（用于信号生成、频谱分析、滤波器设计的核心函数，如 fft、butter、freqz 等），MATLAB Base Toolbox（用于数据处理、图形绘制、字符串操作等基础功能）。
- 硬件配置：CPU Intel Core i7，内存 16GB，硬盘剩余空间 350GB，满足 MATLAB 软件运行与数据处理的硬件需求。
- 开发文档：MATLAB 官方文档，数字信号处理、信号与系统等信号处理教材。

## 二、 系统的设计与实现

### 1. 软件系统界面设计

本软件系统基于 MATLAB App 设计工具箱开发，整体界面简洁直观，操作流程清晰，可以完成基本的心电信号心率检测的功能。该软件系统包含三个标签页，分别完成滤波器设计、心电信号滤波处理和心率检测的功能，下面对两个标签页对应的界面分别进行介绍：

#### 1.1. 标签页 1：滤波器设计

标签页 1 实现滤波器的设计并绘制滤波器频率响应的功能，布局主要分为上侧参数控制区和下侧绘图展示区两大区域，如图 1 所示：

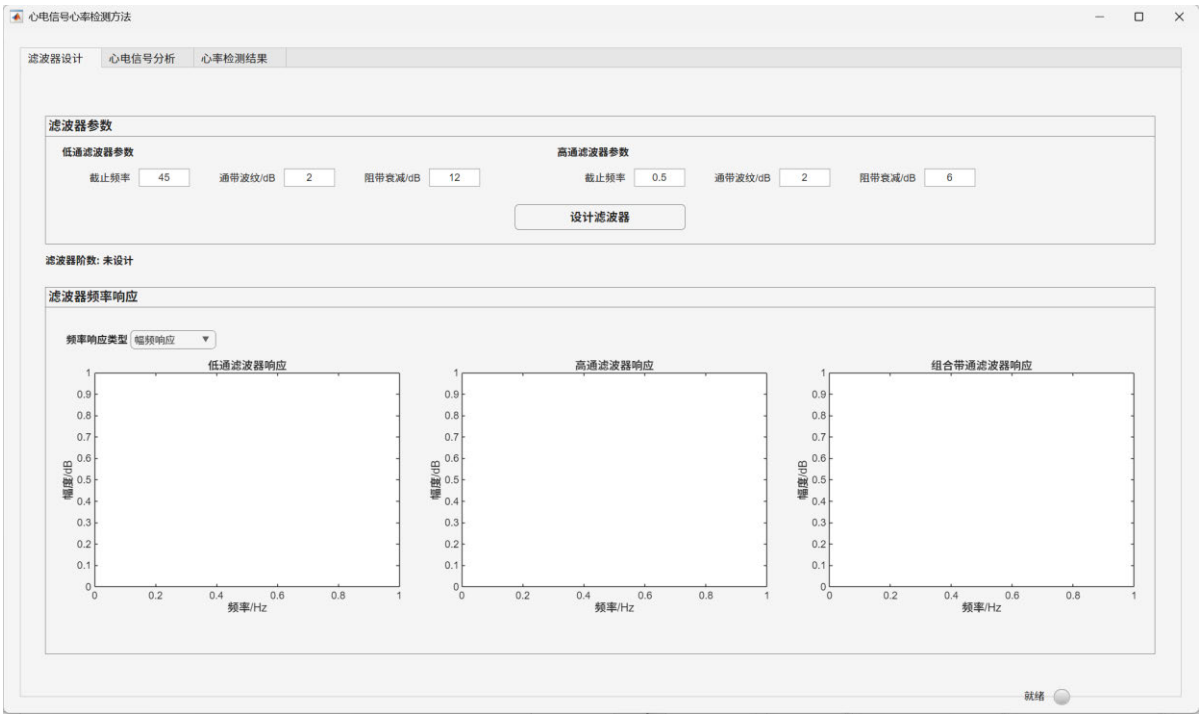


图 1：标签页 1 初始化界面

上图所示界面为直接打开系统，未经任何操作的初始化界面。

#### 1.1.1. 上侧参数控制区

标签页 1 的上侧为滤波器参数面板。滤波器参数面板包含所设计的高通滤波器和低通滤波器的截止频率、通带波纹和阻带衰减等指标，默认输入参数如上图所示，实验中保持默认参数即可。完成滤波器参数的输入后，点击“设计滤波器”按钮即可按照输入的指标进行滤波器的设计，并在下侧绘图展示区绘制滤波器的频率响应。

#### 1.1.2. 下侧绘图展示区

标签页 1 的下侧为滤波器频率响应面板，面板中包含一个“频率响应”下拉框和三个水平排列的坐标区，这三个坐标区分别绘制所设计的低通滤波器、高通滤波器和组合的带通滤波器的频率响应。通过切换“频率响应类型”下拉框，可选择绘制所设计滤波器的幅频响应和相频相应。

### 1.2. 标签页 2：心电信号分析

标签页 2 实现心电信号的滤波处理和频谱分析，并绘制原始心电信号和滤波处理后的心电信号的时域波形和频谱图，布局分为上侧数据加载区和下侧绘图展示区两大区域，如图 2 所示：

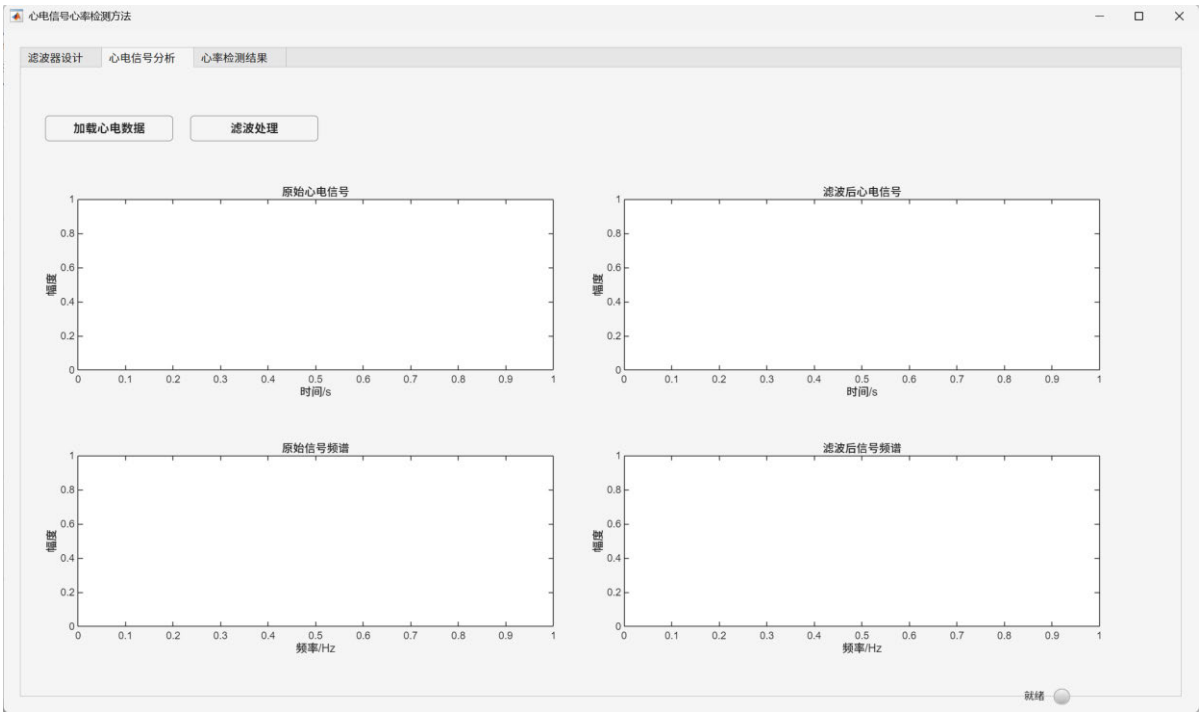


图 2：标签页 2 初始化界面

上图所示界面为直接打开系统，未经任何操作的初始化界面。

#### 1.2.1. 上侧数据加载区

标签页 2 的最上侧为“加载心电数据”和“滤波处理”两个按钮，点击“加载心电数据”按钮可选择存储心电信号的.mat 文件进行加载，若加载失败则会弹窗提示错误原因。加载数据完成后即可点击“滤波处理”按钮对加载的原始心电数据进行滤波处理。

#### 1.2.2. 下侧绘图展示区

标签页 2 下侧包含 4 个 2×2 排列的坐标区。其中左上方和和左下方的坐标区用于绘制原始心电信号的时域波形和频谱，在加载心电信号后自动绘制。右上方和右下方的坐标区用于绘制滤波处理后心电信号的时域波形和频谱，在滤波处理后自动绘制。

### 1.3. 标签页 3：心率检测结果

标签页 3 实现利用时域法和频域法对滤波处理后的心电信号进行心率检测，并绘制两种检测方法的心率变化曲线、给出心率的平均值和方差，布局分为上侧参数控制区和下侧绘图展示区，如图 3 所示：

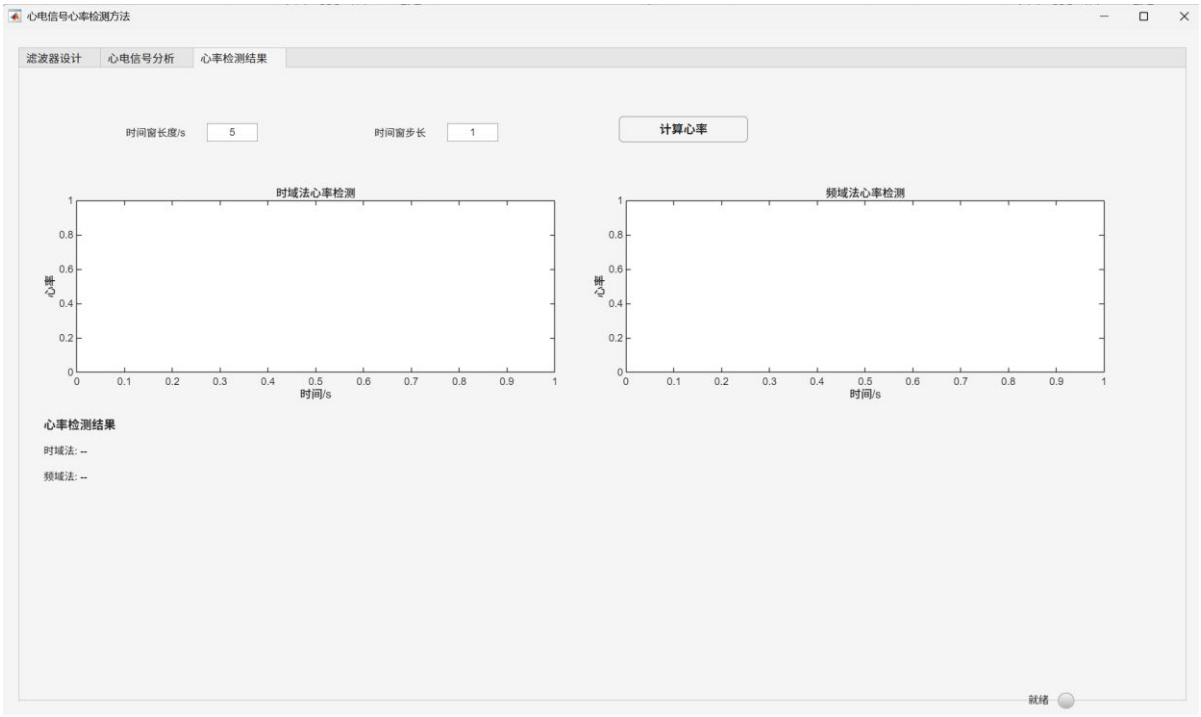


图 3：标签页 3 初始化界面

上图所示界面为直接打开系统，未经任何操作的初始化界面。

#### 1.3.1. 上侧参数控制区

标签页 3 的最上侧为心率检测的时间窗长度和时间窗步长编辑字段，默认输入参数如上图所示，与题目要求相同。编辑字段的右侧为“计算心率”按钮，点击后可自动时域法和频域法的心率检测。

#### 1.3.2. 下侧绘图展示区

标签页 3 下侧包含 2 个 水平排列的坐标区。其中左侧坐标区用于绘制时域法检测的心率变化曲线，右侧坐标区用于绘制频域法检测的心率变化，在加载心电信号后自动绘制。与此同时在这两个坐标区的下方还会展示时域法和频域法心率检测结果的平均值和方差，以供用户参考对比。

## 2. 核心模块的具体实现方法

### 2.1. 滤波器设计模块

#### 2.1.1. 基本原理

要实现 0.5~45Hz 的带通滤波，首先要设计一个对应截止频率的带通滤波器。为了提高滤波器设计的灵活性，并限制滤波器的阶数不要太高，本实验采取低通滤波器和高通滤波器组合的方式设计带通滤波器。为设计方便起见，并保证在通带范围内滤波器的频率响应尽可能平坦，本实验将采用巴特沃斯滤波器进行 IIR 滤波器设计。

N 阶巴特沃斯低通滤波器的幅度平方响应为：

$$|H_a(j\omega)|^2 = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2N}} \tag{1}$$

其中，$N$为滤波器阶数，$\omega_c$是 3dB 截止频率（单位：$rad/s$）。设定滤波器的截止频率、过渡带宽、通带波纹和阻带衰减等参数，根据公式(1)可计算出模拟巴特沃斯滤波器的最小阶数 N 和 3dB 截止频率 $\omega_c$，据此可求出滤波器的系统函数$H_a(s)$。

对于 IIR 滤波器，可采用双线性变换法进行设计。根据实验要求可设定低通滤波器的通带截止频率$\omega_s$为 45Hz，过渡带宽为 5Hz（即阻带截止频率$\omega_p$为 50Hz）；高通滤波器的通带截止频率$\omega_s$为 0.5Hz，过渡带宽为 0.1Hz（即阻带截止频率$\omega_p$为 0.4Hz），但上述的截止频率不能直接用于巴特沃斯模拟滤波器的设计。首先将实验中设定的模拟截止频率$\omega_p$、$\omega_s$转换为数字频率$\Omega_p$、$\Omega_s$，然后按照以下公式进行频率预畸：

$$\omega = \frac{2}{T}tan\left(\frac{\Omega}{2}\right) \tag{2}$$

由此得到模拟原型滤波器的截止频率$\omega_p'$、$\omega_s'$。根据预畸后的截止频率$\omega_p'$和$\omega_s'$、通带波纹和阻带衰减等参数按照上述的方法设计巴特沃斯模拟滤波器$H_a(s)$，再根据双线性变换法的映射关系设计数字滤波器$H(z)$，即为

$$H(z) = H_a\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) \tag{3}$$

按照双线性变换法可分别设计低通滤波器$H_1(z)$和$H_2(z)$，则组合后的带通滤波器的系统函数为：

$$H(z) = H_1(z)H_2(z) \tag{4}$$

滤波器设计完成后，需要对待处理的心电信号进行滤波处理，消除低频噪声和高频噪声对心电信号心率检测的影响。设输入的单试次心电信号为$x[n]$，滤波处理后的信号为输入信号与滤波器冲激响应的线性卷积：

$$y[n] = x[n] * h[n] \tag{5}$$

因此有

$$y[n] = -\sum_{k=1}^{M} a_k y[n-k] + \sum_{k=0}^{N} b_k x[n-k] \tag{6}$$

其中$a_k$和$b_k$分别为滤波器系统函数$H(z)$分母和分子的系数，$y[n]$即为滤波处理后的脑电信号。

### 2.1.2. 关键 MATLAB 函数

- buttord：计算滤波器阶数 N 和截止频率 Wn；

- butter：生成 IIR 巴特沃斯滤波器系数；
- freqz：计算频率响应；
- filter：应用滤波器对信号滤波；

### 2.1.3. 核心代码

信号产生模块的核心代码为“设计滤波器”按钮的回调函数和 applyCascadeFilter 函数，如下所示：

```
% Button pushed function: DesignFilterButton
function DesignFilterButtonPushed(app, event)
    % 设计巴特沃斯 IIR 滤波器
    try
        app.StatusLabel.Text = '正在设计滤波器...';
        app.ProgressLamp.Color = 'yellow';
        drawnow;

        % 获取滤波器参数
        app.LowpassCutoff = app.LowpassCutoffEditField.Value;
        app.LowpassPassRipple = app.LowpassPassRippleEditField.Value;
        app.LowpassStopAtten = app.LowpassStopAttenEditField.Value;

        app.HighpassCutoff = app.HighpassCutoffEditField.Value;
        app.HighpassPassRipple = app.HighpassPassRippleEditField.Value;
        app.HighpassStopAtten = app.HighpassStopAttenEditField.Value;

        % 归一化频率
        nyquist = app.Fs / 2;

        % 低通滤波器设计参数
        lpWp = app.LowpassCutoff / nyquist;       % 通带截止频率
        lpWs = (app.LowpassCutoff + 5) / nyquist; % 阻带截止频率（比通带高 5Hz）

        % 高通滤波器设计参数
        hpWp = app.HighpassCutoff / nyquist;      % 通带截止频率
        hpWs = (app.HighpassCutoff - 0.1) / nyquist; % 阻带截止频率
        % 设计低通滤波器
        [lpN, lpWn] = buttord(lpWp, lpWs, app.LowpassPassRipple, app.LowpassStopA
    tten);
        [lpB, lpA] = butter(lpN, lpWn, 'low');

        % 设计高通滤波器
        [hpN, hpWn] = buttord(hpWp, hpWs, app.HighpassPassRipple, app.HighpassSto
    pAtten);
        [hpB, hpA] = butter(hpN, hpWn, 'high');
```

```
35.
36.         % 保存滤波器系数
37.         app.lpCoeff.b = lpB;
38.         app.lpCoeff.a = lpA;
39.         app.lpCoeff.order = lpN;
40.         app.lpCoeff.passRipple = app.LowpassPassRipple;
41.         app.lpCoeff.stopAtten = app.LowpassStopAtten;
42.
43.         app.hpCoeff.b = hpB;
44.         app.hpCoeff.a = hpA;
45.         app.hpCoeff.order = hpN;
46.         app.hpCoeff.passRipple = app.HighpassPassRipple;
47.         app.hpCoeff.stopAtten = app.HighpassStopAtten;
48.
49.         % 计算频率响应
50.         N = 32768;
51.         [app.lpH, app.combinedF] = freqz(lpB, lpA, N, app.Fs);
52.         [app.hpH, ~] = freqz(hpB, hpA, N, app.Fs);
53.
54.         % 计算组合响应
55.         app.combinedH = app.lpH .* app.hpH;
56.         % 绘制频率响应
57.         switch app.FreqzDropDown.Value
58.             case '幅频响应'
59.                 PlotAmpFreqz(app)
60.             case '相频响应'
61.                 PlotPhasedFreqz(app)
62.             otherwise
63.                 PlotAmpFreqz(app)
64.         end
65.
66.         % 显示滤波器阶数信息
67.         app.FilterOrderInfoLabel.Text = sprintf('滤波器阶数 - 低通: %d ,高
    通: %d', lpN ,hpN);
68.         app.StatusLabel.Text = '滤波器设计完成';
69.         app.ProgressLamp.Color = 'green';
70.
71.     catch ME
72.         uialert(app.UIFigure, ME.message, '滤波器设计错误');
73.         app.StatusLabel.Text = '滤波器设计失败';
74.         app.ProgressLamp.Color = 'red';
75.     end
76. end
77.
78. function filterSignal = applyCascadeFilter(app, signal)
```

```
    % 应用级联滤波器
    lpFiltered = filter(app.hpCoeff.b, app.hpCoeff.a, signal);
    filterSignal = filter(app.lpCoeff.b, app.lpCoeff.a, lpFiltered);
end
```

## 2.2. 信号频谱分析模块

### 2.2.1. 基本原理

信号频谱分析模块用于分析原始心电信号和滤波处理后的心电信号的频谱特性。首先按照采样频率 fs 对心电信号进行采样，得到采样序列$x[n]$，对其进行 FFT，即为

$$X[k] = FFT(x[n]) \tag{7}$$

其中$Y[k]$为采样信号的 L 点 FFT，方便计算，通常取 FFT 点数 L 为大于信号采样点数的最小一个 2 的幂次。由于 FFT 仅是 DFT 的快速计算方法，根据 DFT 和 FT 之间的关系，当 k 取值在$0\sim\frac{L}{2}$时对应的模拟频率为$0\sim\frac{f_s}{2}$，则 FFT 序列第 k 个值对应的模拟频率应当为$\frac{kf_s}{L}$。记$X(j\omega)$为对应的连续频谱，则存在下述对应关系：

$$X\left(j\frac{kf_s}{L}\right) = \frac{X[k]}{f_s} \tag{8}$$

对于单边谱密度，还需对除 0 和奈奎斯特频率$\frac{f_s}{2}$的频谱幅度值均乘以 2 进行调整。由公式(8)可以看出，通过序列的 FFT 可以对连续信号的 FT 进行近似估计。

### 2.2.2. 关键 MATLAB 函数

- fft：快速傅里叶变换，将时域信号转换为频域；
- abs：计算信号频谱的模值；
- nextpow2：计算大于等于信号长度的最小 2 的幂，优化 FFT 运算效率；

### 2.2.3. 核心代码

频谱分析模块的主要代码为函数 analyzeECGSpectrum，如下所示：

```
function analyzeECGSpectrum(app)
    % 分析心电信号频谱
    app.StatusLabel.Text = '正在分析心电信号频谱...';
    drawnow;

    % 计算原始信号频谱
    N_raw = length(app.rawECG);

    if N_raw > 0
        % 使用大于等于序列长度的最小 2 的幂次作为 FFT 点数
        nfft_raw = 2^nextpow2(N_raw);

        % 计算 FFT
```

```
        Y_raw = fft(app.rawECG, nfft_raw);
        P2_raw = abs(Y_raw/app.Fs);
        P1_raw = P2_raw(1:nfft_raw/2+1);
        P1_raw(2:end-1) = 2*P1_raw(2:end-1);
        app.rawSpectrum = P1_raw;

        % 生成频率向量
        app.freqVector = app.Fs*(0:(nfft_raw/2))/nfft_raw;
    end

    % 计算滤波后信号频谱
    if ~isempty(app.filteredECG)
        N_filt = length(app.filteredECG);

        % 使用大于等于序列长度的最小 2 的幂次作为 FFT 点数
        nfft_filt = 2^nextpow2(N_filt);

        % 确保滤波后信号的 FFT 点数与原始信号一致，以便使用相同的频率向量
        if nfft_filt ~= nfft_raw
            % 如果长度不同，使用原始信号的 FFT 点数来保持一致
            nfft_filt = nfft_raw;
        end

        % 计算 FFT
        Y_filt = fft(app.filteredECG, nfft_filt);
        P2_filt = abs(Y_filt/app.Fs);
        P1_filt = P2_filt(1:nfft_filt/2+1);
        P1_filt(2:end-1) = 2*P1_filt(2:end-1);
        app.filteredSpectrum = P1_filt;
    end
end
```

## 2.3. 心率检测模块

### 2.3.1. 基本原理

心电信号是心脏心肌细胞电活动的综合反映，一般情况下波形如图 4 所示：

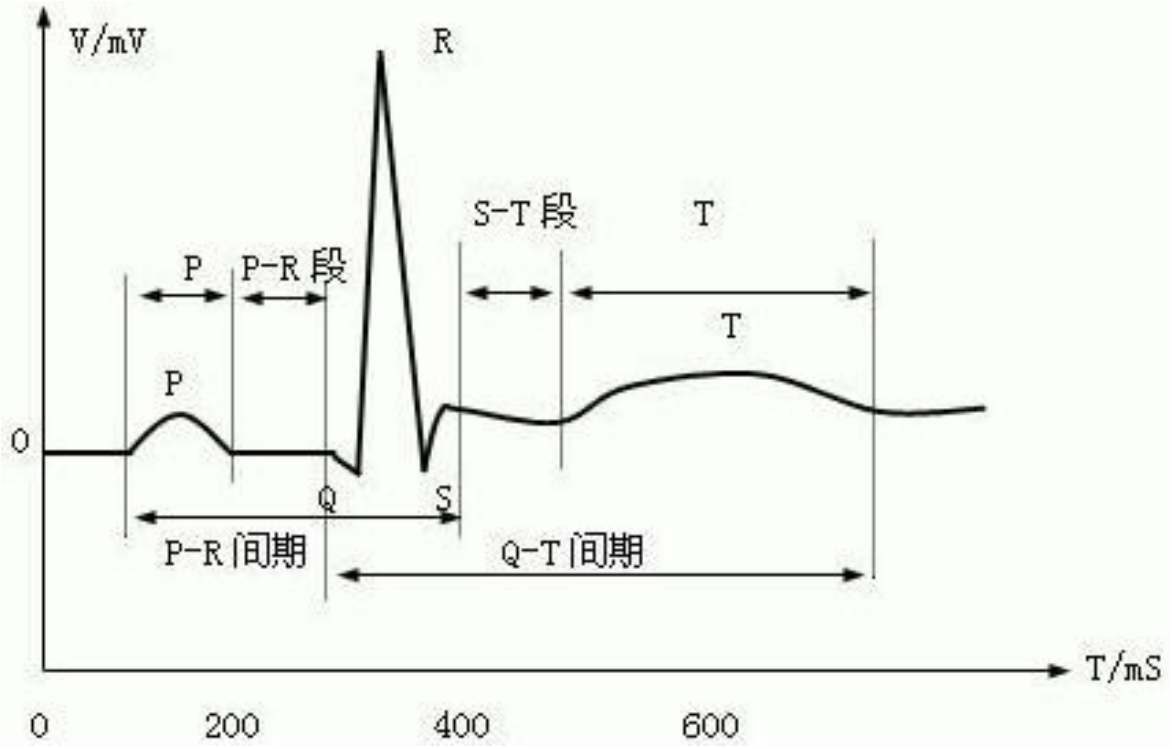


图 4：心电信号波形

心率指正常人安静状态下每分钟心跳的次数，也叫安静心率，一般为 60～100 次/分，可由心电信号 R 峰出现的频率计算心率，心率估计的算法分为时域法和频域法两种。

时域法基于心电信号的 R 峰检测，实验中采用 5 秒长度的滑动时间窗，寻找窗内 R 峰所在的时间点，并设置峰值高度和间隔阈值避免误检。在实验讲义中是通过计算时间窗内 R 峰的个数进而估计每分钟的心率，但这种方法忽略了时间窗内第一个和最后一个 R 峰距离时间窗起始时刻和终止时刻的距离。并且由于时间窗的长度有限，当时间窗内的 R 峰数量发生变化，计算的每分钟心率数值也会发生较大的变化，导致心率变化曲线出现较大的突变，这显然与实际情况不符。在这里对时域法的心率检测进行了一个改进：通过计算时间窗内每两个相邻 R 峰之间的平均时间间隔，即一个心动周期的时间。用 60 秒除以一个心动周期的时间，即可得到每分钟的心率。

频域法基于心电信号的频谱分析，实验中采用 5 秒长度的滑动时间窗，对窗内的心电信号做 FFT，提取 0.5~2.5Hz 频段内的最大峰值频率，将该峰值频率乘 60 即可得到每分钟的平均心率。注意为了减小心率变化曲线发生的波动，FFT 分析选取的频率点要足够密集。因此在对时间窗内的心电信号做 FFT 分析时要对序列进行补零增加 FFT 点数，以减少栅栏效应的影响，从而使心率变化曲线更加平滑。

### 2.3.2. 关键 MATLAB 函数

- findpeaks 检测 R 峰所在的频率；
- diff：计算相邻 R 峰位置间隔；
- mean：计算相邻 R 峰间隔的平均值；
- fft：快速傅里叶变换，计算时间窗信号的频谱；
- max：寻找 0.5~2.5Hz 频段的峰值频率；

### 2.3.3. 核心代码

滤波器设计的核心代码为函数 detectHRTimeDomain 和 detectHRFreqDomain，如下所示：

```
function detectHRTimeDomain(app, ecgSignal, progressDialog, targetProgress)
    % 时域法心率检测
    % 输入：滤波后的心电信号
    % 输出：心率随时间变化数组

    N = length(ecgSignal);
    windowLength = round(app.WindowTimeEditField.Value * app.Fs);
    stepSize = app.WindowStepEditField.Value;

    % 计算滑动窗口数量
    numWindows = floor((N - windowLength) / stepSize) + 1;

    % 初始化结果数组
    app.hrTimeDomain = zeros(numWindows, 1);

    % 计算每个窗口的进度增量
    progressIncrement = targetProgress / numWindows;

    for i = 1:numWindows
        % 计算窗口起始和结束索引
        startIdx = (i-1) * stepSize + 1;
        endIdx = startIdx + windowLength - 1;

        % 提取当前窗口数据
        windowData = ecgSignal(startIdx:endIdx);

        % 检测 R 峰
        [~, peakLocs] = findpeaks(windowData, 'MinPeakHeight', 0.5*max(windowData
), ...
            'MinPeakDistance', 0.3*app.Fs); % R 峰最小间隔 0.3 秒

        % 计算心率（次/分钟）
```

```
        if length(peakLocs) >= 2
            rrIntervals = diff(peakLocs) / app.Fs; % R-R 间隔（秒）
            app.hrTimeDomain(i) = 60 / mean(rrIntervals);
        else
            app.hrTimeDomain(i) = 0;
        end

        % 每 10 个窗口或最后一个窗口更新一次进度条
        if mod(i, 10) == 0 || i == numWindows
            currentProgress = i * progressIncrement;
            progressDialog.Value = currentProgress;
            progressDialog.Message = sprintf('时域法计算进度: %d/%d (%.1f%%)', ...
                i, numWindows, currentProgress*100);
            drawnow;
        end
    end
    % 移除无效值
    app.hrTimeDomain = app.hrTimeDomain(app.hrTimeDomain > 0);
    app.hrTimeDomain = app.hrTimeDomain;


    % 显示心率检测结果
    app.StatusLabel.Text = '正在显示时域法心率检测结果...';
    drawnow;

    % 检查是否有有效的心率数据
    if isempty(app.hrTimeDomain)
        app.ResultsSummaryLabel.Text = '请先计算心率';
        return;
    end

    % 创建时间轴（假设从第 2.5 秒开始，每个窗口中心时间）
    timeStart = app.WindowTimeEditField.Value/2;
    timeStep = app.WindowStepEditField.Value/app.Fs;
    timeAxis = timeStart:timeStep:(timeStart + (length(app.hrTimeDomain)-1)*timeStep);

    % 确保时间轴长度匹配
    minLength = min(length(timeAxis), length(app.hrTimeDomain));
    timeAxis = timeAxis(1:minLength);

    % 绘制时域法心率曲线
    plot(app.TimeDomainHRAxes, timeAxis, app.hrTimeDomain, 'b-', 'LineWidth', 2);
```

```
74.      app.TimeDomainHRAxes.Title.String = '时域法心率检测结果';
75.      app.TimeDomainHRAxes.XLabel.String = '时间/秒';
76.      app.TimeDomainHRAxes.YLabel.String = '心率/BPM';
77.      app.TimeDomainHRAxes.XGrid = 'on';
78.      app.TimeDomainHRAxes.YGrid = 'on';
79.      xlim(app.TimeDomainHRAxes, [0 240]);
80.      ylim(app.TimeDomainHRAxes, [40 160]);
81.      xticks(app.TimeDomainHRAxes, 0:40:240);
82.      xticklabels(app.TimeDomainHRAxes, 0:40:240);
83.
84.
85.      % 计算时域法统计信息
86.      avgTime = mean(app.hrTimeDomain);
87.      stdTime = std(app.hrTimeDomain);
88.      app.TimeDomainResultsLabel.Text = sprintf('平均心
     率: %.1f ± %.1f BPM', avgTime, stdTime);
89.  end
90.
91.  function detectHRFreqDomain(app, ecgSignal, progressDialog, baseProgress, targetP
     rogress)
92.      % 频域法心率检测
93.      % 输入：滤波后的心电信号
94.      % 输出：心率随时间变化数组
95.
96.      N = length(ecgSignal);
97.      windowLength = round(app.WindowTimeEditField.Value * app.Fs);
98.      stepSize = app.WindowStepEditField.Value;
99.
100.     % 计算滑动窗口数量
101.     numWindows = floor((N - windowLength) / stepSize) + 1;
102.
103.     % 初始化结果数组
104.     app.hrFreqDomain = zeros(numWindows, 1);
105.
106.     % 确定 FFT 点数：使用大于等于窗口长度的最小 2 的幂次
107.     nfft = 2^(nextpow2(windowLength) + 2);
108.
109.     % 计算每个窗口的进度增量（从 0.5 开始到 1.0）
110.     progressRange = targetProgress - baseProgress;
111.     progressIncrement = progressRange / numWindows;
112.
113.     for i = 1:numWindows
114.         % 计算窗口起始和结束索引
115.         startIdx = (i-1) * stepSize + 1;
116.         endIdx = startIdx + windowLength - 1;
```

```
117.
118.         % 提取当前窗口数据
119.         windowData = ecgSignal(startIdx:endIdx);
120.
121.         % 计算频谱，使用确定的 FFT 点数
122.         Y = fft(windowData, nfft);
123.         P2 = abs(Y/app.Fs);
124.         P1 = P2(1:nfft/2+1);
125.         P1(2:end-1) = 2*P1(2:end-1);
126.
127.         % 生成频率向量
128.         f = app.Fs*(0:(nfft/2))/nfft;
129.
130.         % 寻找 0.5-2.5Hz 范围内的最大峰值（对应 30-150BPM）
131.         freqRange = (f >= 0.5) & (f <= 2.5);
132.         if any(freqRange)
133.             [~, idx] = max(P1(freqRange));
134.             freqValues = f(freqRange);
135.             dominantFreq = freqValues(idx);
136.             app.hrFreqDomain(i) = dominantFreq * 60; % 转换为 BPM
137.         else
138.             app.hrFreqDomain(i) = 0;
139.         end
140.
141.         % 每 10 个窗口或最后一个窗口更新一次进度条
142.         if mod(i, 10) == 0 || i == numWindows
143.             currentProgress = baseProgress + i * progressIncrement;
144.             progressDialog.Value = currentProgress;
145.             progressDialog.Message = sprintf('频域法计算进
    度: %d/%d (%.1f%%)', ...
146.                 i, numWindows, currentProgress*100);
147.             drawnow;
148.         end
149.     end
150.     % 移除无效值
151.     app.hrFreqDomain = app.hrFreqDomain(app.hrFreqDomain > 0);
152.     app.hrFreqDomain = app.hrFreqDomain;
153.
154.
155.     % 显示心率检测结果
156.     app.StatusLabel.Text = '正在显示频域法心率检测结果...';
157.     drawnow;
158.
159.     % 检查是否有有效的心率数据
160.     if isempty(app.hrFreqDomain)
```

```
        app.ResultsSummaryLabel.Text = '请先计算心率';
        return;
    end

    % 创建时间轴（假设从第 2.5 秒开始，每个窗口中心时间）
    timeStart = app.WindowTimeEditField.Value/2;
    timeStep = app.WindowStepEditField.Value/app.Fs;
    timeAxis = timeStart:timeStep:(timeStart + (length(app.hrTimeDomain)-1)*timeStep);

    % 确保时间轴长度匹配
    minLength = min(length(timeAxis), length(app.hrTimeDomain));
    timeAxis = timeAxis(1:minLength);
    hrFreq = app.hrFreqDomain(1:minLength);

    % 绘制频域法心率曲线
    plot(app.FreqDomainHRAxes, timeAxis, hrFreq, 'r-', 'LineWidth', 2);
    app.FreqDomainHRAxes.Title.String = '频域法心率检测结果';
    app.FreqDomainHRAxes.XLabel.String = '时间/秒';
    app.FreqDomainHRAxes.YLabel.String = '心率/BPM';
    app.FreqDomainHRAxes.XGrid = 'on';
    app.FreqDomainHRAxes.YGrid = 'on';
    xticks(app.FreqDomainHRAxes, 0:40:240);
    xticklabels(app.FreqDomainHRAxes, 0:40:240);
    xlim(app.FreqDomainHRAxes, [0 240]);
    ylim(app.FreqDomainHRAxes, [40 160]);

    % 计算频域法统计信息
    avgFreq = mean(hrFreq);
    stdFreq = std(hrFreq);
    app.FreqDomainResultsLabel.Text = sprintf('平均心率: %.1f ± %.1f BPM', avgFreq, stdFreq);
end
```

# 三、 系统功能测试

为了验证上述设计的心率检测系统是否可以正常工作，需要对其进行功能测试。在这里分别测试了心率检测系统滤波器设计、心电信号的滤波处理和频谱分析、时域法和频域法检测心率等功能，验证系统可以正常工作。

## 1. 滤波器设计

按照题目要求进行滤波器设计，滤波器的幅频响应和相频响应分别如图 5、图 6 所示：

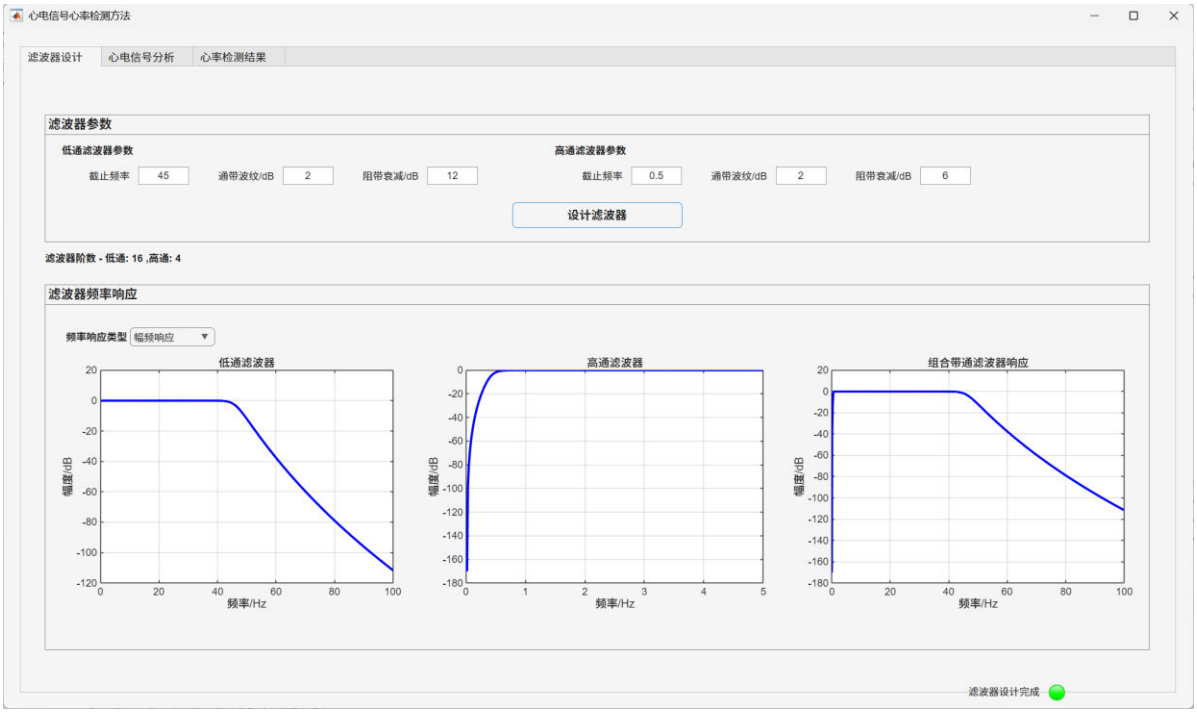


图 5：滤波器幅频响应

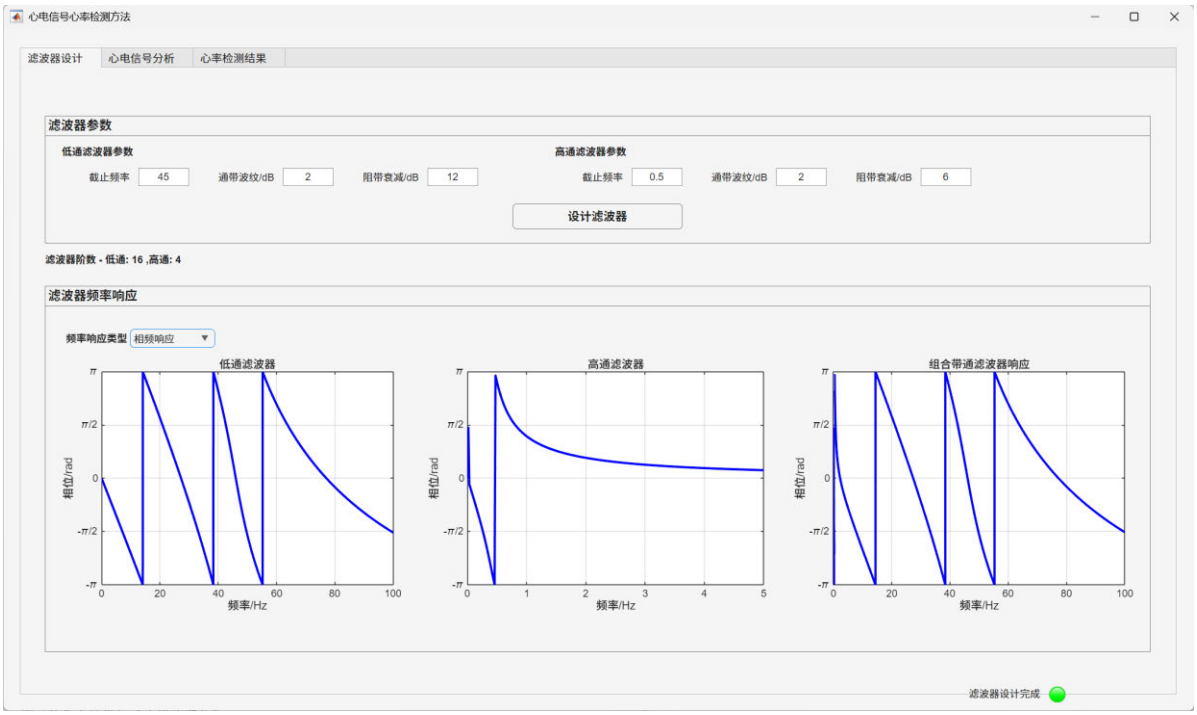


图 6：滤波器相频响应

通过观察滤波器的频率响应曲线，可发现低通滤波器和高通滤波器均有较为平坦的通带，并且截止频率符合设计参数要求。将低通滤波器和高通滤波器组合在一起，即可得到对应的带通滤波器。在本次实验中，由于高通滤波器的截止频率很低，要求过渡带必须充分窄，这在一定程度上会给滤波器的设计带来难度。通过低通和高通滤波器组合构成带通滤波器的设计方法可以很好地满足实验需求，不仅可控制滤波器的阶数不会太大，更可以提高滤波器设计的灵活性，与直接设计带通滤波器相比具有明显的

优势。从所设计滤波器的幅频响应曲线可以看出，所设计的巴特沃斯滤波器具有非常平坦的通带，这样使得通带内的频率响应不会出现很大的起伏，并且在阻带范围内仍具有较大的衰减，可以很好地滤除频带外噪声对心电信号心率检测的影响。

**2. 心电信号的滤波处理和频谱分析**

完成滤波器设计后，在心电信号分析标签页加载心电数据并进行滤波处理，得到原始心电信号的时域波形和频谱图、滤波处理后的心电信号的时域波形和频谱图如图 7 所示：

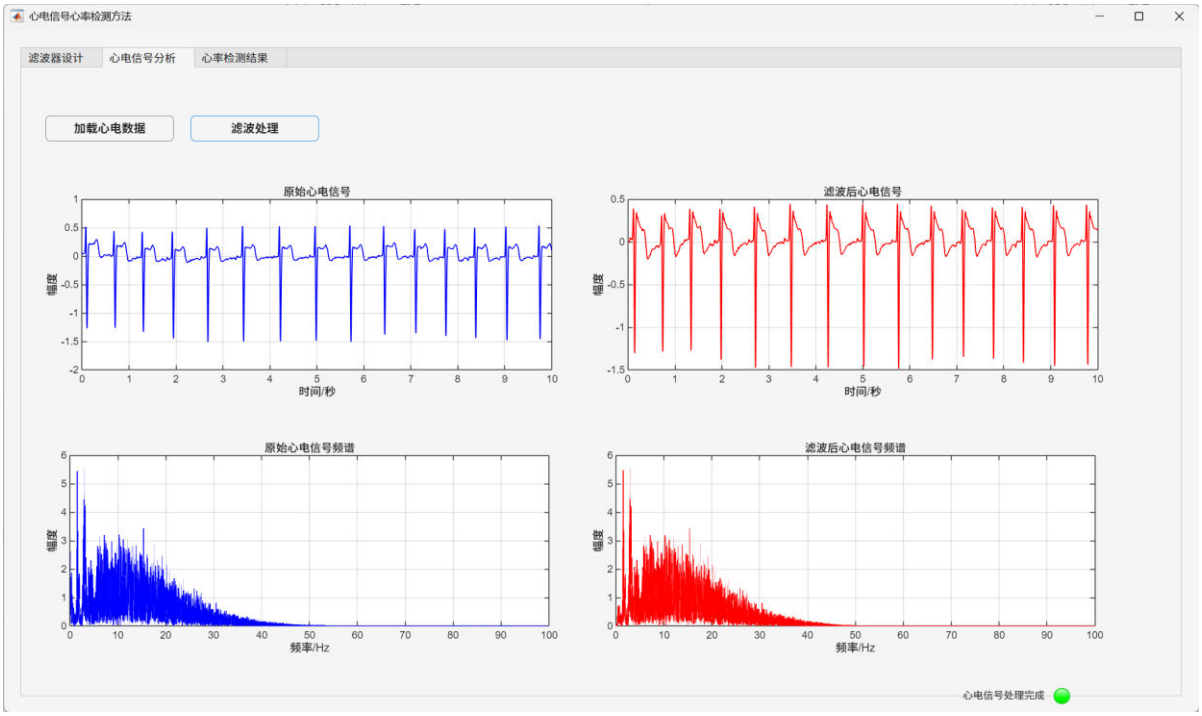


图 7：心电信号的时域波形和频谱图

为清晰展现心电信号的波形特征，心电信号的时域波形图均取前 10s 的波形，从图中可以看到心电信号的 R 峰周期性地重复出现，这与实际情况相符。观察上图中原始心电信号和滤波处理后的心电信号的频谱图可知，通带内（0.5Hz~45Hz）心电信号的频谱在滤波前后几乎一致，而当频率小于 5Hz 或大于 45Hz，滤波处理后心电信号的频谱被显著抑制，这进一步验证了带通滤波器设计的正确性。对比观察原始心电信号和滤波处理后的心电信号的时域波形可知，二者的形状差别不大，这是因为心电信号绝大多数的频率分量均位于通带内，滤波处理对心电信号影响不大。

**3. 心率检测**

设置时间窗长度为 5s，时间窗步长为 1，时域法和频域法心率检测的结果如图 8 所示：

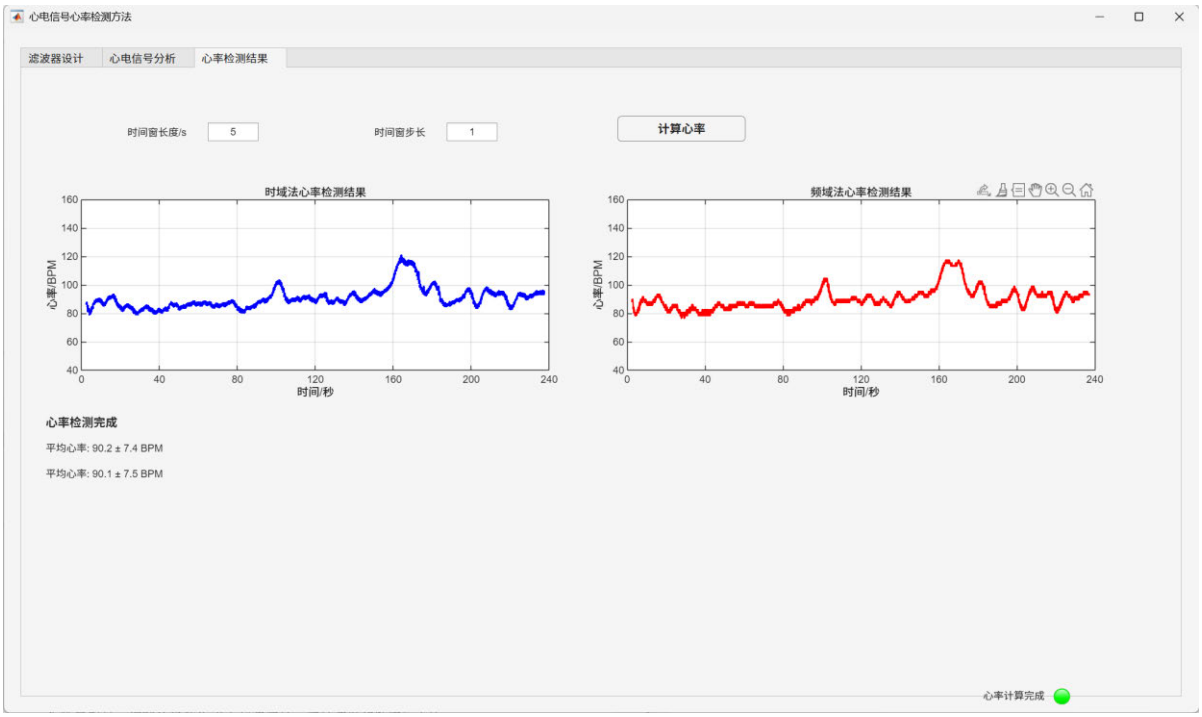


图 8：心率检测结果

对比观察时域法和频域法的心率检测结果，可以发现这两种方法计算的心率基本相同，两条心率变化曲线的形状和趋势基本一致，并且计算得到的平均心率和方差也基本相同，因此时域法和频域法相互印证可初步验证心率检测结果的正确性。与此同时可以发现时域法和频域法得到的心率变化曲线均比较平稳，曲线不会随时间变化出现较大波动，这与实际心率的特点相符。注意到时域法和频域法计算的心率在 160~170s 时明显超过了 100，这与人体正常心率 60~100 有所不符。为了验证是否是原始心电数据存在异常，现画出 162~167s 心电信号时域波形图如图 9 所示：

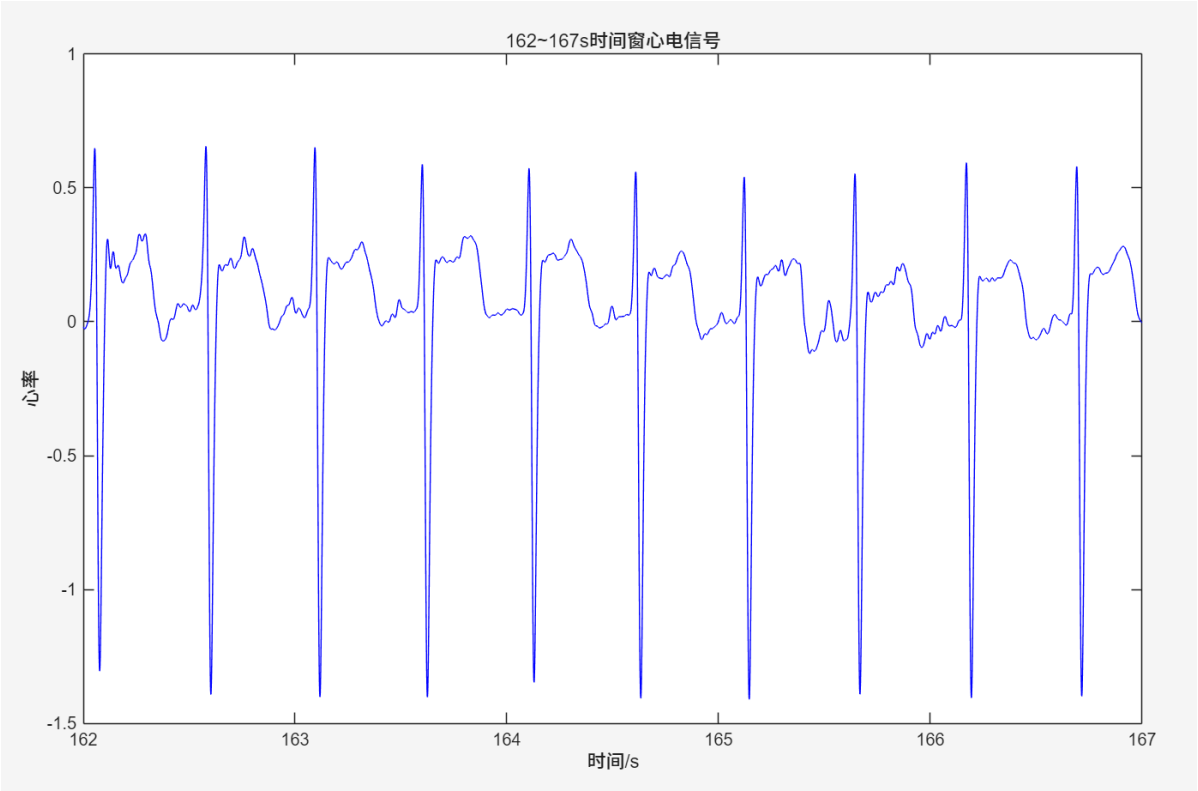


图 9：162~167s 时间窗心电信号

观察上图可以看出，在这个时间窗内心电信号出现了 10 个 R 峰，相邻 R 峰的平均时间间隔为 0.5152s，时域法计算得到的平均心率为 116.4546，这一数值和图 8 计算结果的异常段相符。因此图 8 计算得到的异常结果并不是算法设计出现了错误，而是这一段心电信号数据本身存在异常。这一异常可能是由于仪器测量误差，更可能是因为被试者出现了异常的生理状况，通过这段心电数据可对被试者的身体状况进行一个简要地评估。

## 四、 总结与体会

通过本次心电信号心率检测实验，我收获了扎实的数字信号处理知识与 MATLAB APP 开发实践经验，对心电信号处理的核心原理与工程实现方法有了更为透彻的理解。实验过程中，我不仅深入掌握了利用心电信号 R 峰进行心率检测的核心原理，还熟练运用 MATLAB 实现了 IIR 巴特沃斯带通滤波设计、滑动时间窗 R 峰检测、FFT 频谱分析等核心算法，成功开发出实现心电信号滤波处理和心率检测的模块化 GUI 系统。

其中，实验中我遇到的最大问题就是频域法心率检测时心率变化曲线会出现较大的突变。正常生理状态下心率随时间应当是平稳变化，不应当出现明显的突变，而我在开发初期绘制的心率变化曲线则会随着时间变化出现明显的跳变，这显然是不合理的。为解决这一问题，我反复调试代码，并复习信号处理的相关知识，最终发现引起频域法检测结果的心率变化曲线出现明显跳变的原因是 FFT 频谱分析中存在的栅栏效应。当

FFT 点数不够多时，由于栅栏效应的影响使得相邻频率分量的间距较大，如果检测到的峰值频率出现偏差，再乘以 60 得到的心率检测结果就会出现较大的突变。为了使心率变化曲线更加平滑，必须减小栅栏效应的影响。因此我将原来的 FFT 点数增大至原来的 4 倍，此时频域法检测的心率变化曲线不再出现明显的跳变，与实际心率更加吻合。与此同时，在绘制心率变化曲线时有一段心率明显超过了 100，与理论心率与存在偏差。刚开始我并不理解这个异常出现的原因，还以为是算法出现了错误。后在[redacted]的启发下我通过查看异常段的心电信号的时域波形，发现这一段心率确实是明显加快，心电数据本身存在异常，验证了自己设计算法的正确性。

本次实验让我深刻体会到理论与实践结合的重要意义。之前在课堂上学习的 FFT 频谱分析、IIR 滤波器设计等理论知识，通过这次的心率检测系统开发得到了全面应用。实验中，我不仅使用带通滤波器对心电信号进行滤波处理，利用 FFT 对滤波前后的心电信号进行频谱分析，还通过时域法和频域法实现了心电信号的心率检测，进一步加深了对心电信号特征与心率计算原理的理解，真切感受到该技术在临床监测、健康管理等领域的应用价值。

最后，我要衷心感谢[redacted]的悉心指导。[redacted]对滤波器设计和频谱分析、时域法和频域法进行心率检测等关键技术进行细致讲解，并指出我实验中存在的问题和进一步的探究方向，帮助我快速理清了系统开发思路。通过本次实验，我不仅夯实了数字信号处理与生理信号分析的基础理论，积累了算法调试与 GUI 开发的实践经验，更培养了科学严谨的实验态度和自主解决问题的思维能力。这些收获将为我后续深入学习信号处理进阶算法、开展生理信号监测相关科研实践奠定坚实基础，对未来的专业学习与发展产生积极而深远的影响。

# 实验九 空域滤波器设计

## 一、 总体方案设计

### 1. 需求分析

本项目面向滤除空间中的干扰信号的需求，利用 MATLAB 中 APP 设计实现空域滤波器设计和信号滤波处理等一系列功能。项目的全部功能均可通过图形窗口界面进行可视化交互操作，不需要再通过编写代码的方式实现。具体的需求分析阐述如下：

- 核心参数配置：支持空域滤波器设计的核心参数配置，用户可在 GUI 界面中输入阵元个数、快拍个数、采样频率、期望信号和干扰信号的入射角度、幅度、频率、是否添加噪声和信噪比等核心参数，并按照题目要求提供默认值。
- 空域滤波器算法实现：支持按照用户输入的参数计算导向矢量、生成滤波前接收信号、估计协方差矩阵、求解最优权矢量并对接收信号进行空域滤波处理。
- 结果可视化展示：支持展示权矢量幅度和相位变化图、空域滤波器方向图、滤波前后时域波形对比图，直观展示空域滤波器的设计效果。
- 性能指标计算：支持信号干扰比（SIR）、信干噪比（SINR）和 3dB 波束宽度的计算和展示，衡量空域滤波器的抗干扰能力与方向分辨率。

### 2. 设计目标

- 软件系统设计：开发一款可视化的 MATLAB App 工具，实现空域滤波器设计和信号滤波处理等一系列功能。
- 用户友好交互：设计的空域滤波器软件系统提供友好的用户界面，便于用户输入相应的参数进行设计和处理，操作便捷，并提供状态信息和错误告警。
- 多维度展示：通过空域滤波器方向图、滤波前后信号时域波形的展示，并输出滤波后信号的关键性能指标，直观呈现空域滤波器滤波效果。

### 3. 设计方案

#### 3.1. 系统总体结构

本系统基于 MATLAB App Designer 开发，采用模块化分层架构，整体分为 GUI 交互层、功能模块层和数据存储层这三部分。每一层均实现了相应的系统功能，具体如下所示：

- GUI 交互层：提供用户操作入口和结果展示窗口，包括按钮、参数输入框、坐标轴等组件。
- 功能模块层：空域滤波器设计系统的核心算法实现层，用于封装导向矢量计算、协方差矩阵估计、最优权矢量求解和信号滤波处理等算法。

- 数据存储层：负责存储仿真过程中的中间数据与最终结果，为 GUI 交互层和功能模块层提供数据支持。

### 3.2. 系统功能模块规划与核心功能实现

系统的功能模块层分为 GUI 界面配置模块、参数配置模块、空域滤波器设计模块、性能指标计算模块、结果可视化模块和辅助功能模块。各模块独立封装、功能明确，具体实现功能阐述如下：

#### 3.2.1. GUI 界面配置模块

该模块负责创建并布局整个 APP 所有图形组件，初始化组件状态和默认值。

#### 3.2.2. 参数配置模块

该模块负责读取用户在 GUI 界面输入的空域滤波器设计参数，验证参数的有效性，并将其输入到空域滤波器设计模块。

#### 3.2.3. 空域滤波器设计模块

该模块负责根据用户输入的参数进行空域滤波器设计和原始信号的滤波处理，是整个程序代码的核心模块。该模块首先计算导向矢量，然后生成原始信号和接收信号，根据接收信号和用户输入的快拍数估计协方差矩阵，并加入对角加载提高数值稳定性。接下来根据协方差矩阵计算最优权矢量，根据最优权矢量对接受信号进行滤波并计算每个角度的方向图响应，最后通过结果可视化模块绘制权矢量幅度和相位变化图、空域滤波器方向图、滤波前后时域波形对比图。

#### 3.2.4. 性能指标计算模块

该模块负责计算输出信号的信号干扰比（SIR）、信干噪比（SINR）和 3dB 波束宽度，并将结果展示出来。

#### 3.2.5. 结果可视化模块

该模块负责绘制权矢量幅度和相位变化图、空域滤波器方向图、滤波前后时域波形对比图。

#### 3.2.6. 辅助功能模块

该模块提供了清除图像、关闭界面的辅助功能，给出了当前系统状态和操作提示，并对数据缺失、参数异常等状态进行错误告警，提升用户的使用体验。

### 3.3. 各模块之间的关系

各功能模块相互协作、数据互通，大致按照操作顺序递进关联。在初始阶段 GUI 界面配置模块首先执行，创建并展示整个 GUI 界面。接下来用户在 GUI 界面输入参数，通过参数配置模块读取并验证参数，并将其传递至空域滤波器设计模块。在空域滤波

器设计模块中完成整个空域滤波器设计和信号滤波处理过程，并将计算结果存储到数据存储层。最后性能指标计算模块和结果可视化模块从数据存储层读取参数，计算性能指标并将相关结果可视化展示。

### 4. 开发环境

- 操作系统：Windows11 64 位操作系统。
- 开发软件：MATLAB R2025b。
- MATLAB 工具包：App Designer（用于 GUI 设计与代码生成），Signal Processing Toolbox（用于信号生成、频谱分析、滤波器设计的核心函数），MATLAB Base Toolbox（用于数据处理、图形绘制、字符串操作等基础功能）。
- 硬件配置：CPU Intel Core i7，内存 16GB，硬盘剩余空间 350GB，满足 MATLAB 软件运行与数据处理的硬件需求。
- 开发文档：MATLAB 官方文档，数字信号处理、信号与系统等信号处理教材。

## 二、 系统的设计与实现

### 1. 软件系统界面设计

本软件系统基于 MATLAB App 设计工具箱开发，采用左侧参数控制和性能指标面板+右侧可视化结果展示的经典布局，整体界面简洁直观，操作流程清晰，可以完成基本空域滤波器设计和信号滤波处理的基本功能，初始化界面如图 1 所示：

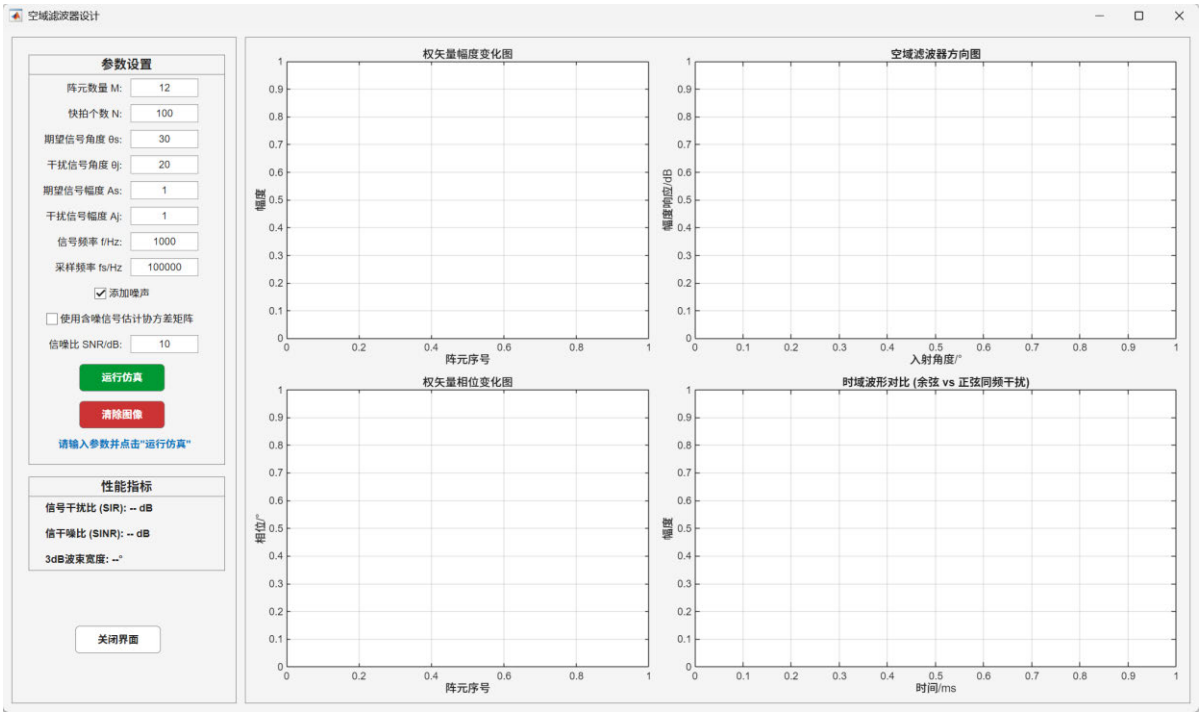


图 1：标签页 1 初始化界面

上图所示界面为直接打开系统，未经任何操作的初始化界面，对该界面的详细介绍如

下所示：

### 1.1. 左侧区域

左侧区域从上到下分别为参数设置面板和性能指标面板。在参数设置面板中用户可输入阵元数量、快拍个数、采样频率、期望信号和干扰信号角度、幅度、频率等空域滤波器设计参数，并可以选择是否添加噪声并输入信噪比，与此同时系统还支持估计协方差矩阵是否使用加噪后的信号进行，探究噪声对空域滤波器设计的影响。在这些输入参数的下方是“运行仿真”、“清除图像”按钮，点击“运行仿真”按钮后可按照输入参数进行空域滤波器设计和信号的滤波处理。点击“清除图像”按钮即可清除坐标区绘制的所有图像。参数设置面板的下方是性能指标面板，该面板在用户点击“运行仿真”按钮进行空域滤波器设计后显示信号干扰比（SIR）、信干噪比（SINR）和 3dB 波束宽度等参数。在左侧区域的最下方是“关闭界面”按钮，用户点击后可立即关闭界面。

### 1.2. 右侧区域

右侧区域包含 4 个 2×2 排列的坐标区。其中左上方和左下方的坐标区用于绘制权矢量幅度和相位变化图，右上方的坐标区用于绘制所设计的空域滤波器的方向图，右下方的坐标区用于绘制滤波处理前后信号的时域波形，并叠加绘制理想期望信号进行对比。这些图像在用户点击“运行仿真”按钮后自动绘制。

## 2. 核心模块的具体实现方法

### 2.1. 基本原理

本实验的核心模块是空域滤波器设计模块，封装了空域滤波器设计和信号滤波处理的关键代码，具体原理介绍如下。

实验中设计一种相控阵天线法对不同方向入射的信号进行滤波处理，以达到保留期望方向上的信号并抑制干扰方向上的信号的目的。在这种方法中，阵列输出选取一个适当的加权向量以补偿各个阵元的传播延时，从而使在某一期望方向上阵列输出可以同相叠加，进而使阵列在该方向上产生一个主瓣波束，而对其他方向上产生较小的响应，用这种方法对整个空间进行波束扫描就可确定空中待测信号的方位。

以一维 M 元线阵为例，其阵列模型如图 2 所示，假设其接收到 L 个窄带信号（信源），阵元间距为$d$（一般假设为波长的一半），θ 为信号入射方向与法向量所形成的角。

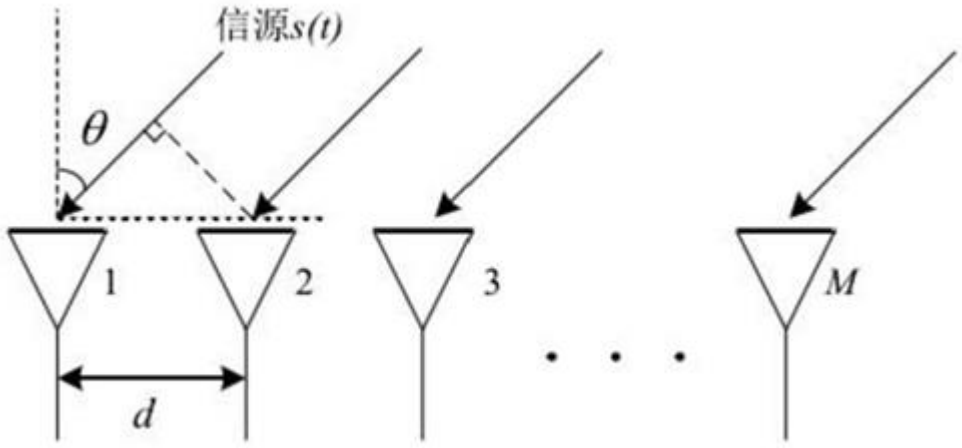


图 2：M 元均匀线阵

相邻两个阵元接收的信号到达时间差为$t=\frac{dsin\theta}{c}$，相位差为$\tau^{(i)}=\frac{2\pi f(i-1)dsin\theta}{c}=\frac{2\pi dsin\theta}{\lambda},i=1,\cdots,M$，那么针对单个辐射源$(\theta)$，阵列接收信号$X(t)$可以表示为：

$$X(t)=\begin{bmatrix}x_1(t)\\x_2(t)\\\cdots\\x_M(t)\end{bmatrix}=\begin{bmatrix}1\\e^{\frac{j2\pi dsin\theta}{\lambda}}\\\cdots\\e^{\frac{j2\pi(M-1)dsin\theta}{\lambda}}\end{bmatrix}s(t)=a(\theta)s(t) \tag{1}$$

其中，$a(\theta)$叫做导向矢量，为$M\times 1$的矩阵，推广到多个辐射源$(\theta_1,\theta_2,\ldots,\theta_k)$：

$$X(t)=a(\theta_1)s_1(t)+a(\theta_2)s_2(t)+\cdots+a(\theta_k)s_k(t)=[a(\theta_1),a(\theta_2),\ldots,a(\theta_k)]\begin{bmatrix}s_1(t)\\s_2(t)\\\cdots\\s_k(t)\end{bmatrix}$$

$$=A(\theta)S(k) \tag{2}$$

考虑噪声影响，$n(t)=\begin{bmatrix}n_1(t)\\n_2(t)\\\cdots\\n_M(t)\end{bmatrix}$。因此，均匀线阵信号模型为 $X(t)=A(\theta)S(k)+n(t)$。

若假设空间源场中有一个感兴趣的信号$s(t)$(或称期望信号，其波达方向为$\theta_d$)和$J$个不感兴趣的信号$i_j(t),j=1,\ldots,J$(或称干扰信号，其波达方向为$\theta_{i_j}$)。则阵列接收信号用矩阵形式表示，则有：

$$\begin{bmatrix}x_1(t)\\x_2(t)\\\cdots\\x_M(t)\end{bmatrix}=\left[a(\theta_d),a\left(\theta_{i_1}\right),\ldots,a\left(\theta_{i_j}\right)\right]\begin{bmatrix}s(t)\\i_1(t)\\\cdots\\i_J(t)\end{bmatrix}+\begin{bmatrix}\mathrm{n}_1(\mathrm{t})\\\mathrm{n}_2(\mathrm{t})\\\cdots\\\mathrm{n}_\mathrm{M}(\mathrm{t})\end{bmatrix} \tag{3}$$

波束形成算法：每个通道用一个复加权系数来调整该通道的幅度和相位。对于不

同的权矢量，来自不同方向的电磁波有不同的响应，从而形成不同方向的空间波束，通过调整权系数 w，使得天线阵列在某一方向上增益最大，也就指向了某个方向，其算法原理如图 3 所示：

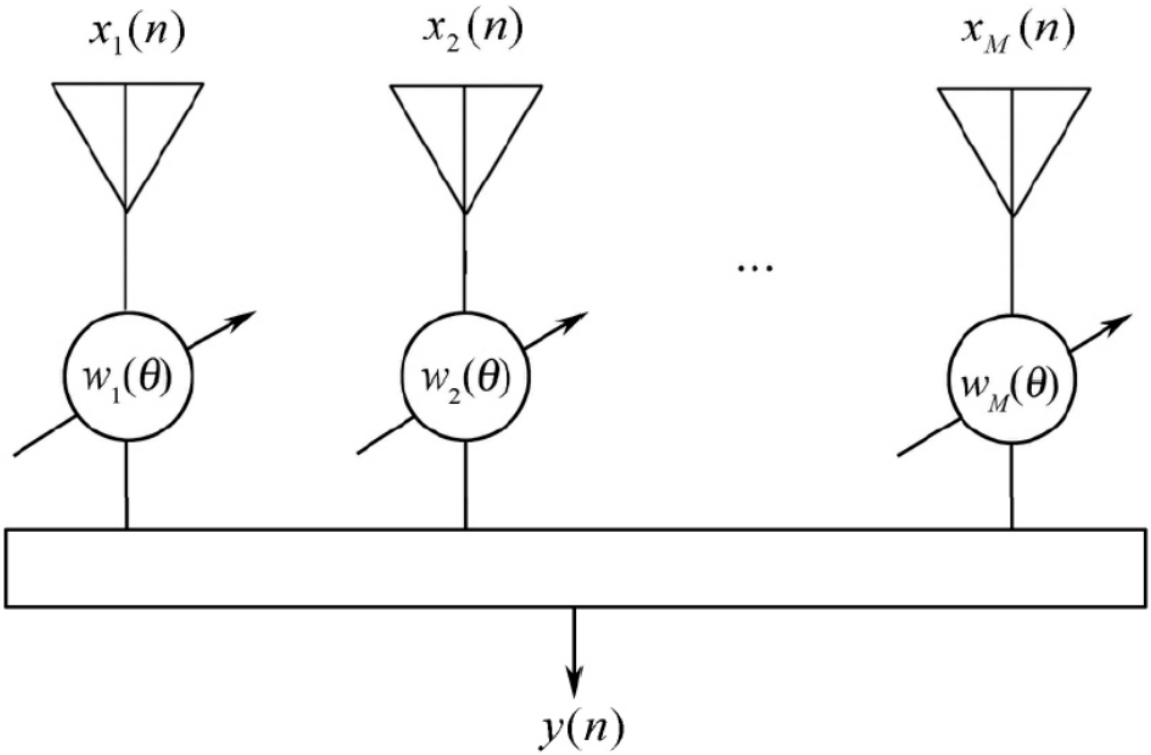


图 3：波束成型算法示意图

N 个快拍的波束形成器输出可以表示为：

$$y(t)=\sum_{i=1}^{M} w_i^*(\theta)x_i(t)=w^H(\theta)x(t),(t=1,\dots,N) \tag{4}$$

其输出的平均功率为：

$$P(w)=E\{|y(t)|^2\}=w^HE\{x(t)x^H(t)\}w=w^HRw \tag{5}$$

其中$R$为阵列输出的协方差矩阵，可采用前$N$个快拍的实际信号进行估计。注意为了提高数值稳定性，需要对估计的协方差矩阵对角线元素加一个小量进行对角加载，避免奇异矩阵无法求逆。

为了保证方向$\theta_d$期望信号的正确接收，并且完全抑制其他 J 个干扰，由此得到关于权矢量的约束条件：

$$\begin{cases} w^Ha(\theta_d)=1 \\ w^Ha(\theta_{ij})=0 \end{cases} \tag{6}$$

因此抑制干扰和噪声应该一同考虑，此时波束形成器最佳权矢量可以描述为

$$\min_w P(w),st.w^Ha(\theta_d)=1 \tag{7}$$

运用拉格朗日乘子法，构造代价函数$J(w)$为：

$$J(w) = w^H R w - \eta\left(1 - w^H a(\theta_d)\right) \tag{8}$$

可以解得：

$$w = \eta R^{-1} a(\theta_d) \tag{9}$$

两端取共轭转置，并且右乘$a(\theta_d)$，同时注意到约束条件$w^H a(\theta_d) = 1$，可以得到：

$$\eta = \frac{1}{a(\theta_d)^H (R^{-1})^H a(\theta_d)} \tag{10}$$

最终解得权矢量为：

$$w = \frac{R^{-1} a(\theta_d)}{a(\theta_d)^H (R^{-1})^H a(\theta_d)} \tag{11}$$

该权矢量可以使$a(\theta_d)$对应方向的信号通过，其余方向的信号和噪声最大程度的抑制。

### 2.2. 关键 MATLAB 函数

- exp：自然指数运算，构造导向矢量；
- randn：标准高斯白噪声生成，用于添加噪声；
- \：矩阵伪逆求解，求解线性方程组，提高数值稳定性；
- trace：矩阵迹计算，用于确定对角加载的修正系数；
- eye：单位矩阵生成，用于对角加载
- abs：绝对值/复数模计算，求解方向图响应幅度与权矢量的模值；
- log10：以 10 为底的对数计算，将归一化的方向图幅度转换为 dB 值；
- max：最大值求解，确定方向图峰值用于幅度归一化；

### 2.3. 核心代码

空域滤波器设计的核心代码由函数 steering_vector、generate_and_estimate_covariance、calculate_mvdr_weights 和 calculate_pattern 组成，分别实现导向矢量计算、接收信号生成与协方差矩阵估计、最优权矢量求解和空域滤波器方向图计算等功能，核心代码如下所示：

```
% 计算导向矢量
function a = steering_vector(~, theta, M, d, lambda)
    theta_rad = deg2rad(theta);
    a = zeros(M, 1);
    for m = 0:M-1
        a(m+1) = exp(1j * 2 * pi * d * m * sin(theta_rad) / lambda);
    end
end

% 生成测试信号并估计协方差矩阵
```

```
function [R_estimated, Xt, st, jt, a_s, a_j, time_axis] = generate_and_estimate_covariance(app, M, N, theta_s, theta_j, As, Aj, add_noise, SNR_dB)
    % 信号参数
    f = app.fEditField.Value; % 信号频率 (Hz)
    fs = app.fsEditField.Value; % 采样频率 (Hz)
    c = 3e8;
    lambda = c / f; % 波长
    d = lambda / 2; % 阵元间距

    % 导向矢量
    a_s = app.steering_vector(theta_s, M, d, lambda);
    a_j = app.steering_vector(theta_j, M, d, lambda);
    Ns = max(N, 5*fs);

    % 时间轴
    time_axis = (0:Ns-1) / fs;

    % 数字角频率
    omega = 2 * pi * f / fs;

    % ==================== 生成测试信号 ====================
    % 期望信号: 正弦信号
    % 干扰信号: 余弦信号（与正弦同频，相位差 90 度）
    st = As * sin(omega * (0:Ns-1));
    jt = Aj * cos(omega * (0:Ns-1));
    % st = As * cos(omega * (0:N-1));
    % jt = Aj * sin(omega * (0:N-1));

    % 噪声处理
    if add_noise
        % 信号功率
        Ps = As^2 / 2;  % 正弦信号功率 = A^2/2

        % 噪声功率
        Pn = Ps / (10^(SNR_dB/10));

        % 生成实数高斯噪声
        nt = sqrt(Pn) * randn(M, Ns);
    else
        nt = zeros(M, Ns);
    end

    % 接收信号
    Xt = a_s * st + a_j * jt + nt;

```

```
    % ==================== 估计协方差矩阵 ====================
    % 使用快拍估计协方差矩阵：R = (1/N) * X * X^H
    if app.RCheckBox.Value == 1
        X_t = Xt(:,1:N);
        R_estimated = (X_t * X_t') / N;
    else
        X_t = a_s * st + a_j * jt;
        X_t = X_t(:,1:N);
        R_estimated = (X_t * X_t') / N;
    end

    % 对角加载（提高数值稳定性）
    epsilon = 1e-6 * trace(R_estimated) / M;
    R_estimated = R_estimated + epsilon * eye(M);
end

% 计算 MVDR 权矢量
function w = calculate_mvdr_weights(~, R, a_s)
    % 使用更稳定的求逆方法
    w = (R \ a_s) / (a_s' * (R' \ a_s));
end

% 计算方向图
function [theta_scan, pattern, pattern_db] = calculate_pattern(app, w, M, ~)
    % 信号频率
    f = app.fEditField.Value;
    c = 3e8;
    lambda = c / f;
    d = lambda / 2;

    theta_scan = -180:0.1:180;
    pattern = zeros(size(theta_scan));

    for k = 1:length(theta_scan)
        a_tmp = app.steering_vector(theta_scan(k), M, d, lambda);
        pattern(k) = abs(w' * a_tmp);
    end

    % 归一化到 0dB
    pattern = pattern / max(pattern);
    pattern_db = 20 * log10(pattern);
end
```

## 三、 系统功能测试

为了验证上述设计的空域滤波器是否可以正常工作，需要对其进行功能测试。在

这里使用实验讲义所给参数对空域滤波器的方向图和滤波能力进行了测试，测试结果如下所示。

**1. 空域滤波器设计（使用含噪信号估计协方差矩阵）**

首先按照默认参数进行空域滤波器设计，在估计协方差矩阵时直接使用生成的含噪接收信号，结果如图 4 所示：

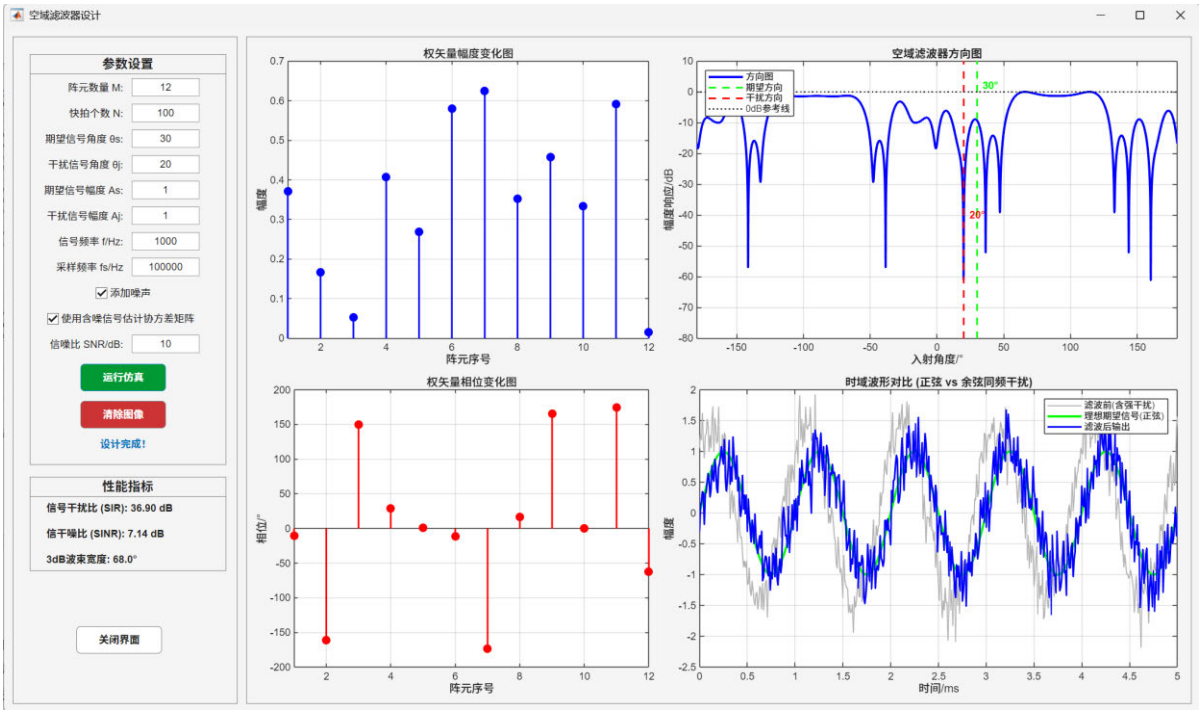


图 4：空域滤波器设计（使用含噪信号估计协方差矩阵）

为了观察方便，信号的时域波形只绘制前 5 个周期的形状。观察上图可以看出，虽然滤波后的输出信号已经在一定程度上恢复出了期望信号，但仍存在比较严重的失真，信号干扰比和信干噪比均较低。并且从方向图可以看出，此时空域滤波器的增益在30°并未达到最大值，而是呈现一种无规律的分布状态，在多个角度增益均出现大于30°时的情况，但对于20°方向的干扰信号有明显的抑制效果。这说明如果还存在其他方向的干扰信号，空域滤波器就无法有效滤除。与此同时在不同的试验条件下，信号噪声的分布不同，最优权矢量的幅度和相位也不同，并且方向图的分布也会发生较大的变化，这会对空域滤波器的设计造成较大的困难。为了降低空域滤波器设计的难度，降低滤波后信号的失真程度并提高空域滤波器的方向选择性，因此在设计空余滤波器时应使用无噪的期望信号和干扰信号的叠加信号估计协方差矩阵。

**2. 空域滤波器设计（使用无噪信号估计协方差矩阵）**

在这里同样使用默认参数进行空域滤波器设计，只是在估计协方差矩阵时改用无噪的期望信号和干扰信号的混合信号来进行，结果如图 5 所示：

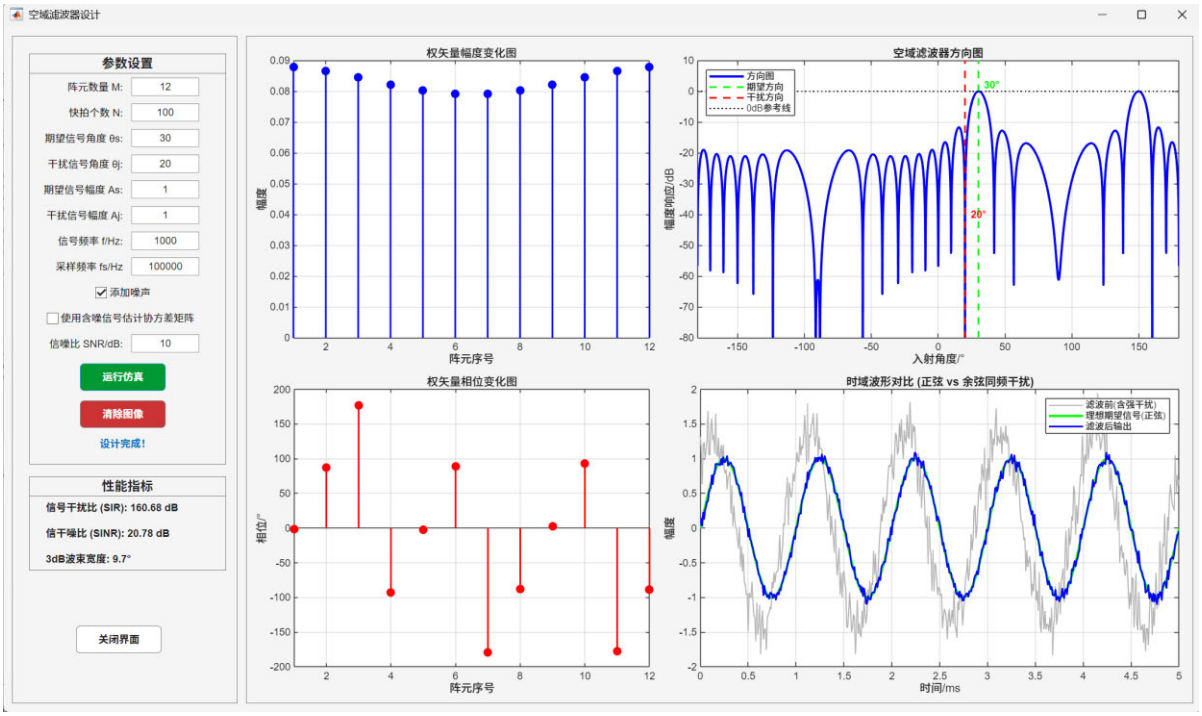


图 5：空域滤波器设计（使用无噪信号估计协方差矩阵）

观察上图可以看出，此时空域滤波器的方向图在30°和150°时出现了明显的峰值，在其他方向均在-10dB 以下，3dB 波束宽度仅为9.7°，这说明此时空域滤波器具有良好的方向选择性，能够有效滤除其他方向上的干扰信号。与此同时，计算出的最优权矢量的幅度和相位是确定值，具体分布如图 5 所示，可据此直接进行空域滤波器设计。滤波后的输出信号虽然仍存在小幅噪声，但与理想期望信号已经十分接近，无明显的幅值、相位失真，信号干扰比和信干噪比均较高，表明此时空域滤波器对期望信号有很好的恢复效果。

通过上述对比分析，在设计空域滤波器时应当使用无噪的理想信号估计协方差矩阵，据此求出最优权矢量，完成空域滤波器设计，再用设计的空域滤波器从接收信号中滤除干扰信号得到期望信号。噪声的大小也会影响空域滤波器的工作效果，在信噪比为 5dB 和无噪条件下，空域滤波器的工作效果如图 6、图 7 所示：

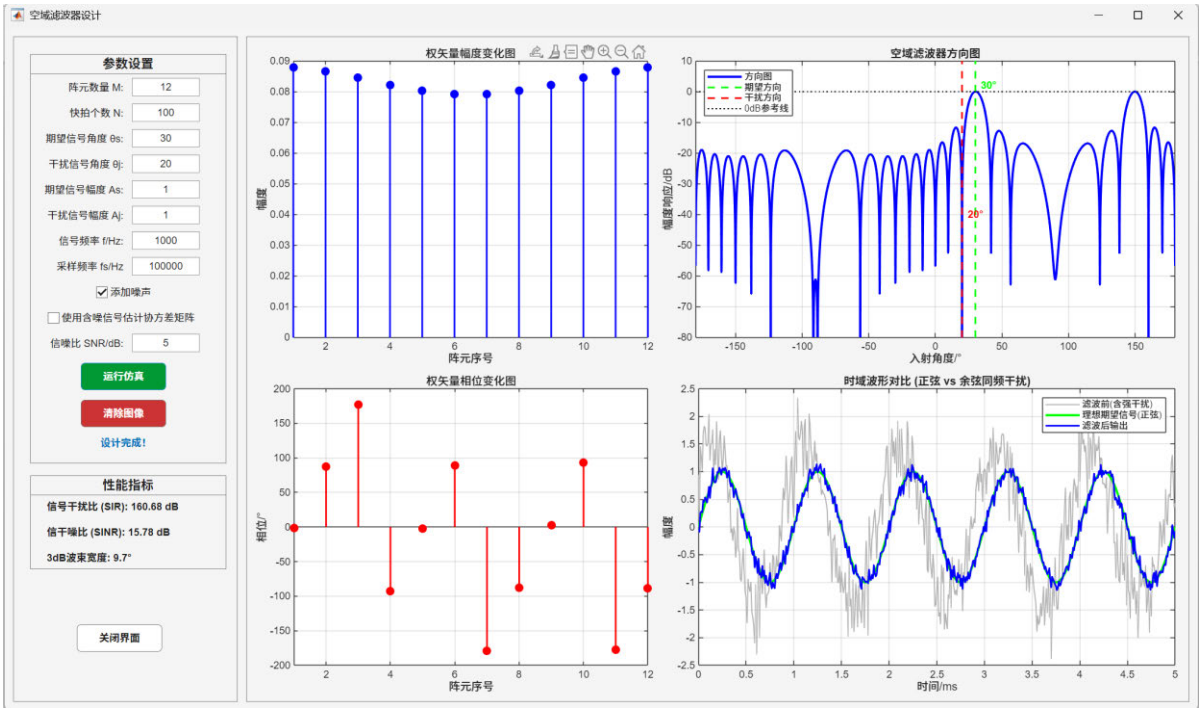


图 6：信噪比 5dB 条件下空域滤波器工作效果

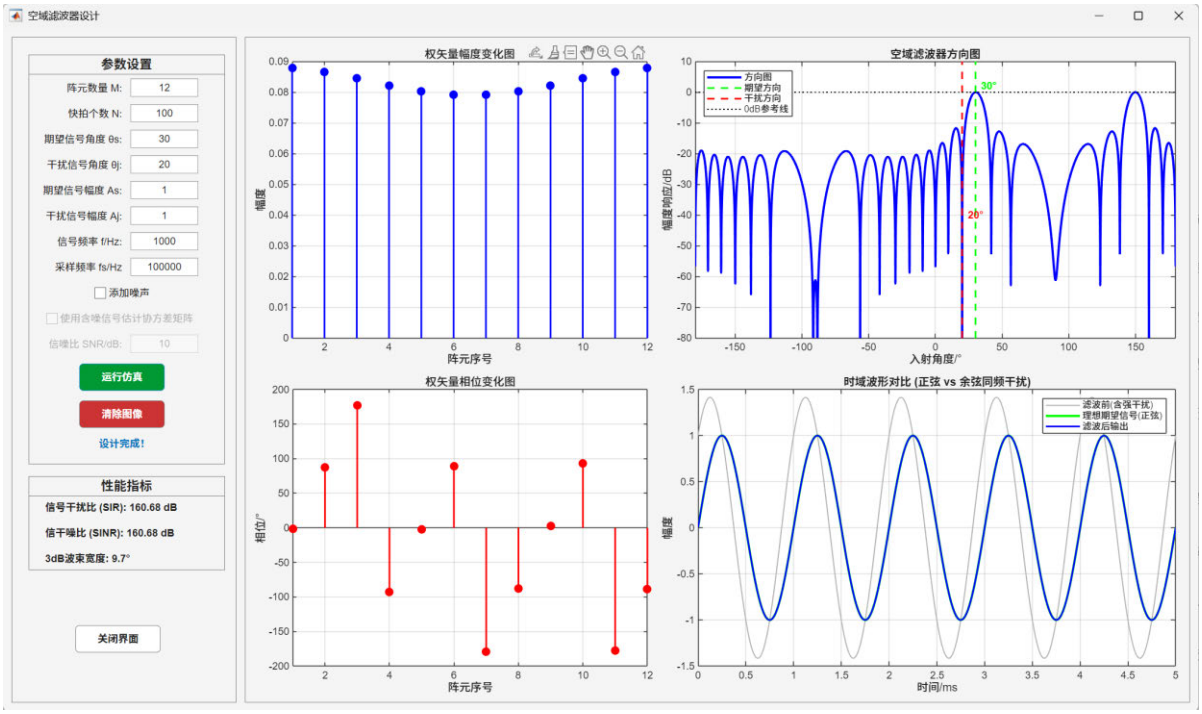


图 7：无噪条件下空域滤波器工作效果

观察上图可以看出，信噪比为 5dB 时滤波后信号的失真程度明显大于 10dB 的情况，信干噪比也进一步降低。但是由图 5 和图 6 可以看出，此时信号的信干噪比低于原始信号的信噪比，这说明空域滤波器是具有一定的滤波降噪效果的，可以滤除其他方向上的噪声。由图 7 可以看出，无噪条件下滤波后的信号与期望信号几乎完全一致，几乎不存在幅值、相位失真，信号干扰比非常高，干扰信号已经几乎滤除干净。这说明

空域滤波器在无噪条件下具有很好的工作效果，可以有效滤除干扰信号。

为进一步探究空域滤波器对干扰方向上信号的滤除效果，将期望信号角度和干扰信号角度进行互换，结果如图 8 所示：

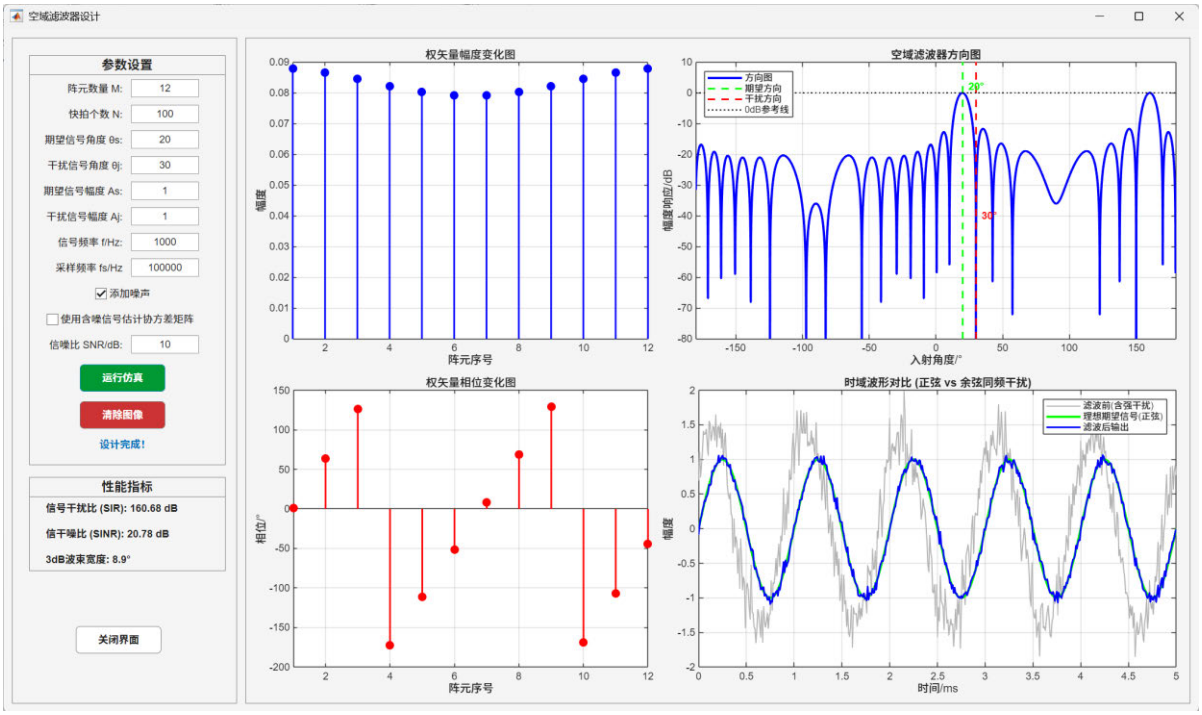


图 8：期望信号角度和干扰信号角度互换后空域滤波器工作效果

观察上图可以看出，此时空域滤波器的方向图在20°和160°时出现了明显的峰值，在其他方向均在-10dB 以下，3dB 波束宽度仅为8.9°。这说明此时空域滤波器仍可有效选择20°的期望信号，具有良好的方向选择性，进一步验证了空域滤波器设计的正确性。

## 3. 优缺点

### 3.1. 优点

- 方向选择性强：在保证期望信号无失真的条件下，有效滤除其他方向上的干扰信号，并且 3dB 波束宽度较窄。
- 可分离同频信号：依托设计的空域滤波器，可有效分离同频信号，具有较大的应用价值。
- 信号保真度好：通过权矢量约束条件的设计确保滤波后的信号相较于期望信号无幅值、相位失真，滤波后的信号与期望信号高度吻合。

### 3.2. 缺点

- 依赖协方差矩阵：当快拍数不足时，矩阵估计误差会导致方向图零点偏移、干扰抑制失效。

- 计算复杂度高：矩阵求逆复杂度为$O(M^3)$（M 为阵元数），大规模阵列下计算速度较慢。
- 缺乏自适应性：需预先知晓信源信息，在期望信号和干扰信号发生变化时无法自适应调整。

# 四、 总结与体会

通过本次空域滤波器设计实验，我收获了扎实的信号处理专业知识与 MATLAB APP 开发实践经验，对空域滤波器设计的核心原理与实现方法有了更为透彻的理解。实验过程中，我不仅深入掌握了均匀线阵模型、导向矢量构造、协方差矩阵估计等空域滤波核心理论，还熟练运用 MATLAB 实现了最优权矢量求解、方向图计算、多维度结果可视化等关键算法，成功开发出空域滤波器设计和信号滤波处理的模块化 GUI 系统。

其中，实验中我遇到的最大问题是协方差矩阵估计不准确导致方向图零点偏移，干扰抑制效果未达预期。初期设计时，使用有限快拍数估计协方差矩阵后，方向图在干扰信号入射方向未形成有效深空零点，干扰抑制能力较弱。为解决这一问题，我反复调试代码并回顾理论知识，最终发现是协方差矩阵存在数值奇异性导致权矢量求解偏差。通过加入对角加载修正，有效提升了矩阵的数值稳定性，最终使得干扰信号的抑制效果显著提升。

本次实验让我深刻体会到理论与实践结合的重要意义。之前在课堂上学习的阵列信号处理、矩阵运算、方向图计算等理论知识，通过这次空域滤波器系统开发得到了全面应用。实验中，我不仅实现了对同频正交信号的空域分离，验证了滤波器对期望信号的无失真保留与干扰抑制能力，还通过量化 SIR、SINR、3dB 波束宽度等指标，进一步加深了对空域滤波方向选择性、鲁棒性等核心特性的理解，真切感受到该技术在雷达、通信、声呐等领域的应用价值。

最后，我要衷心感谢[redacted]的悉心指导。[redacted]在课堂上为我们耐心讲解了导向矢量构造、协方差矩阵估计、权矢量求解等关键技术细节，为我顺利完成空域滤波器的设计提供了巨大的帮助。通过本次实验，我不仅夯实了数字信号处理与空域滤波的基础理论，积累了算法调试与 MATLAB APP 信号处理系统开发的实践经验，更培养了科学严谨的实验态度和自主解决问题的思维能力。这些收获将为我后续深入学习进阶波束形成算法、开展阵列信号处理相关科研实践奠定坚实基础，对未来的专业学习与发展产生积极而深远的影响。

# 实验十 包络检测器设计

## 一、 总体方案设计

### 1. 需求分析

本项目面向窄带通信系统中无线电信号$S(t) = bcos(\omega_0 t)$检测的需求，利用MATLAB中APP设计实现信号包络的提取和目标无线电信号的检测。项目的全部功能均可通过图形窗口界面进行可视化交互操作，不需要再通过编写代码的方式实现。具体的需求分析阐述如下：

- 信号与噪声处理：生成符合要求的高斯白噪声，通过窄带带通滤波器得到方差为1的窄带噪声，构造含高频无线电信号和噪声的接收信号$V(t)$。
- 接收信号包络提取：基于希尔伯特变换提取接收信号的包络$a(t)$。
- 阈值与漏警率求解：针对不同信噪比通过概率密度函数的交点确定检测阈值$V_T$，分别计算理论漏警率和1000次仿真的实际漏警率。
- 结果可视化展示：展示窄带滤波器幅频响应、信号和噪声时域波形与功率谱、概率密度函数（含理论与实际分布）、幅度、阈值、方差等参数随信噪比变化曲线及漏警率变化曲线。

### 2. 设计目标

- 软件系统设计：开发一款可视化的MATLAB App工具，实现包络检测器的设计和目标无线电信号检测等一系列功能。
- 用户友好交互：设计的包络检测器软件系统提供友好的用户界面，便于用户输入相应的参数进行设计和处理。软件系统操作便捷，用户在切换下拉框选项时可自动切换绘图，系统提供状态信息和错误告警。
- 计算准确性和运行效率：阈值求解误差控制在合理范围，实际漏警率与理论值的偏差符合统计仿真要求。1000 次实际信号仿真需在可接受时间内完成，配备进度条实时展示处理进度。

### 3. 设计方案

#### 3.1. 系统总体结构

本系统基于MATLAB App Designer开发，采用模块化分层架构，整体分为GUI 交互层、功能模块层和数据存储层这三部分。每一层均实现了相应的系统功能，具体如下所示：

- GUI交互层：GUI可视化组件，包括输入框、下拉框、按钮、坐标轴等组件，负责参数输入、用户操作响应与结果展示。

- 功能模块层：包络检测器设计和目标无线电信号检测核心算法实现层，用于封装窄带滤波器设计、噪声和信号生成、信号包络提取、检测阈值计算和漏警概率求解等算法。
- 数据存储层：负责存储仿真过程中的中间数据与最终结果，为 GUI 交互层和功能模块层提供数据支持。

### 3.2. 系统功能模块规划与核心功能实现

系统的功能模块层分为参数配置与校验模块、窄带滤波器设计模块、噪声和信号生成模块、信号包络提取模块、检测阈值与漏警率计算模块和结果可视化模块。各模块独立封装、功能明确，具体实现功能阐述如下：

#### 3.2.1. 参数配置和校验模块

该模块负责接收用户输入的所有参数，对其合法性进行校验，并将其输入到后续核心算法实现的模块。

#### 3.2.2. 窄带滤波器设计模块

该模块根据用户输入的中心频率和带宽的参数计算滤波器截止频率，根据输入的滤波器阶数设计巴特沃斯带通滤波器，计算幅频响应并通过结果绘制模块绘图。

#### 3.2.3. 噪声和信号生成模块

该模块首先生成高斯白噪声，使用设计的窄带滤波器进行滤波得到窄带高斯白噪声，并将其方差归一化至用户设定值。根据用户输入的信噪比计算信号幅度，与窄带高斯白噪声叠加后构成接收信号$V(t)$。

#### 3.2.4. 信号包络提取模块

该模块通过对接收信号进行希尔伯特变换得到其解析信号，取绝对值即可提取接收信号的包络。

#### 3.2.5. 检测阈值和漏警概率计算模块

该模块负责计算窄带高斯白噪声和不同信噪比信号的包络的理论概率密度函数，并通过计算二者的交点求出相应信噪比信号的检测阈值，通过数值积分的方法求解信号包络理论概率密度函数在检测阈值左侧曲线下的面积求解漏警概率。

#### 3.2.6. 结果可视化模块

该模块负责绘制窄带滤波器幅频响应、信号和噪声时域波形与功率谱、概率密度函数（含理论与实际分布）、幅度、阈值、方差等参数随信噪比变化曲线及漏警率变化曲线。

### 3.3. 各模块之间的关系

各功能模块相互协作、数据互通，大致按照操作顺序递进关联。在初始阶段参数配置和校验模块读取并验证用户输入的参数，并将其传递至窄带滤波器设计模块。噪声和信号生成模块中窄带高斯白噪声的生成需要窄带滤波器设计模块设计的滤波器对高斯白噪声进行滤波处理，生成的信号再输入到信号包络提取模块提取包络。包络数据输入到检测阈值和漏警概率计算模块计算阈值和漏警概率，最后结果可视化模块读取前面所有模块的数据，将其计算结果直观地展现出来。

**4. 开发环境**

- 操作系统：Windows11 64 位操作系统。
- 开发软件：MATLAB R2025b。
- MATLAB 工具包：App Designer（用于 GUI 设计与代码生成），Signal Processing Toolbox（用于信号生成、频谱分析、滤波器设计的核心函数，如 fft、periodogram、butter、freqz 等），Statistics and Machine Learning Toolbox（提供 ksdensity、besseli 等概率统计函数），MATLAB Base Toolbox（用于数据处理、图形绘制、字符串操作等基础功能）。
- 硬件配置：CPU Intel Core i7，内存 16GB，硬盘剩余空间 350GB，满足 MATLAB 软件运行与数据处理的硬件需求。
- 开发文档：MATLAB 官方文档，数字信号处理、信号与系统等信号处理教材。

## 二、 系统的设计与实现

**1. 软件系统界面设计**

本软件系统基于 MATLAB App 设计工具箱开发，采用左侧参数控制面板+右侧可视化结果展示的经典布局，整体界面简洁直观，操作流程清晰，可以完成基本的信号包络检测功能，初始化界面如图 1 所示：

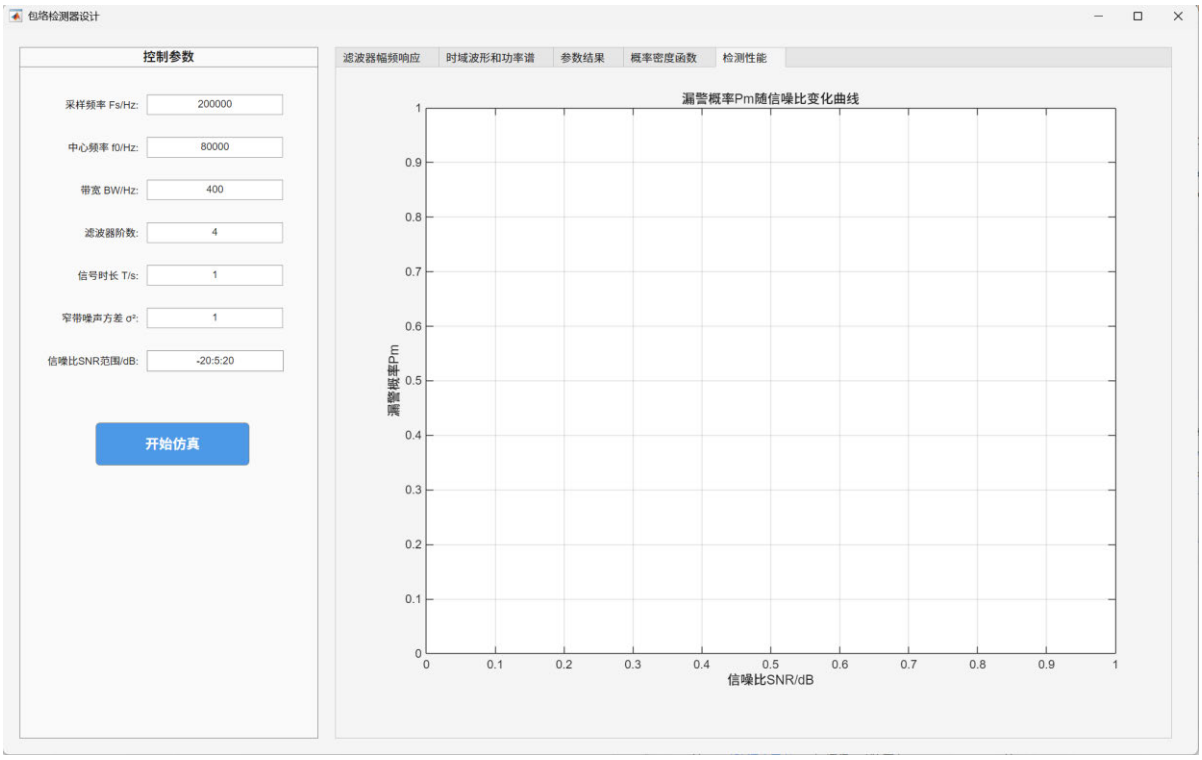


图 1：标签页 1 初始化界面

上图所示界面为直接打开系统，未经任何操作的初始化界面。

界面的左侧是垂直排列的控制参数面板，右侧包含 5 个标签页，每个标签页分别展示不同的结果，用户可通过顶部标签页切换查看实验的各个结果

### 1.1. 左侧区域

左侧区域为控制参数面板。用户在控制参数面板中可输入采样频率、中心频率、带宽、滤波器阶数、信号时长、窄带噪声方差和信噪比范围等设计参数，系统会自动校验输入的设计参数是否合理，默认参数如上图所示。在这些输入参数的下方是“开始仿真”按钮，点击后立即进行包络检测器的设计和漏警概率的计算。

### 1.2. 右侧区域

右侧区域包含 5 个标签页，依次为滤波器幅频响应、时域波形和功率谱、参数结果、概率密度函数和检测性能，每个标签页聚焦单一功能实现，通过独立坐标轴展示对应结果，支持动态刷新与交互操作。

#### 1.2.1. 标签页 1：滤波器幅频响应

标签页 1 仅包含一个坐标区，初始状态如图 1 所示。用于绘制所设计的巴特沃斯窄带滤波器的幅频特性曲线。

#### 1.2.2. 标签页 2：时域波形和功率谱

标签页 2 顶部水平排列“选择信号”和“选择信噪比/dB”下拉框，分别用于选择

信号类型（窄带高斯白噪声$N(t)$、接收信号$V(t)$）以及接受信号的信噪比大小。下部为垂直排列的两个坐标区，分别用于绘制所选中信号的时域波形和功率谱密度。标签页 2 初始状态如图 2 所示：

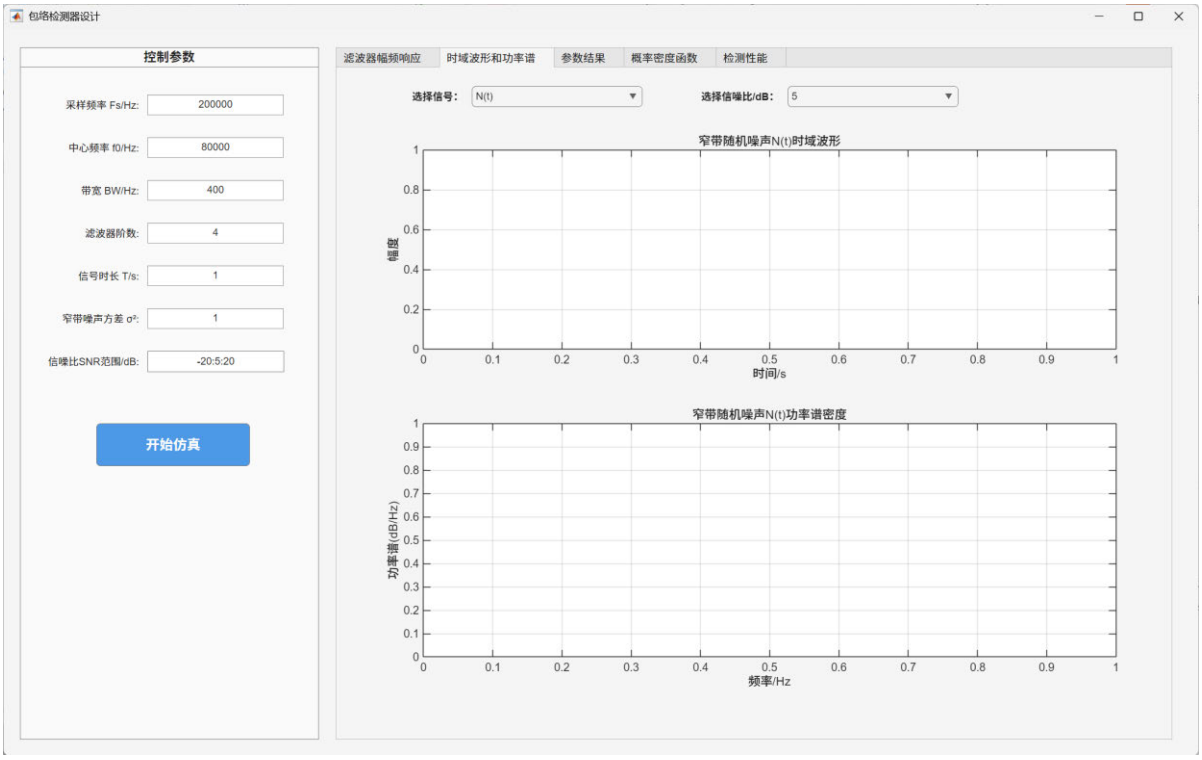


图 2：标签页 2 初始化界面

### 1.2.3. 标签页 3：参数结果

标签页 3 为垂直排列的 3 个坐标轴，分别展示噪声方差、信号幅度和检测阈值随信噪比变化曲线，初始状态如图 3 所示：

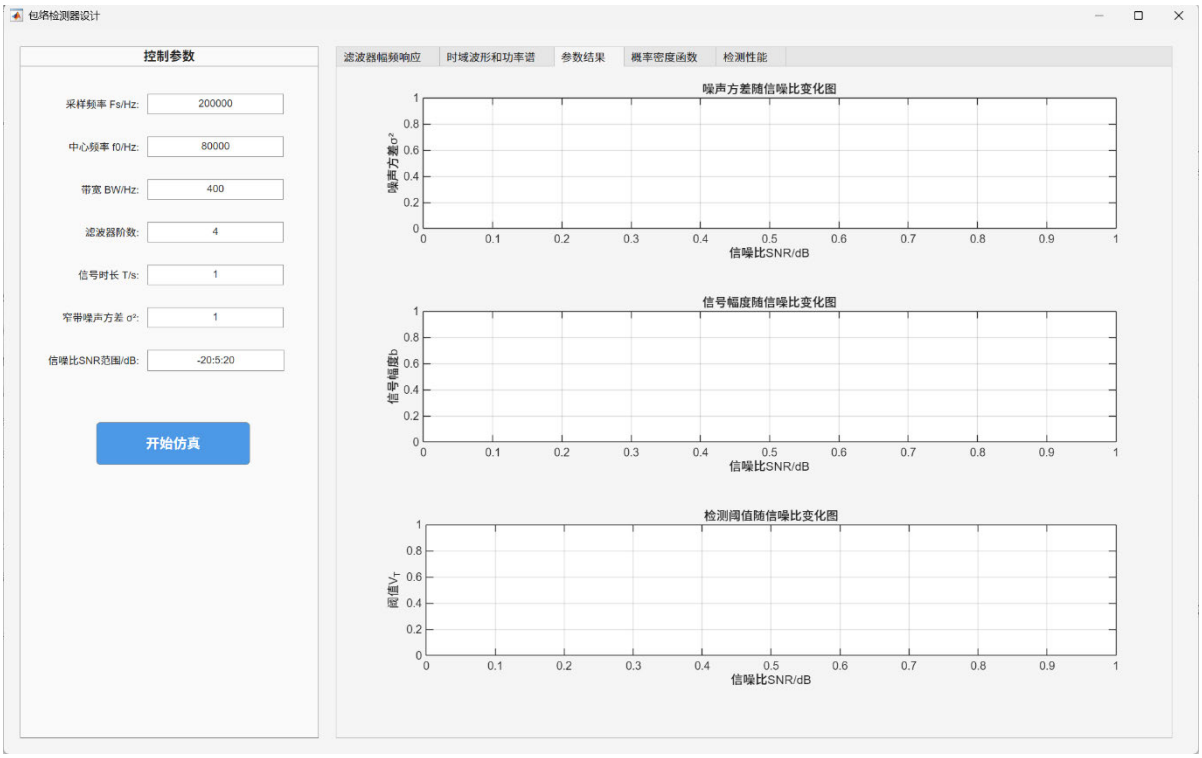


图 3：标签页 3 初始化界面

### 1.2.4. 标签页 4：概率密度函数

标签页 4 的顶部为“选择信噪比/dB”下拉框，用于选择接收信号的信噪比。下方是一个坐标区，用于绘制窄带高斯白噪声和所选接收信号的理论概率密度函数和所选接收信号的实际概率密度函数，并标出检测阈值$V_T$，初始状态如图 4 所示：

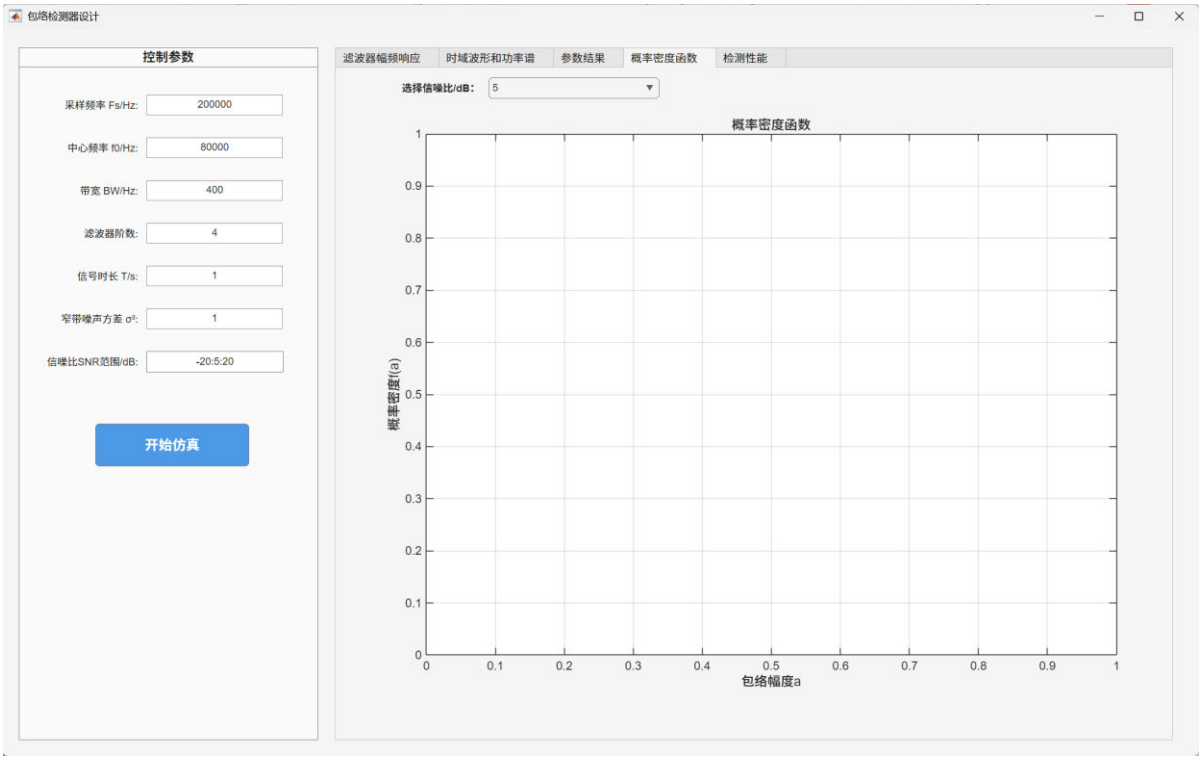


图 4：标签页 4 初始化界面

### 1.2.5. 标签页 5：检测性能

标签页 5 仅包含一个坐标区，初始状态如图 5 所示，以半对数坐标展示理论漏警率和实际漏警率随信噪比变化曲线。

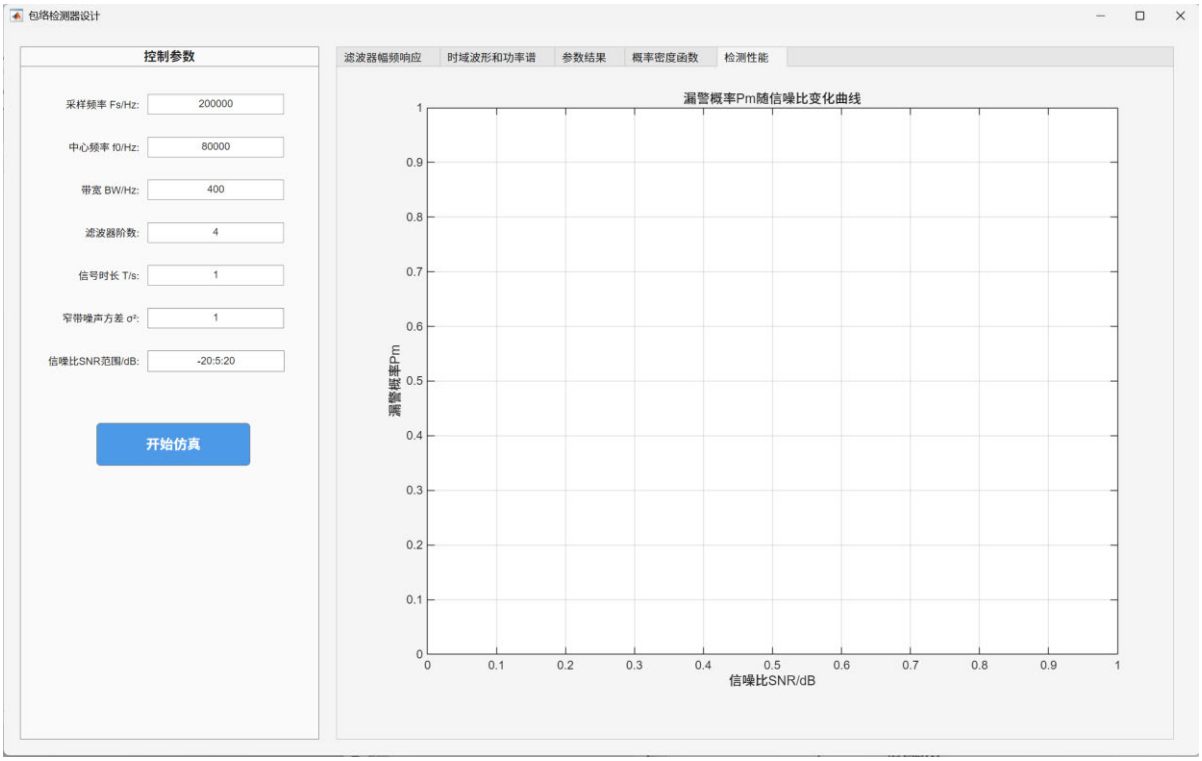


图 5：标签页 5 初始化界面

## 2. 核心模块的具体实现方法

### 2.1. 窄带滤波器设计模块

#### 2.1.1. 基本原理

本实验中窄带滤波器的设计原理和实验八类似，均为设计 IIR 巴特沃斯带通滤波器。根据输入的中心频率和带宽，计算出带通滤波器的截止频率。以采样频率的一半为基准，将截止频率归一化至[0,1]区间。调用 MATLAB 函数计算滤波器系统函数分子和分母系数，最后计算滤波器幅频响应，绘制幅频特性曲线，验证滤波器性能。

#### 2.1.2. 关键 MATLAB 函数

- butter：巴特沃斯滤波器设计函数；
- freqz：计算数字滤波器的频率响应；

#### 2.1.3. 核心代码

```
% 1. 设计窄带带通滤波器并立即更新图形
f_low = app.f0 - app.BW/2;
f_high = app.f0 + app.BW/2;
Wn = [f_low f_high] / (app.Fs/2);
[b_filt, a_filt] = butter(app.order, Wn, 'bandpass');
```

```
6.
7.  % 获取滤波器频率响应
8.  [h_resp, f_resp] = freqz(b_filt, a_filt, 4096, app.Fs);
9.  app.filter_response = [f_resp, 20*log10(abs(h_resp))];
10.
11. % 立即更新滤波器幅频响应图
12. cla(app.FilterAxes);
13. plot(app.FilterAxes, f_resp, 20*log10(abs(h_resp)), 'b', 'LineWidth', 1.5);
14. title(app.FilterAxes, sprintf('窄带带通滤波器幅频响应
    (f_0=%dHz, B=%dHz)', app.f0, app.BW));
15. xlabel(app.FilterAxes, '频率/Hz');
16. ylabel(app.FilterAxes, '幅度/dB');
17. grid(app.FilterAxes, 'on');
18. legend(app.FilterAxes, '窄带带通滤波器幅频响应曲线', 'Location','northeast');
19. xlim(app.FilterAxes, [app.f0-2000, app.f0+2000]);
20. drawnow;  % 立即更新滤波器图形
```

## 2.2. 噪声和信号生成模块

### 2.2.1. 基本原理

为了生成窄带高斯白噪声，首先生成一串均值为 0，方差为 1 的高斯白噪声，利用窄带滤波器设计模块中设计的窄带滤波器进行滤波处理，得到高斯白噪声，最后对其进行归一化将其方差调整至用户设定值，归一化的计算公式为：

$$N(t) = \frac{N(t)}{\sqrt{Var(N(t))}} \tag{1}$$

为了生成实际接收信号，根据信噪比计算信号幅度 b，计算公式为：

$$SNR = 10lg\frac{Var(b)}{Var(N(t))} \tag{2}$$

生成随机相位的正弦信号，与窄带高斯白噪声叠加后得到实际接收信号，其表达式为：

$$V(t) = bcos(\omega_0 t + \theta) + N(t) \tag{3}$$

其中$\theta$为$[0, 2\pi]$内的随机相位

功率谱密度采用周期图法计算，计算公式如下所示：

$$P(f) = \frac{\Delta t}{N}\left|\sum_{n=0}^{N-1} x[n]\, e^{-j2\pi f \Delta t n}\right|^2 \tag{4}$$

其中$\Delta t = \frac{1}{f_s}$为采样间隔。对于单边周期图，除 0 和奈奎斯特频率$\frac{f_s}{2}$外的所有频率处的值都乘以 2 以便保持总功率。为便于观察，绘制图像时将功率谱转换为 dB 刻度。

### 2.2.2. 关键 MATLAB 函数

- randn：生成均值为 0、方差为 1 的高斯白噪声序列；

- filter：对信号进行数字滤波；
- std/var：分别计算信号的标准差和方差
- periodogram：采用周期图法计算功率谱密度

### 2.2.3. 核心代码

```
% 1. 计算信号长度与时间向量
N = round(app.T * app.Fs);
t = (0:N-1) / app.Fs;

% 2. 生成高斯白噪声并滤波得到窄带噪声
noise_raw = randn(1, N);
noise_nb = filter(b_filt, a_filt, noise_raw);

% 3. 归一化窄带噪声至指定方差（默认 sigma2=1）
noise_std = std(noise_nb);
noise_nb = noise_nb / noise_std * sqrt(app.sigma2);

% 4. 存储窄带噪声时域数据（截取前 1000 点，避免 GUI 绘图过于密集）
app.noise_time = [t(1:min(1000,N)); noise_nb(1:min(1000,N))];

% 5. 计算窄带噪声功率谱密度
[Pxx_noise, F_noise] = periodogram(noise_nb, rectwin(length(noise_nb)), 4096, app.Fs);
app.noise_freq = [F_noise, 10*log10(Pxx_noise)];

% 6. 遍历信噪比，生成对应接收信号（以单个 SNR 为例）
snr_val = app.SNR_dB_vec(i); % 取第 i 个信噪比点
b = sqrt(2 * app.sigma2 * 10^(snr_val/10)); % 计算正弦信号幅度
theta = rand * 2 * pi; % 生成[0, 2pi]随机相位
s_t = b * cos(2*pi*app.f0*t + theta); % 生成正弦信号
v_t_display = s_t + noise_nb; % 叠加窄带噪声，得到接收信号

% 7. 存储接收信号时域与频域数据
app.signal_time_data{i} = [t(1:min(1000,N)); v_t_display(1:min(1000,N))];
[Pxx_signal, F_signal] = periodogram(v_t_display, rectwin(length(v_t_display)), 4096, app.Fs);
app.signal_freq_data{i} = [F_signal, 10*log10(Pxx_signal)];
```

## 2.3. 信号包络提取模块

### 2.3.1. 基本原理

对于窄带高斯白噪声$N(t)$，通过如下莱斯表达式，由解调原理，得到两路正交分量$N_I(t)$和$N_Q(t)$，式中$\hat{N}(t)$为$N(t)$的希尔伯特变换：

$$N(t) = N_I(t)cos\omega_0 t - N_Q(t)sin\omega_0 t \tag{5}$$

$$N_I(t) = N(t)cos\omega_0 t + \widehat{N}(t)sin\omega_0 t \tag{6}$$

$$N_Q(t) = -N(t)sin\omega_0 t + \widehat{N}(t)cos\omega_0 t \tag{7}$$

由(3)可知接收信号$V(t)$可写为：

$$V(t) = [N_I(t) + bcos\theta]cos\omega_0 t - \left[N_Q(t) + bsin\theta\right]sin\omega_0 t \tag{8}$$

其包络表达式为：

$$a(t) = \sqrt{(N_I(t) + bcos\theta)^2 + \left(N_Q(t) + bsin\theta\right)^2} \tag{9}$$

对于接收信号$V(t)$，其解析信号$V_a(t)$定义为：

$$V_a(t) = V(t) + j\hat{V}(t) \tag{10}$$

其中$\hat{V}(t)$为$V(t)$的希尔伯特变换，计算公式为：

$$\hat{V}(t) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{x(\tau)}{t-\tau}d\tau \tag{11}$$

解析信号的模值即为原信号的包络$a(t)$，即$a(t) = |V_a(t)|$。直接使用希尔伯特变换求解信号包络无需显式求解窄带噪声的正交分量$N_I(t)$和$N_Q(t)$，可以简化计算步骤。

### 2.3.2. 关键 MATLAB 函数

- randn：生成均值为 0、方差为 1 的高斯白噪声序列；
- filter：对信号进行数字滤波；
- std/var：分别计算信号的标准差和方差
- periodogram：采用周期图法计算功率谱密度

### 2.3.3. 核心代码

```
% 1. 蒙特卡洛仿真循环内（单次试次），生成接收信号（已在前述模块实现）
n_t = randn(1, N); % 生成新的高斯白噪声
n_t = filter(b_filt, a_filt, n_t); % 窄带滤波
n_t = n_t / std(n_t) * sqrt(app.sigma2); % 归一化方差
theta_k = rand * 2 * pi; % 随机相位
s_t_k = b * cos(2*pi*app.f0*t + theta_k); % 正弦信号
v_t = s_t_k + n_t; % 接收信号

% 2. 希尔伯特变换构建解析信号
v_analytic = hilbert(v_t);

% 3. 提取接收信号包络
envelope = abs(v_analytic);

% 4. 存储包络数据（用于后续概率密度分析）
envelope_store = [envelope_store, envelope];
app.envelope_data{i} = envelope_store; % 按信噪比分类存储
```

## 2.4. 检测阈值和漏警概率计算模块

### 2.4.1. 基本原理

无信号情况下，窄带高斯白噪声服从瑞利分布：

$$f_A(a) = \frac{a}{\sigma^2} e^{-\frac{a^2}{2\sigma^2}} \tag{12}$$

含有信号的包络服从广义瑞利（莱斯）分布：

$$f_A(a) = \frac{a}{\sigma^2} e^{-\frac{a^2+b^2}{2\sigma^2}} I_0\left(\frac{ab}{\sigma^2}\right) \tag{13}$$

其中$a > 0$。当信噪比很小时，上述分布趋近于瑞利分布。当信噪比很大时，上述分布近似为高斯分布，即：

$$f_A(a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(a-b)^2}{2\sigma^2}} \tag{14}$$

不同信噪比下的信号包络概率分布如图 6 所示：

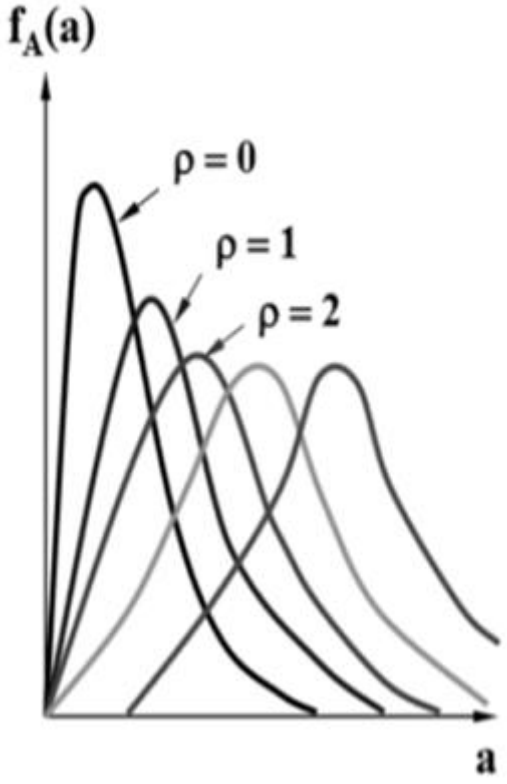


图 6：广义瑞利分布的概率密度函数

为了求解判断阈值$V_T$，用 ksdensity 画出有信号与无信号条件下的一维概率密度，先确定$V_T$的大致范围。然后在此范围下，求瑞利分布和不同信噪比的信号分布函数的交点，即为每一信噪比条件下确定$V_T$。注意在求解过程中设定范围逐点扫描求交点，多个交点选择最合理者，并且每种信噪比下搜索范围不同。

在确定判断阈值后，接下来可以对漏警概率进行求解。理论值：对上述画出的有信

号条件下的一维概率密度曲线，求小于$V_T$时的曲线面积，再求有信号全曲线下的面积，两者的比值即为漏警率。实际值：重复生成 1000 次接收信号$V(t)$，通过 MATLAB 计算包络，并与阈值$V_T$比较，小于阈值说明检测不到信号，大于阈值说明检测到信号，计算 1000 次仿真中信号检测的漏警概率的平均值（每次的漏警概率的平均）。

### 2.4.2. 关键 MATLAB 函数

- besseli：计算修正贝塞尔函数；
- trapz：梯形数值积分函数；
- ksdensity：计算经验概率密度函数

### 2.4.3. 核心代码

```
% 1. 定义幅度轴（用于概率密度计算）
a_axis = 0:0.001:30;
sigma = sqrt(app.sigma2);
b = app.res_b(i); % 第 i 个信噪比对应的信号幅度

% 2. 计算理论概率密度函数
% 瑞利分布（H0：无信号）
pdf_H0 = (a_axis / app.sigma2) .* exp(-a_axis.^2 / (2*app.sigma2));
% 莱斯分布（H1：有信号）
pdf_H1 = (a_axis / app.sigma2) .* exp(-
(a_axis.^2 + b^2)/(2*app.sigma2)) .* besseli(0, a_axis * b / app.sigma2);

% 3. 求解检测阈值 V_T（找两分布差值最小点）
diff_pdf = abs(pdf_H0 - pdf_H1);
valid_idx = find(a_axis > 0 & a_axis < b+3); % 限定有效搜索范围，避免无效值
[~, min_idx_local] = min(diff_pdf(valid_idx));
idx_Vt = valid_idx(min_idx_local);
Vt = a_axis(idx_Vt);
app.res_Vt(i) = Vt; % 存储阈值

% 4. 计算理论漏警率（梯形积分法）
mask_integral = a_axis <= Vt; % 积分区间[0, Vt]
Pm_theory = trapz(a_axis(mask_integral), pdf_H1(mask_integral));
app.res_Pm_theo(i) = Pm_theory;

% 5. 1000 次蒙特卡洛仿真计算实际漏警率
num_trials = 1000;
Pm_act = 0;
for k = 1:num_trials
    % 生成单次试次的接收信号与包络（略，同模块 4）
    n_t = randn(1, N);
    n_t = filter(b_filt, a_filt, n_t);
```

```
32.      n_t = n_t / std(n_t) * sigma;
33.      theta_k = rand * 2 * pi;
34.      s_t_k = b * cos(2*pi*app.f0*t + theta_k);
35.      v_t = s_t_k + n_t;
36.      v_analytic = hilbert(v_t);
37.      envelope = abs(v_analytic);
38.
39.      % 统计单次试次的漏警比例
40.      Pm_act = Pm_act + sum(envelope < Vt)/length(envelope);
41.  end
42.
43.  % 6. 计算平均实际漏警率
44.  app.res_Pm_act(i) = Pm_act / num_trials;
45.
46.  % 7. 计算经验概率密度（用于 PDF 可视化）
47.  [f_est, xi_est] = ksdensity(app.envelope_data{i}, a_axis);
```

## 三、 系统功能测试

为了验证上述设计的包络检测器是否可以正常工作，需要对其进行功能测试。在这里使用实验讲义所给参数对该系统的包络检测能力和漏警概率计算进行了测试，测试结果如下所示。

### 1. 窄带滤波器设计

首先设计一个中心频率为 80000Hz，带宽为 400Hz 的窄带滤波器，滤波器的幅频响应如图 7 所示：

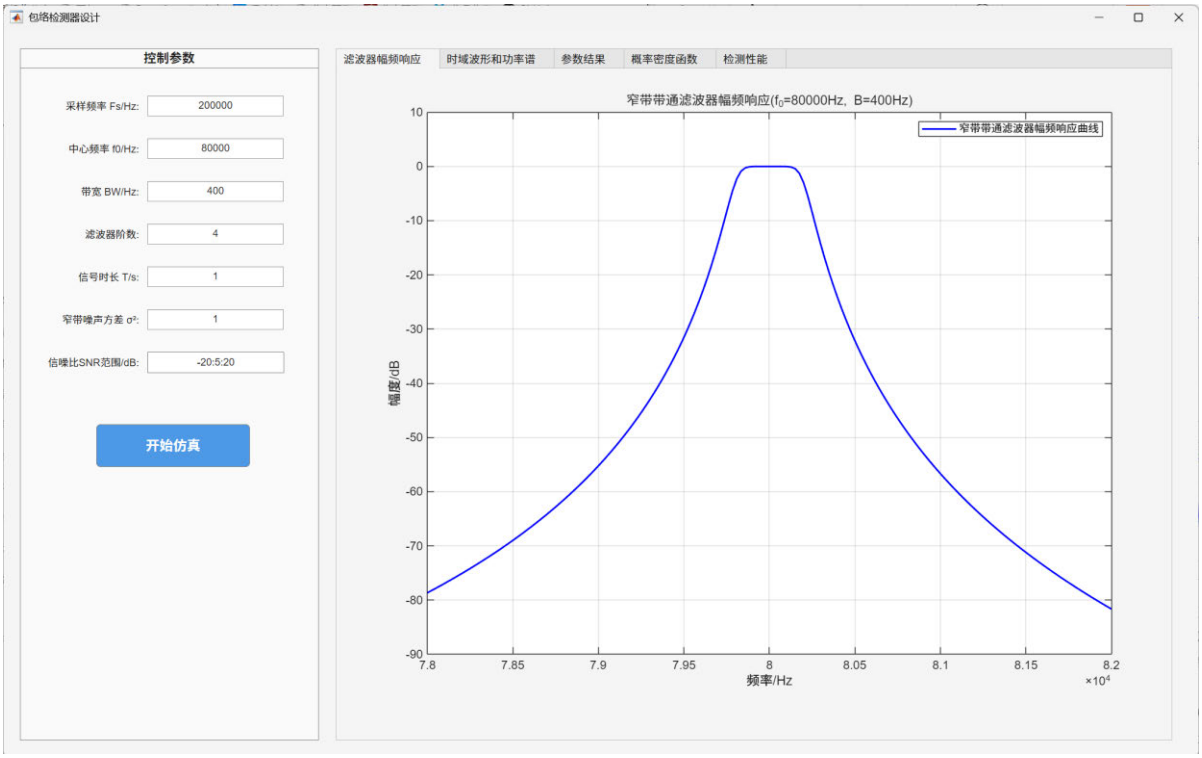


图 7：滤波器幅频响应

为了观察方便，滤波器的幅频响应只绘制中心频率左右 4000Hz 范围内的曲线。观察上图可以发现，设计的带通滤波器在以 80000Hz 为中心的 400Hz 带宽内是很平坦的通带，通带之外迅速衰减。因此可以很好地滤除带外噪声，得到窄带高斯白噪声信号。

## 2. 噪声和信号时域波形与功率谱

### 2.1. 窄带高斯白噪声

对生成的高斯白噪声进行带通滤波和归一化操作后，即可得到窄带高斯白噪声，选择接收信号为$N(t)$，时域波形和功率谱如图 8 所示：

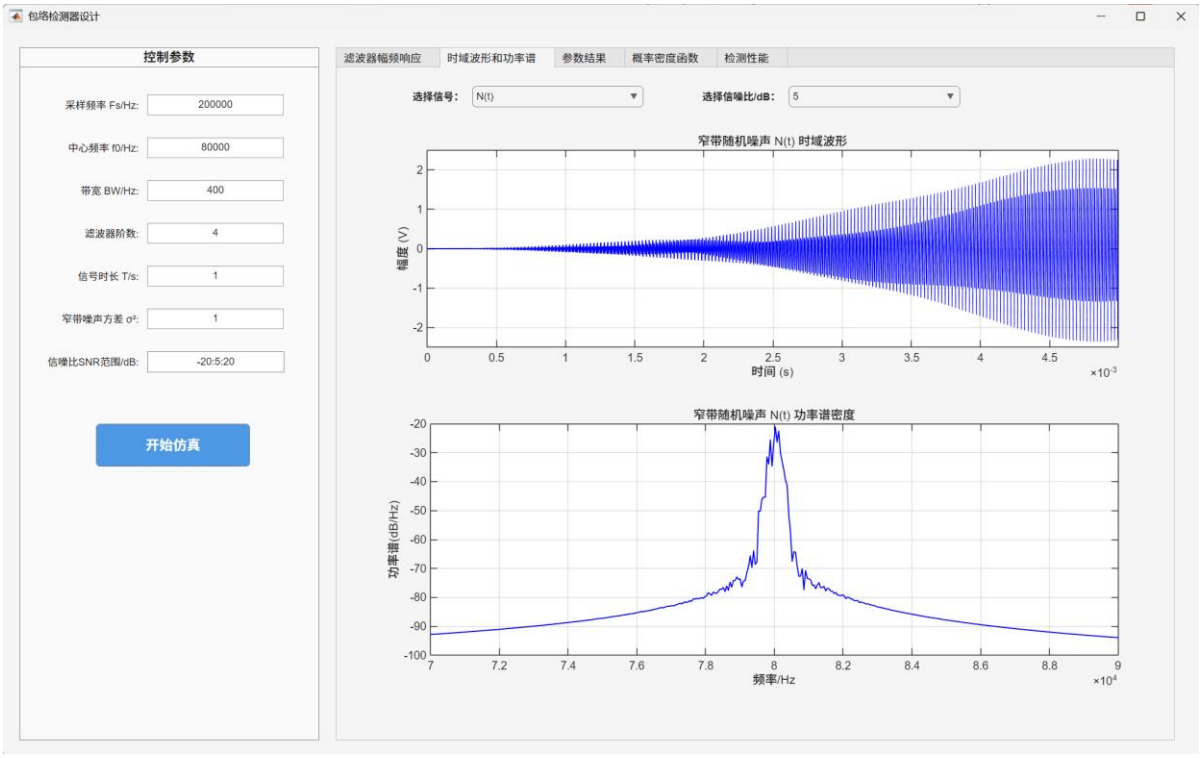


图 8：窄带高斯白噪声时域波形和功率谱

观察上图可以看出，窄带高斯白噪声的功率谱密度集中在 80000Hz 左右，这与其窄带特性是相符的，从而验证窄带高斯白噪声生成的正确性。

### 2.2. 接收信号

将系统切换到时域波形和功率谱标签页，选择接收信号为接收信号$V(t)$，信噪比为 5dB，时域波形和功率谱如图 9 所示：

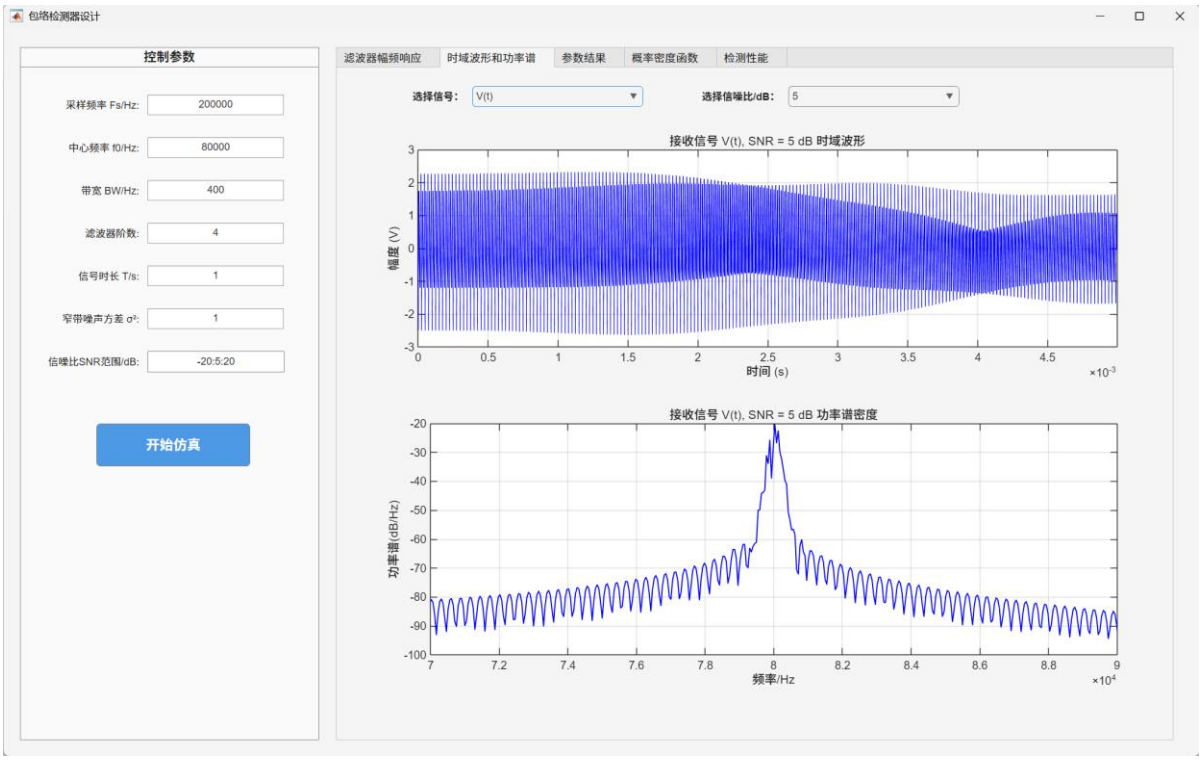


图 9：接收信号时域波形和功率谱

在低信噪比（-20dB）和高信噪比（20dB）条件下，接收信号的时域波形和功率谱分别如图 10、11 所示：

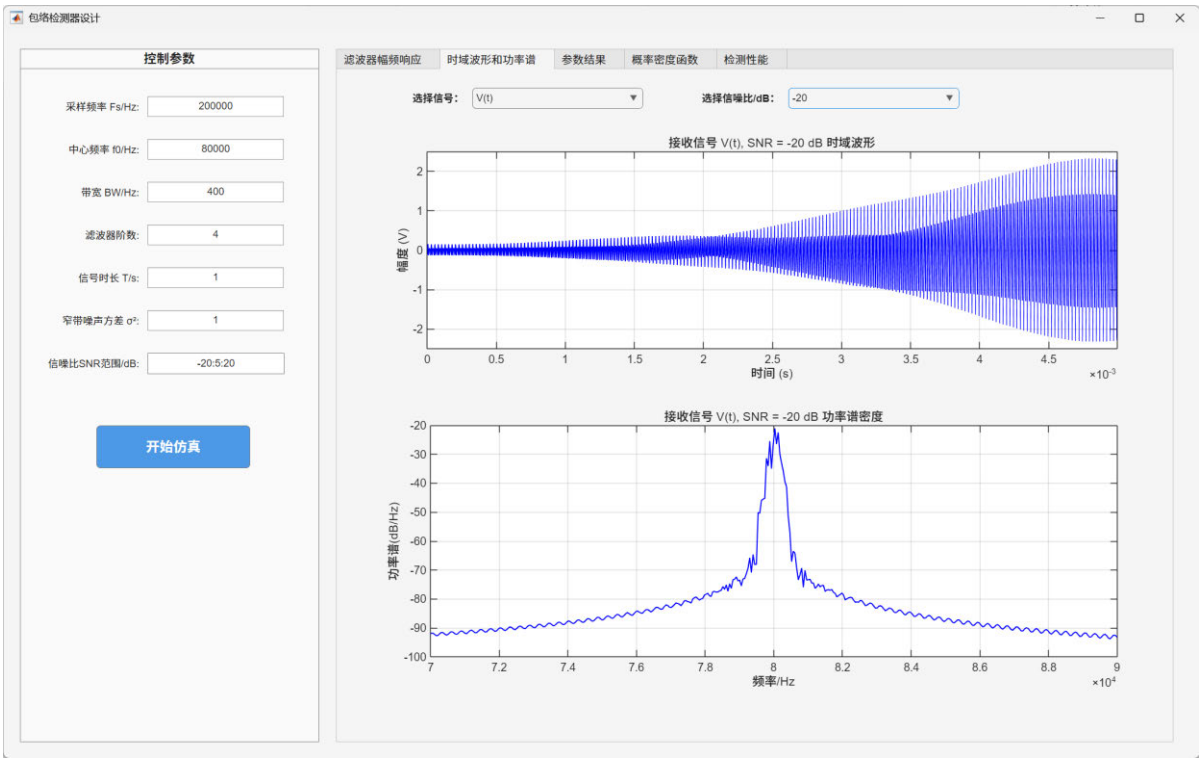


图 10：低信噪比接收信号时域波形和功率谱

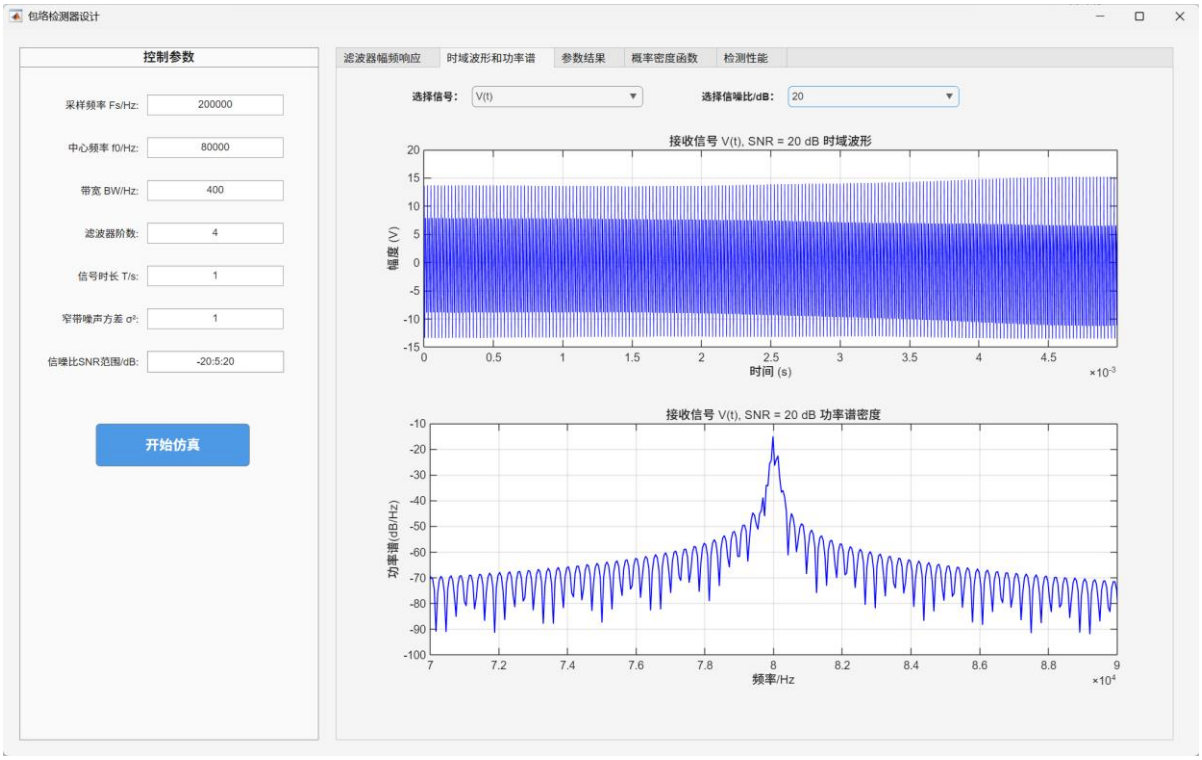


图 11：高信噪比接收信号时域波形和功率谱

观察图 9、10 和 11 可知，当接收信号的信噪比较低时，其时域波形和功率谱和窄带高斯白噪声较为接近。当接收信号信噪比较高时，其时域波形和功率谱与目标无线电信号较为接近。这一现象与实际相符，进一步验证了接收信号构造的正确性。与此同时，接收信号的功率谱密度均集中于 80000Hz 左右，这与窄带信号的特点相符。

### 3. 参数结果变化曲线

将系统切换到参数结果标签页，可查看窄带高斯白噪声方差、信号幅度以及检测阈值随信噪比变化曲线，如图 12 所示：

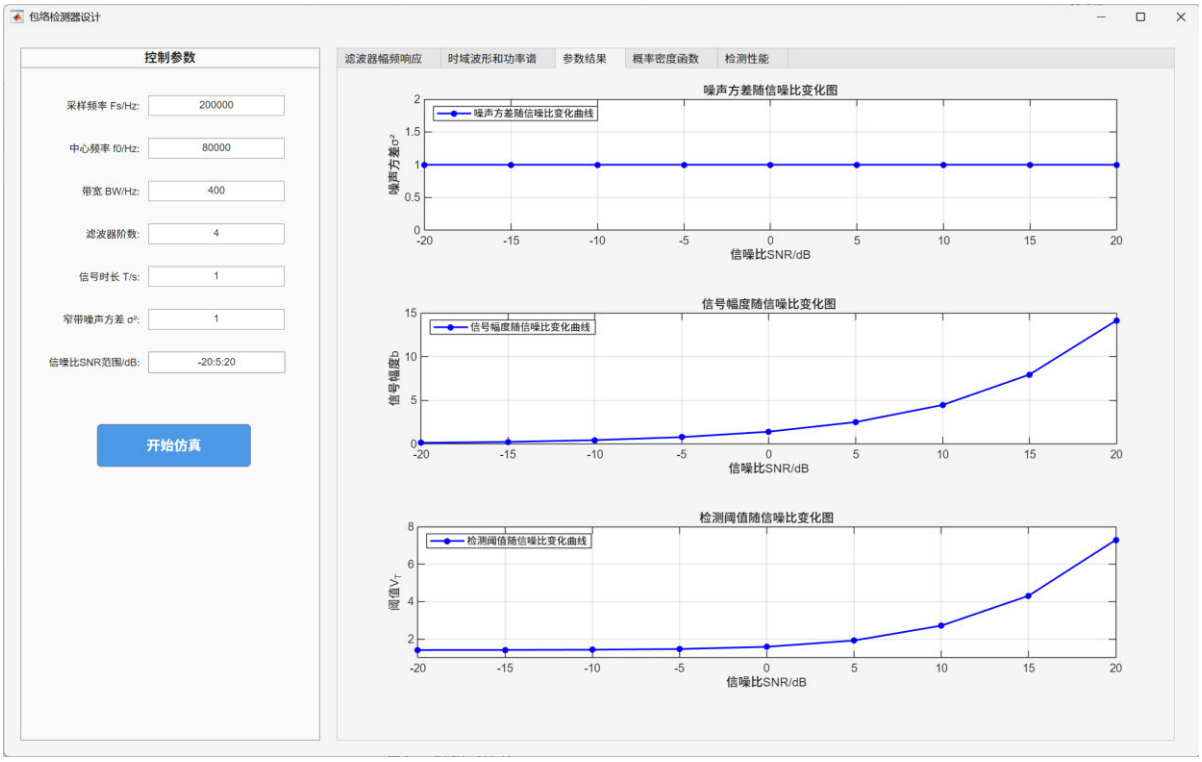


图 12：参数结果变化曲线

观察上图可以看出，窄带高斯白噪声的方差随信噪比几乎不变。这是因为在实验中高斯白噪声经带通滤波器滤波后将其方差归一化至用户设定值，而图 12 中的方差与左侧设定结果相符，因此这是正常现象。信号幅度和检测阈值随信噪比的升高不断增大，这也是合理的实验现象。由信噪比计算公式可知信噪比越高，信号幅度就越大。接收信号的概率密度函数曲线会整体右移，与瑞利分布曲线的交点也会发生右移，因此检测阈值也会不断增大。

**4. 概率密度函数**

将系统切换到概率密度函数标签页，设定接收信号信噪比为 5dB，可查看窄带高斯白噪声、接收信号包络的理论概率密度函数曲线以及接收信号的实际概率密度函数曲线，并标出检测阈值$V_T$，如图 13 所示：

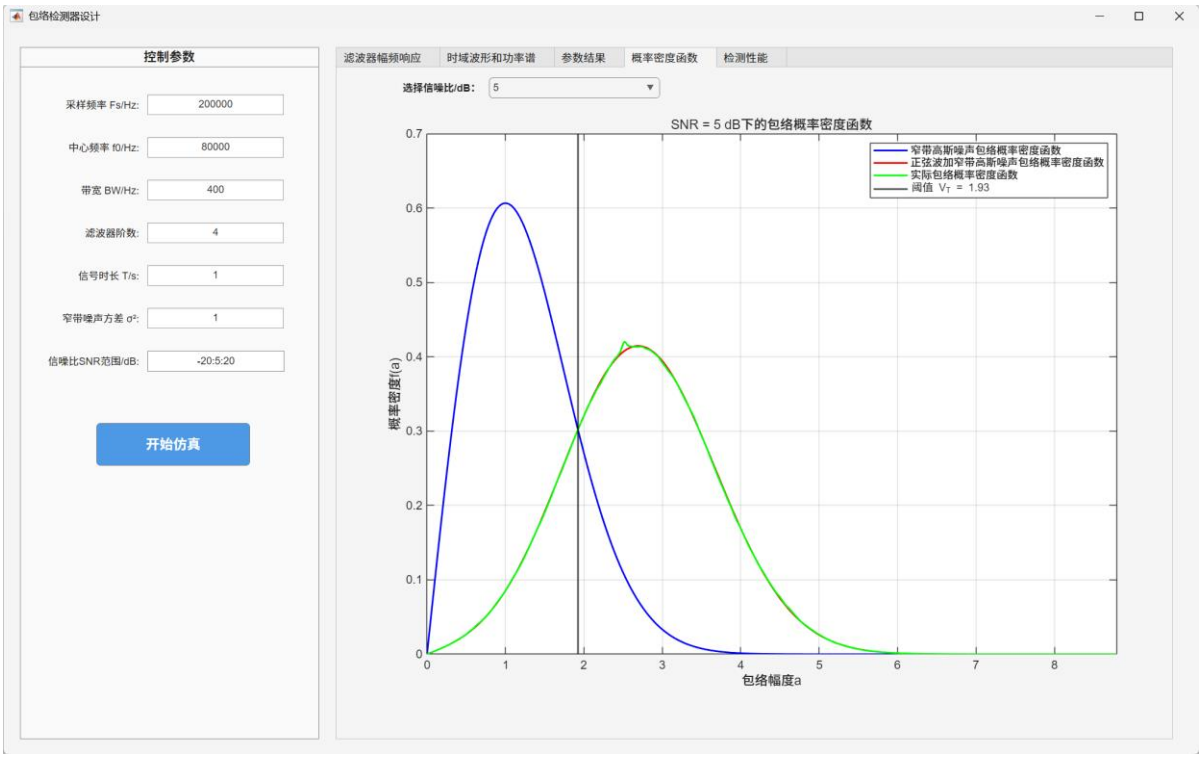


图 13：概率密度函数曲线

在低信噪比（-20dB）和高信噪比（20dB）条件下，接收信号包络的理论和实际概率密度函数曲线如图 13、14 所示：

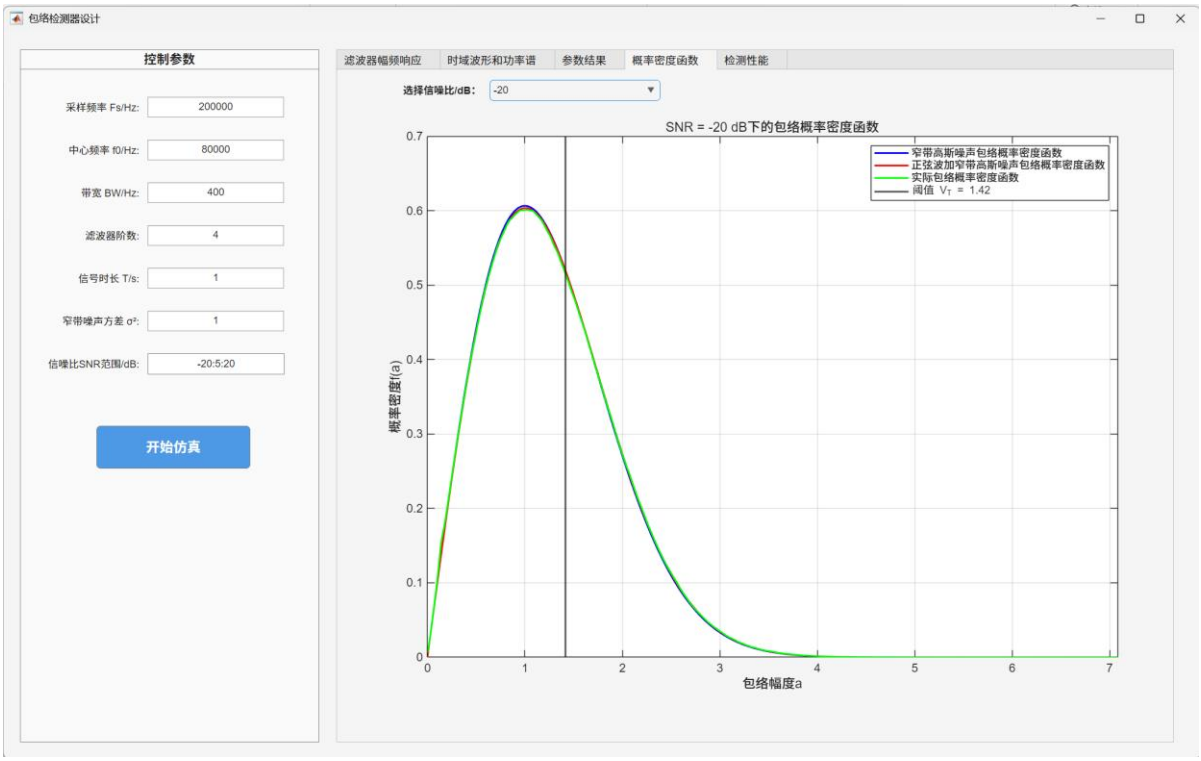


图 14：低信噪比概率密度函数曲线

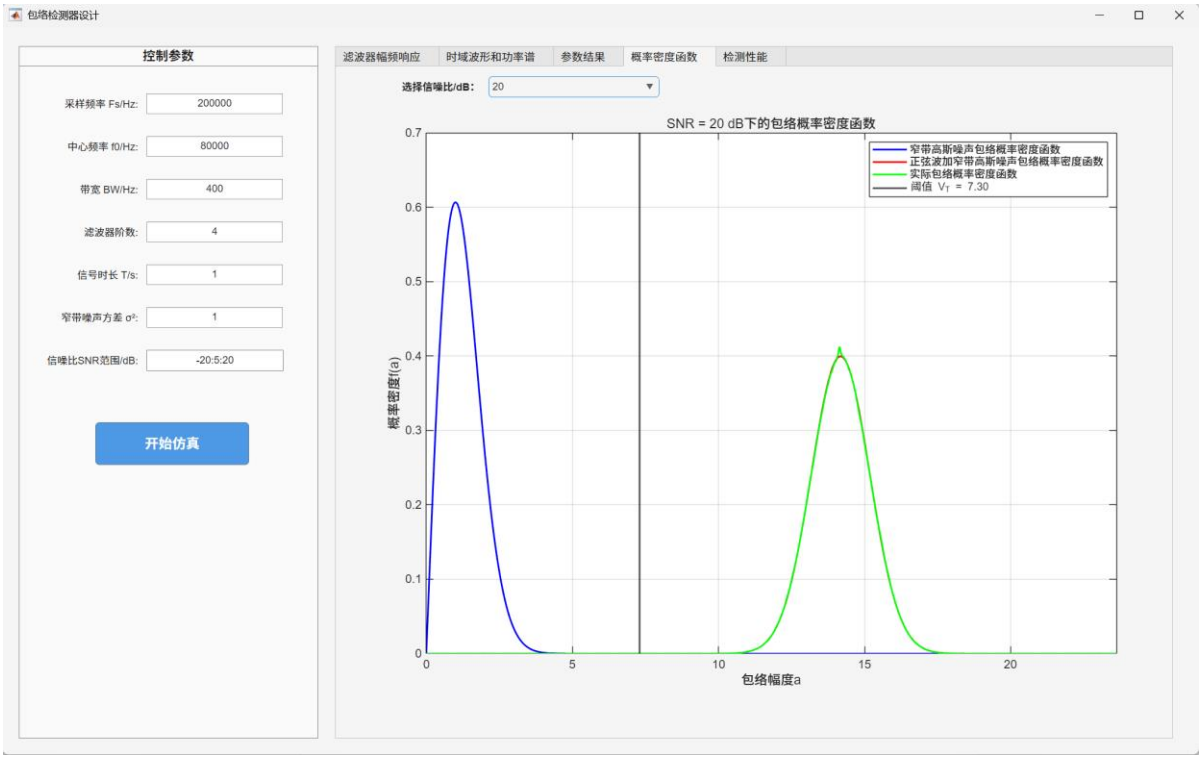


图 15：高信噪比概率密度函数曲线

观察图 13、14 和 15 可知，窄带高斯白噪声和接收信号的包络分别呈瑞利分布和广义瑞利分布，并且接收信号包络的实际概率密度函数与理论值非常接近，这进一步验证了仿真结果的正确性。随着信噪比的增大，接收信号包络的概率密度函数逐渐从瑞利分布变为高斯分布，与广义瑞利分布的结果相同。在这三种情况下，检测阈值$V_T$均恰好为接收信号概率密度函数与瑞利分布的交点，与题目要求相同，因此可据此进行漏警概率的求解。

## 5. 漏警概率曲线

将系统切换到检测性能标签页，可查看理论漏警概率和实际漏警概率随信噪比变化曲线，如图 16 所示：

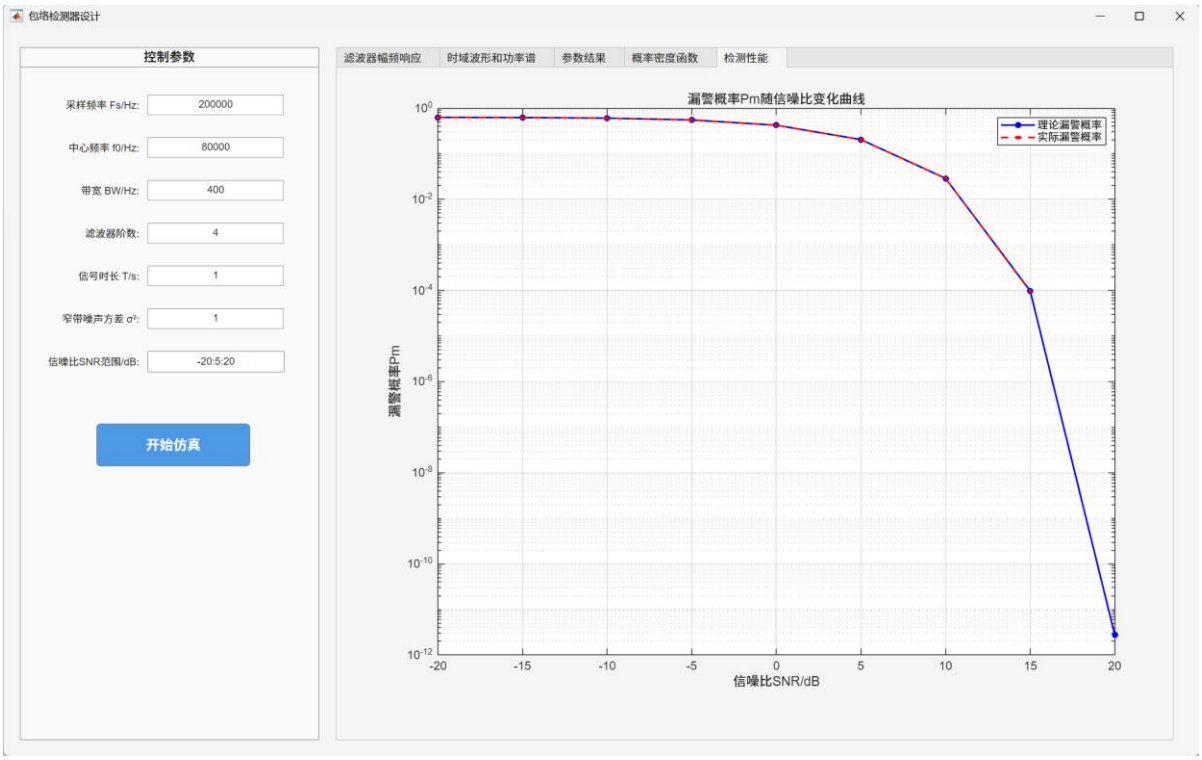


图 16：漏警概率曲线

为方便观察，漏警概率曲线绘制成半对数坐标的形式，以 10 的负幂次沿着 y 轴往下扩展。观察上图可以发现，实际仿真漏警概率曲线和理论值紧密贴合，验证了仿真结果的正确性。随着信噪比增大，漏警概率迅速降低并无限趋近于 0。因此在通信系统中若要有效识别目标信号，必须保证足够大的信噪比。

## 四、 总结与体会

通过本次包络检测器设计实验，我收获了扎实的随机信号分析知识与 MATLAB App 开发实践经验，对包络提取和漏警概率计算的核心原理有了更为深刻的理解。实验过程中，我不仅深入掌握了窄带随机信号、功率谱密度、希尔伯特变换等核心理论，还熟练运用 MATLAB 实现了窄带滤波器设计、信号与噪声生成、包络提取、检测阈值求解、蒙特卡洛仿真等关键算法，成功开发出功能完整、交互友好的包络检测器模块化 GUI 系统。

其中，实验中我遇到的最大问题是不同信噪比下检测阈值求解不准确，导致漏警率理论值与实际值均计算错误。初期设计时，我统一设定一个较大的搜索范围。由于瑞利分布和莱斯分布的概率密度函数在自变量较大时会无限趋近于 0，并且计算机计算这两种分布时只能取一系列离散点，从而无法取到真正的交点。因此如果通过逐点扫描差值最小的方法寻找交点，最终结果可能是远大于 0 的搜索范围最大值，从而错过真正的交点，导致漏警率计算错误。为解决这一问题，我反复调试代码并结合理论推导，

最终采用根据信号幅度 b 动态调整搜索范围的方案，同时优化 ksdensity 函数的采样精度，有效提升了阈值求解的准确性，使理论与实际漏警率曲线紧密贴合。

本次实验让我深刻体会到理论与实践结合的重要意义。之前在课堂上学习的概率分布、希尔伯特变换、随机信号分析等理论知识，通过这次包络检测器系统开发得到了全面应用。实验中，我不仅编程实现窄带高斯白噪声生成、随机信号包络提取、检测阈值判定和漏警概率计算等功能，验证了不同信噪比下包络检测的性能，还通过量化漏警率、阈值等指标，进一步加深了对信号检测中噪声抑制和阈值优化等核心特性的理解，真切体会到该技术在窄带通信、雷达信号检测等领域的应用价值。

最后，我要衷心感谢[redacted]的悉心指导。[redacted]在课堂上为我们耐心讲解了窄带高斯白噪声的构造、瑞利/莱斯分布特性、希尔伯特变换应用、蒙特卡洛仿真设计等算法设计方法，帮助我顺利完成了这次包络检测器的设计。通过本次实验，我不仅夯实了数字信号处理与随机信号分析的基础理论，积累了算法调试与 MATLAB App 开发的实践经验，更培养了科学严谨的实验态度和自主解决问题的思维能力。这些收获将为我后续深入学习高阶信号检测算法、开展通信系统相关科研实践奠定坚实基础，对未来的专业学习与发展产生积极而深远的影响。