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

基于 MATLAB APP 设计的信号的产生、分析与处理系统 1
一、 总体方案设计 1
1. 需求分析 1
2. 设计目标 1
3. 设计方案 2
3.1. 系统总体结构 2
3.2. 系统功能模块规划与核心功能实现 2
3.2.1. UI 控制模块 2
3.2.2. 信号生成模块 3
3.2.3. 谱分析模块 3
3.2.4. 滤波器设计模块 3
3.2.5. 信号滤波模块 4
3.2.6. 结果绘制模块 4
3.3. 各模块之间的关系 4
4. 开发环境 5
二、 系统的设计与实现 6
1. 软件系统界面设计 6
1.1. 标签页 1：典型信号生成与频谱分析 6
1.1.1. 左侧参数控制区 6
1.1.2. 右侧绘图展示区 7
1.2. 标签页 2：数字滤波器设计 7
1.2.1. 左侧参数控制区 8
1.2.2. 右侧绘图展示 9
1.3. 界面设计特点 9
2. 核心模块的具体实现方法 9
2.1. 信号产生模块 9
2.1.1. 基本原理 9
2.1.2. 关键 MATLAB 函数 10
2.1.3. 核心代码 10
2.2. 谱分析模块 15
2.2.1. 基本原理 15

2.2.2. 关键 MATLAB 函数 15
2.2.3. 核心代码 15
2.3. 滤波器设计模块 19
2.3.1. 基本原理 19
2.3.2. 关键 MATLAB 函数 20
2.3.3. 核心代码 20
2.4. 滤波处理模块 24
2.4.1. 基本原理 24
2.4.2. 关键 MATLAB 函数 24
2.4.3. 核心代码 24
三、 系统功能测试 27
1. 典型信号生成和频谱分析 27
1.1. 正弦波信号 27
1.2. 方波信号 28
1.3. 混合正弦波信号 29
1.4. 加载文件信号 31
1.5. 信号加噪处理 32
2. 滤波器设计 34
2.1. 滤波器功能测试 35
2.1.1. 低通滤波器 35
2.1.2. 高通滤波器 37
2.1.3. 带通滤波器 38
2.1.4. 带阻滤波器 40
2.2. 典型信号滤波分析 42
2.2.1. 有噪正弦波信号的降噪处理 42
2.2.2. 方波信号滤波分析 45
四、 总结与体会 47

基于 MATLAB APP 设计的信号的产生、分析与处理系统

一、 总体方案设计

1. 需求分析

本项目面向常见典型信号的分析处理需求，利用 MATLAB 中 APP 设计实现信号波形产生、频谱分析、数字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：

- 信号生成需求：支持正弦波、方波、混合正弦波等常见典型信号产生以及外部信号文件的导入，同时提供高斯白噪声和均匀白噪声的添加功能。
- 参数设置需求：对不同种类的典型信号支持信号基本参数的自定义设置，其中混合正弦波支持任意多个不同频率分量对应的参数设置。支持信号加噪和不加噪两种模式，并且噪声的均值和方差可由用户设置。
- 频谱分析需求：支持对产生的信号进行谱分析，其中对确定性信号进行频谱分析，对加噪后的随机信号进行功率谱分析。
- 滤波器设计需求：支持低通、高通、带通和带阻等滤波器的设计，滤波器的基本指标参数可由用户设计。支持 IIR 和 FIR 两种不同类型的数字滤波器设计，每种类型的数字滤波器支持选用不同的设计方法。
- 信号处理需求：生成的信号可以直接传递到滤波器设计模块进行滤波处理和分析，方便进行实时的信号处理。
- 错误提示需求：具备完善的操作有效性检查，当用户的操作或输入参数不符合规范时，系统会提示错误信息并告诉用户正确的操作方法，避免程序运行出现异常状况。
- 可视化分析需求：实现原始信号时域波形、频谱或功率谱的实时绘制，滤波器幅频和相频响应可视化呈现，滤波后信号时域波形与频谱的对比展示，支持峰值标注与坐标轴刻度的自动优化。
- 交互操作需求：提供简洁直观的图形用户界面，界面布局清晰明了。支持信号类型、滤波器类型的下拉切换，参数控件随功能选择自动显隐。操作方法简单，符合用户认知，不需要编写代码，在图形窗口中可以完成所有操作。

2. 设计目标

- 开发一款可视化的 MATLAB App 工具，实现典型信号生成、谱分析、滤波器设计、信号滤波处理的一体化信号处理实验平台，满足基本的信号处理需求。
- 覆盖信号处理基础核心知识点，包括典型信号的时域特性与频域分析、数字滤波器

的设计原理与实现方法、噪声对信号的影响及滤波降噪效果验证。

- 实现灵活的参数设置与结果的可视化分析，帮助使用者快速理解信号与滤波器的内在联系，降低信号处理实验的入门门槛。
- 构建高鲁棒性、易操作的交互界面，确保不同层次的使用者都能顺利完成基本的信号分析和处理，同时提供清晰的代码结构与注释，便于后续功能扩展与二次开发。
- 达成课程设计的教学目标，使使用者掌握 MATLAB App Designer 的开发流程，理解信号处理的核心算法，提升工程实践与问题解决能力。

3. 设计方案

3.1. 系统总体结构

本系统采用分层式架构和模块化设计的思想，基于 MATLAB App Designer 工具箱实现。系统整体分为三层，自上而下依次为交互界面层、功能模块层、数据存储层，每一层的实现功能阐述如下：

- 交互界面层：作为用户与系统的交互入口，负责接收用户操作与信号参数输入，展示信号处理结果并将其可视化呈现，所有 UI 组件统一管理与初始化。
- 功能模块层：系统的核心算法实现层，用于封装各类信号处理算法。功能模块响应交互界面层的操作触发，完成数据处理与逻辑运算，并将结果反馈至交互界面层进行展示。
- 数据存储层：负责存储系统运行过程中的关键数据与状态信息，为功能模块层提供数据支撑，实现数据在不同模块间的共享与传递。

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 UI 控制模块、信号生成模块、谱分析模块、滤波器设计模块、信号滤波模块和结果绘制模块。各模块独立封装、功能明确，具体实现功能阐述如下：

3.2.1. UI 控制模块

负责 UI 组件的初始化、显隐控制、状态更新，确保在用户的不同操作下显示正确的图形界面，具体实现功能如下：

- 初始化所有 UI 组件的位置、默认参数与显隐状态，构建统一的图形界面。
- 在用户通过下拉列表切换信号类型或滤波器类型时，首先隐藏所有非公共控件，根据用户选择的信号类型、滤波器类型，自动显示对应功能的控件，隐藏无关控件，简化界面操作。
- 封装 UI 组件的属性设置逻辑，减少代码冗余，提升界面维护的便捷性。

3.2.2. 信号生成模块

该模块负责根据用户设置的参数生成典型信号，加载外部信号文件，添加指定类型的噪声，并存储生成的信号与采样频率信息，具体实现功能如下：

- 支持正弦波、方波、混合正弦波的自定义生成，根据用户输入的信号幅度、频率、相位、占空比等参数，生成符合要求的时域信号。
- 支持.mat 格式外部信号文件的导入，自动提取信号数据，校验文件有效性，更新文件加载状态标志。
- 支持高斯白噪声、均匀白噪声的添加，根据用户输入噪声的均值与方差生成噪声序列，并与原始信号叠加，生成有噪信号。
- 具备完善的参数有效性校验，对用户输入的无效参数（如采样、信号频率≤0，信号时长<0，方差<0、混合正弦波信号参数数量不匹配、文件加载失败等）弹出错误提示，确保信号的正确生成。
- 将生成的原始信号、有噪信号与采样频率存储至系统私有属性，为后续频谱分析与滤波处理提供数据支撑。
- 触发结果绘制模块，完成生成信号的时域波形绘制。

3.2.3. 谱分析模块

该模块负责对信号生成模块生成的信号进行谱分析，实现 FFT 频谱或功率谱的计算与绘制，标注主要频率分量，并将结果可视化呈现，具体实现功能如下：

- 对无噪的确定性信号采用 FFT 算法进行频谱分析，计算信号的幅频特性，绘制 FFT 频谱图。
- 对有噪的随机信号计算功率谱密度，转换为 dB 刻度绘制功率谱图，抑制噪声对频域分析的干扰，提升频率分量的辨识度。
- 触发结果绘制模块，完成信号的频谱图或功率谱图绘制。

3.2.4. 滤波器设计模块

该模块负责根据用户设置的滤波器类型和参数，设计符合要求的 IIR 或 FIR 滤波器，并计算滤波器系统函数的分子和分母的系数，具体实现功能如下：

- 根据用户输入的滤波器参数，完成频率参数的归一化处理。
- 支持 IIR 滤波器设计：涵盖巴特沃斯型、切比雪夫 I 型、切比雪夫 II 型、椭圆型四种类型，完成滤波器阶数计算与系数求解。
- 支持 FIR 滤波器设计：涵盖矩形窗、汉宁窗、汉明窗、布莱克曼窗、凯瑟窗五种窗函数，通过计算过渡带宽确定滤波器阶数，采用 fir1 函数完成滤波器系数

求解，FIR 滤波器分母系数固定为 1。

- 具备完善的参数校验与错误捕获功能，对用户输入的无效通带和阻带截止频率进行提示，捕获滤波器设计过程中的异常并反馈给用户。
- 返回滤波器的分子系数 b 与分母系数 a，为后续信号滤波处理与滤波器频率响应可视化分析提供数据支撑。

3.2.5. 信号滤波模块

该模块负责采用设计好的滤波器对原始信号进行滤波处理，去除噪声或无用频率分量，生成滤波后的信号，具体实现功能如下：

- 检验原始信号与采样频率是否有效，确保滤波前已在信号生成模块生成或加载信号。
- 调用滤波器设计模块，获取有效的滤波器系数 b 与 a。
- 对生成的原始信号进行线性滤波处理，生成滤波后信号。
- 触发结果绘制模块，完成滤波后信号波形、频谱以及滤波器频率响应的可视化展示。

3.2.6. 结果绘制模块

该模块负责系统所有图形化结果的绘制，包括原始信号时域波形、滤波器响应、滤波后信号时域波形与频谱，具体实现功能如下：

- 绘制信号生成模块产生信号的时域波形，添加标题、坐标轴标注、网格与边框，自适应调整坐标轴范围。
- 绘制信号的频谱图或 dB 刻度的功率谱图，自动检测信号前 3 个主要频率峰值，进行红色标记与频率数值标注。支持坐标轴的自动优化与格式化，横坐标频率范围限制在 0~fs/2，其中 fs 为用户设置的采样频率。纵坐标根据数据范围自适应调整，提升图形的可读性。
- 绘制滤波器设计模块中所设计滤波器的 dB 刻度的幅频响应与相频响应，相频响应添加 0、±π 参考线与格式化刻度标注，直观展示滤波器的频率特性。
- 绘制滤波后信号的时域波形与 FFT 频谱图，与原始信号的时频域结果形成对比，清晰展示滤波降噪或频率选择的效果。
- 统一图形绘制样式，确保界面图形的美观性与一致性。

3.3. 各模块之间的关系

各功能模块相互协作、数据互通，形成闭环的信号处理流程，模块间的核心交互关系如下：

1. 启动流程：用户打开系统后，UI 控制模块初始化所有 UI 组件，完成界面搭建，数据存储层初始化私有属性，并设置所有输入参数为默认值。
2. 信号生成和谱分析流程：用户通过交互界面层输入参数，点击生成信号按钮进入信号生成模块，信号生成模块完成参数校验、信号生成、噪声添加并通过结果绘制模块绘制信号的时域波形。接下来自动进入谱分析模块计算生成信号的 FFT 频谱或功率谱，再利用结果绘制模块进行可视化展示，最后在数据存储层存储生成信号和采样频率为后续的滤波处理做好准备。
3. 滤波器设计和信号滤波处理流程：用户通过交互界面层设置滤波器类型和参数，然后进入滤波器设计模块完成滤波器的设计和对应滤波器系数的求解，再根据数据存储层存储的原始信号在信号滤波模块完成滤波处理，最后利用结果绘制模块绘制滤波器频率响应与滤波后信号时频域图形，并将其反馈至交互界面层展示。

模块间的依赖关系：

- 谱分析模块、信号滤波模块和结果绘制模块均依赖信号生成模块获取原始信号，信号生成模块是本系统的核心基础模块。
- 信号滤波模块还依赖滤波器设计模块获取对应的滤波器系数。
- 结果绘制模块依赖信号生成模块、频谱分析模块、滤波器设计模块、信号滤波模块获取各类结果数据。
- UI 控制模块为所有功能模块提供交互支撑，所有功能模块的结果最终通过 UI 控制模块反馈至交互界面层。

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、fir1、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，采用左侧参数控制面板+右侧可视化结果展示的经典布局，通过标签页的切换可分别选择“典型信号生成与频谱分析”和“数字滤波器设计”这两大功能模块。软件系统整体界面简洁直观，操作流程清晰，可以完成基本的信号分析处理的功能。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：典型信号生成与频谱分析

标签页 1 实现典型信号波形产生和谱分析的功能，布局分为左侧参数控制区和右侧的绘图展示区两大部分，如图 1 所示：

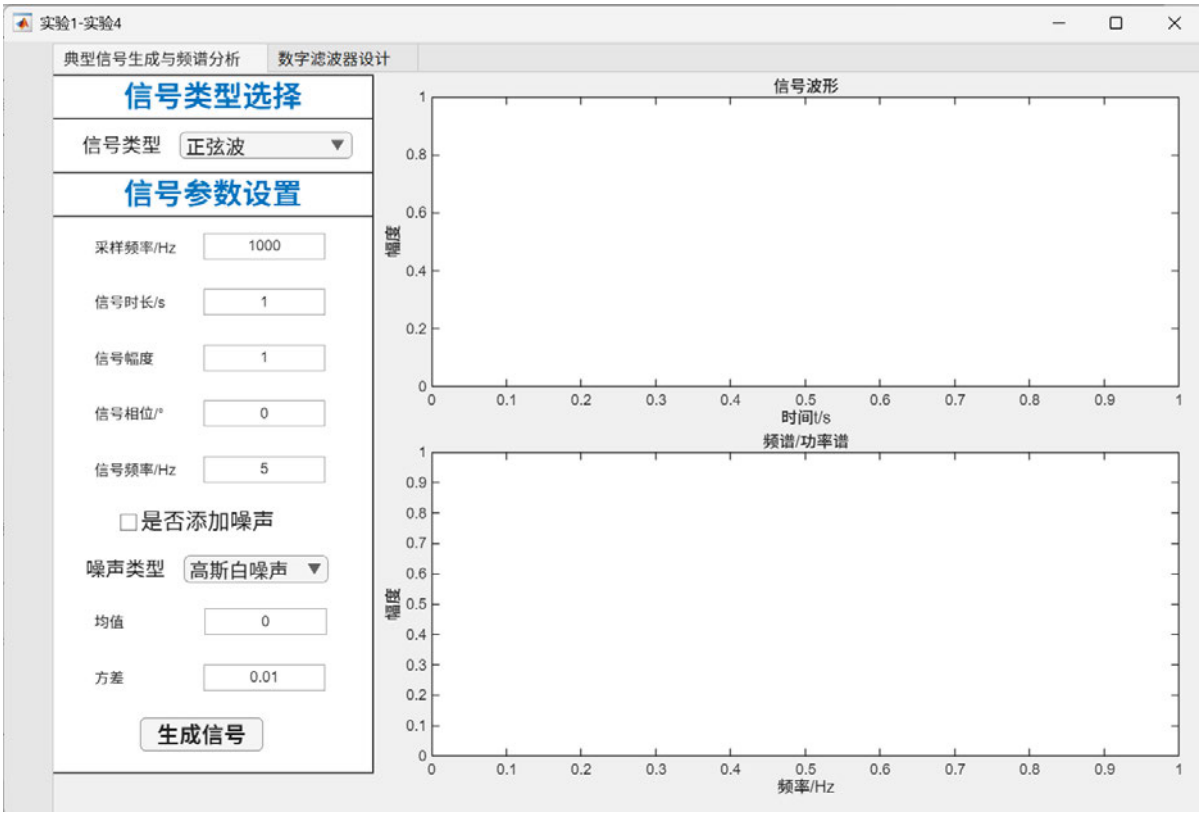

图 1：标签页 1 初始化界面

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. 左侧参数控制区

标签页 1 左侧从上到下分别包含两个面板，依次为信号类型选择和信号参数设置面板。信号类型选择面板包含一个信号类型下拉框，默认选中正弦波，支持正弦波、方波、混合正弦波、加载文件信号等多种信号类型的切换。信号参数设置面板包含生成信号的参数，其中采样频率（默认值 1000Hz）为所有生成信号的公共参数，信号时长（默认值 1s）是除了加载文件信号之外所有生成信号的公共参数，各种生成信号又

有各自的专用参数：

- 正弦波：信号幅度（默认值 1）、信号相位（默认值0°）、信号频率（默认值 5Hz），所有参数以数值的形式输入。
- 方波：信号幅度（默认值 1）、占空比（默认值 50%）、信号频率（默认值 5Hz），所有参数以数值的形式输入。
- 混合正弦波：信号幅度（默认值 1,1）、信号相位（默认值0°, 0°）、信号频率（默认值 5Hz,150Hz），所有参数以逗号分隔的文本输入，经后台程序处理为多个频率分量依次对应的参数。
- 加载文件信号：“选择文件”按钮，注意信号时长由文件直接决定，无需额外输入。

注意，专用参数编辑字段的显隐由选中的信号类型决定，当选中某一种生成信号时，其对应专用参数的编辑字段会显示出来，不需要的专用参数的编辑字段会隐藏。不同生成信号的系统界面和图 1 相比会有所不同，为避免过于冗余此处不再展示，相关界面可查看后续系统功能测试部分。除此之外，所有生成信号均可通过“是否添加噪声”复选框决定是否进行加噪处理（默认不加噪），选中后可通过“噪声类型”下拉框选择所加噪声为高斯白噪声还是均匀白噪声（默认为高斯白噪声），同时可以对噪声的均值（默认值 0）和方差（默认值 0.01）进行设置。

完成信号类型选择和信号参数设置后，可以点击底部的“生成信号”按钮生成对应的信号并在右侧绘图展示区绘制相应的图像。

1.1.2. 右侧绘图展示区

标签页 1 右侧包含两个垂直排列的坐标区。其中上侧为信号波形绘图区，用于绘制生成信号的时域波形。下侧为频谱/功率谱绘图区，对于无噪的确定性信号绘制 FFT 频谱，对于有噪的随机信号绘制功率谱，并自动标记三个主要频率。

1.2. 标签页 2：数字滤波器设计

标签页 2 实现滤波器设计以及对标签页 1 生成的信号进行滤波处理的功能，布局同样分为左侧参数控制区和右侧的绘图展示区两大部分，如图 2 所示

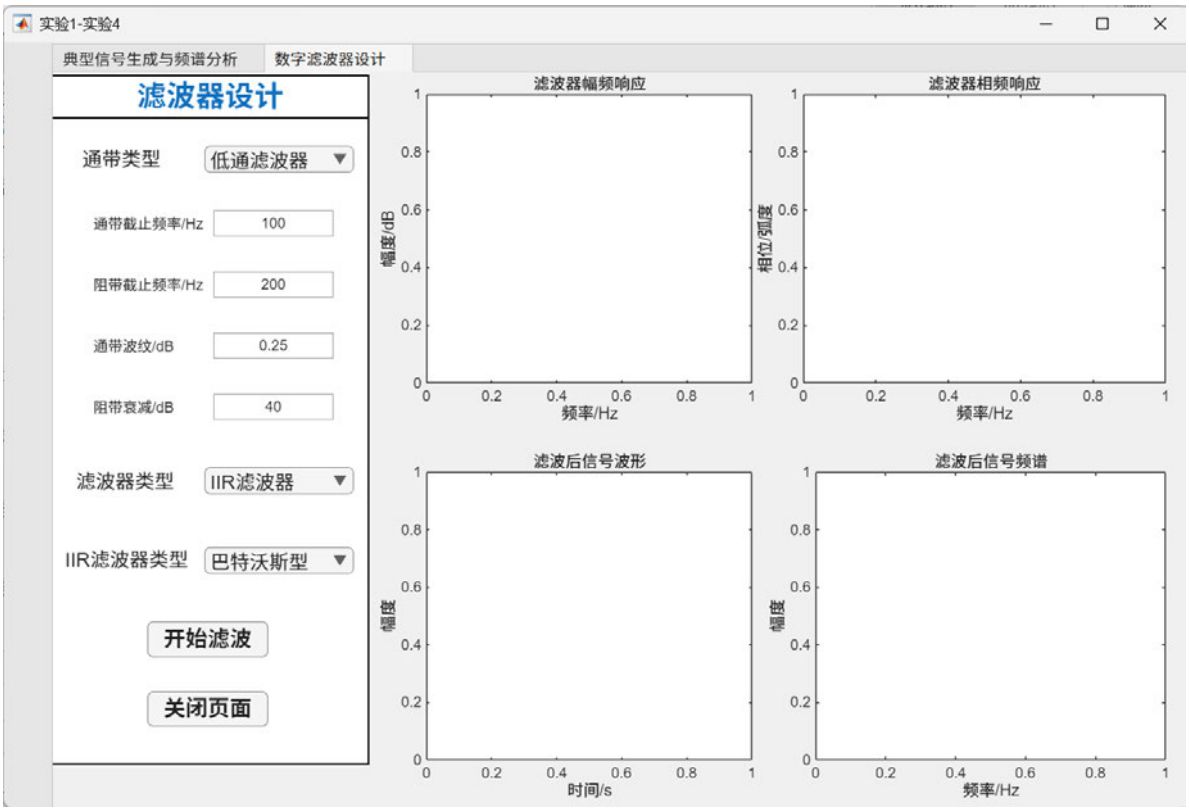

图 2：标签页 2 初始化界面

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. 左侧参数控制区

标签页 1 左侧仅包含一个“滤波器设计”面板。该面板的第一个下拉框用于选择设计滤波器的通带类型（默认为低通滤波器），支持在低通、高通、带通、带阻之间切换。不同通带类型滤波器的通带波纹（默认值 0.25dB）和阻带衰减（默认值 40dB）参数均以数值的形式直接输入，但是截止频率的输入方式和默认值存在不同：

- 低通滤波器：通带截止频率（默认值 100Hz）和阻带截止频率（默认值 200Hz）均只需要输入一个对应数值即可。
- 高通滤波器：通带截止频率（默认值 200Hz）和阻带截止频率（默认值 100Hz）均只需要输入一个对应数值即可。
- 带通滤波器：通带截止频率（默认值 125Hz,175Hz）和阻带截止频率（默认值 100Hz,200Hz）以文本的形式输入两个逗号分隔的数值。
- 带阻滤波器：通带截止频率（默认值 100Hz,200Hz）和阻带截止频率（默认值 125Hz,175Hz）以文本的形式输入两个逗号分隔的数值。

注意，这两种不同类型的编辑字段随着通带类型的选择自动显隐。通过“滤波器类型”下拉框可选择设计 IIR 滤波器还是 FIR 滤波器，当选择设计 IIR 滤波器在下方显

示“IIR 滤波器类型”下拉框，支持巴特沃斯型、切比雪夫 I/II 型、椭圆型滤波器设计（默认为巴特沃斯型）。当选择设计 FIR 滤波器在下方显示“FIR 滤波器类型”下拉框，支持利用矩形窗、汉宁窗、汉明窗、布莱克曼窗和凯瑟窗进行滤波器设计（默认为矩形窗）。

完成滤波器参数设置和类型选择后，可以点击下面的“开始滤波”按钮设计对应的滤波器并对标签页 1 中生成的信号进行滤波处理，同时在右侧绘图展示区绘制相应的图像。完成滤波处理后，若不想进一步操作，可点击“关闭页面”按钮关闭系统。

1.2.2. 右侧绘图展示区

标签页 2 右侧包含 4 个 2×2 排列的坐标区。其中左上方坐标区用于绘制所设计滤波器的幅频响应，右上方的坐标区用于绘制所设计滤波器的相频响应，左下方的坐标区用于绘制滤波后信号的时域波形，右下方的坐标区用于绘制滤波后信号的 FFT 频谱。

1.3. 界面设计特点

- 动态适配性：各个控件显隐随功能选择自动切换，避免界面拥挤，提升操作效率。
- 可视化展示：绘图区与参数设置区分隔明确，信号处理结果直观展示。
- 操作闭环性：系统实现从参数输入、参数设置、功能实现到结果展示的完整操作链，实现信号的全流程处理。

2. 核心模块的具体实现方法

2.1. 信号产生模块

2.1.1. 基本原理

信号产生模块基于典型信号的数学模型，通过设置的信号参数生成典型时域信号，并支持噪声叠加和外部信号文件的导入。不同种类信号的基本原理如下所示：

正弦波的时域表达式为：

$$s(t) = Acos(2\pi ft + \varphi) \tag{1}$$

其中 A 为信号幅度，f 为信号频率，φ为信号初始相位，可由用户输入。

方波可通过矩形脉冲周期延拓生成，其中占空比为控制高电平持续时间占比。

混合正弦波为多个不同参数正弦波的线性叠加，其时域表达式为：

$$s(t) = \sum_{i=1}^{n} A_i \cos(2\pi f_i t + \varphi_i) \tag{2}$$

其中A_i、f_i和φ_i为各个频率分量的初始相位，用户可在对应的编辑字段中以逗号分隔依次输入，MATLAB 可通过字符串处理的方法将其处理成对应数值。

均值为μ，方差为σ^2的高斯白噪声服从$N(\mu, \sigma^2)$分布；均值为μ，方差为σ^2的均匀白噪声服从$U(a, b)$分布，其中 a,b 可由以下公式计算：

$$a = \mu - \sqrt{3\sigma^2},\ b = \mu + \sqrt{3\sigma^2} \tag{3}$$

2.1.2. 关键 MATLAB 函数

- cos：计算余弦函数，生成正弦波时域数据；
- square：生成方波信号，支持占空比参数；
- randn：生成高斯白噪声序列；
- rand：生成均匀分布随机序列，用于构造均匀白噪声；
- strsplit：分割文本输入的混合信号参数（幅度、频率、相位）；
- str2double：将文本参数转换为数值类型；
- uigetfile：打开文件选择对话框，读取外部.mat 信号文件；
- load：加载.mat 文件中的信号数据；

2.1.3. 核心代码

信号产生模块的核心代码为“生成信号”按钮的回调函数，如下所示：

```
function generateSignalButtonPushed(app, event)
    % 获取公共参数
    signalType = app.SignalType.Value;
    fs = app.fsedit.Value;
    L = app.tedit.Value;
    addNoise = app.checknoise.Value;

    % 参数检查
    if fs <= 0
        errordlg('采样频率必须大于 0', '参数错误');
        return;
    end

    if L <= 0
        errordlg('信号时长必须大于 0', '参数错误');
        return;
    end

    % 生成时间轴
    N = floor(fs * L);
    t = (0:N)/fs;

    % 生成基础信号
    switch signalType
        case '正弦波'
            A = app.ampedit.Value;
            phi = app.pedit.Value;
            f = app.fedit.Value;
```

```
29.
30.             if f <= 0
31.                 errordlg('信号频率必须大于 0', '参数错误');
32.                 return;
33.             end
34.
35.             s = A * cos(2*pi*f*t + deg2rad(phi));
36.             title_str = sprintf('正弦
    波 (f=%.1fHz, A=%.1f, ϕ=%.1f°)', f, A, phi);
37.
38.         case '方波'
39.             A = app.ampedit.Value;
40.             duty_cycle = app.Dutyratio.Value;
41.             f = app.fedit.Value;
42.
43.             if f <= 0
44.                 errordlg('信号频率必须大于 0', '参数错误');
45.                 return;
46.             end
47.
48.             if duty_cycle < 0 || duty_cycle > 100
49.                 errordlg('占空比必须在 0~100%之间', '参数错误');
50.                 return;
51.             end
52.
53.             s = A * square(2*pi*f*t, duty_cycle);
54.             title_str = sprintf('方波 (f=%.1fHz, A=%.1f, 占空比
    =%.1f%%)', f, A, duty_cycle);
55.
56.         case '混合正弦波'
57.             % 解析文本参数
58.             A_text = strtrim(app.hunheampedit.Value);
59.             f_text = strtrim(app.hunhefedit.Value);
60.             phi_text = strtrim(app.hunhepedit.Value);
61.
62.             % 分割字符串
63.             A_strs = strsplit(A_text, ',');
64.             f_strs = strsplit(f_text, ',');
65.             phi_strs = strsplit(phi_text, ',');
66.
67.             % 转换为数值
68.             A_values = str2double(A_strs);
69.             f_values = str2double(f_strs);
70.             phi_values = str2double(phi_strs);
71.
```

```
            % 检查参数有效性
            if any(isnan(A_values)) || any(isnan(f_values)) || any(isnan(phi_values))
                errordlg('请输入有效的数值（用逗号分隔）', '参数错误');
                return;
            end

            if length(A_values) ~= length(f_values) || length(A_values) ~= length(phi_values)
                errordlg('幅度、频率和相位的数量必须相同', '参数错误');
                return;
            end

            if any(f_values <= 0)
                errordlg('所有频率必须大于 0', '参数错误');
                return;
            end

            % 生成混合信号
            s = zeros(size(t));
            for i = 1:length(A_values)
                s = s + A_values(i) * cos(2*pi*f_values(i)*t + deg2rad(phi_values(i)));
            end

            % 构建标题
            title_str = '混合正弦波';
            for i = 1:length(A_values)
                title_str = sprintf('%s (A%d=%.1f, f%d=%.1fHz, ϕ%d=%.1f°)', ...
                    title_str, i, A_values(i), i, f_values(i), i, phi_values(i));

            end

        case '加载文件信号'
            if app.filecheck == 0
                errordlg('请先打开文件！','参数错误');
                return;
            end
            s = app.generatedSignal;
            fs = app.generatedFs;

            % 生成时间轴
            N = length(s);
            t = (0:N-1)/fs;

```

```
            % 绘制时域信号
            plot(app.signal, t, s,'b');
            xlabel(app.signal, '时间 t/s');
            ylabel(app.signal, '幅度');
            title(app.signal, '加载的信号波形');
            grid(app.signal, 'on');
            box(app.signal, 'on');
            xlim(app.signal, [t(1),t(end)])
            ylim(app.signal, [min(s),max(s)])

            % 检查是否添加噪声
            addNoise = app.checknoise.Value;

            if addNoise
                % 获取噪声参数
                mu = app.aveedit.Value;
                sigma2 = app.varedit.Value;
                noiseType = app.NoiseType.Value;

                if sigma2 < 0
                    errordlg('方差不能为负数', '错误');
                    return;
                end

                % 生成噪声
                switch noiseType
                    case '高斯白噪声'
                        noise = mu + sqrt(sigma2) * randn(size(s));
                    case '均匀白噪声'
                        a = mu - sqrt(3*sigma2);
                        b = mu + sqrt(3*sigma2);
                        noise = a + (b-a) * rand(size(s));
                end

                s = s + noise;
            end

            % 频谱分析
            plotSpectrum(app, s, fs, '加载文件信号', addNoise);
            return;
    end

    % 添加噪声（如果选中）
    if addNoise
        mu = app.aveedit.Value;
```

```
        sigma2 = app.varedit.Value;
        noiseType = app.NoiseType.Value;

        if sigma2 < 0
            errordlg('方差不能为负数', '参数错误');
            return;
        end

        % 生成噪声
        switch noiseType
            case '高斯白噪声'
                noise = mu + sqrt(sigma2) * randn(size(s));
                noise_str = sprintf(' + 高斯噪声(μ=%.2f,σ²=%.2f)', mu, sigma2);
            case '均匀白噪声'
                a = mu - sqrt(3*sigma2);
                b = mu + sqrt(3*sigma2);
                noise = a + (b-a) * rand(size(s));
                noise_str = sprintf(' + 均匀噪声(μ=%.2f,σ²=%.2f)', mu, sigma2);
        end

        s = s + noise;
        title_str = [title_str, noise_str];
    end

    % 绘制时域波形
    plot(app.signal, t, s ,'b', 'LineWidth', 1.5);
    xlabel(app.signal, '时间 t/s');
    ylabel(app.signal, '幅度');
    title(app.signal, title_str);
    grid(app.signal, 'on');
    box(app.signal, 'on');

    % 设置横坐标范围为整个信号持续时间
    xlim(app.signal, [min(t), max(t)]);

    % 自动调整纵坐标范围
    if ~isempty(s)
        y_min = min(s);
        y_max = max(s);
        y_range = y_max - y_min;
        % 设置纵坐标范围，留出 10%的边距
        if y_min == y_max
            ylim(app.signal, [y_min - 1, y_max + 1]);
        else
            ylim(app.signal, [y_min - 0.1*y_range, y_max + 0.1*y_range]);
```

```
        end
    end

    % 频谱分析
    plotSpectrum(app, s, fs, signalType, addNoise);
    app.generatedSignal = s;
    app.generatedFs = fs;
end
```

2.2. 谱分析模块

2.2.1. 基本原理

谱分析模块用于分析生成信号的谱特性，对于确定信号$x(t)$应分析 FFT 频谱。首先按照采样频率 fs 对信号进行采样，得到采样序列$x[n]$，对其进行 FFT，即为

$$X[k] = FFT(x[n]) \tag{4}$$

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：

$$X\left(j\frac{kf_s}{L}\right) = \frac{X[k]}{f_s} \tag{5}$$

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(5)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。

对于有噪随机信号，则采用周期图法计算功率谱密度，计算公式如下所示：

$$P(f) = \frac{\Delta t}{N}\left|\sum_{n=0}^{N-1} x[n]\, e^{-j2\pi f \Delta t n}\right|^2 \tag{6}$$

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。

在计算出信号的频谱或功率谱后，通过阈值筛选（幅度大于最大值的 10%）和排序，标注前 3 个主要频率分量，方便用户观察。

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；
- nextpow2：计算大于等于信号长度的最小 2 的幂，优化 FFT 运算效率；
- periodogram：计算信号功率谱密度（周期图法）

2.2.3. 核心代码

频谱分析模块的主要代码为函数 plotSpectrum，如下所示：

```
function plotSpectrum(app, s, fs, signalType, addNoise)
    % 绘制信号频谱（无噪）或功率谱（含噪）

    N = length(s);
    L = 2^nextpow2(N); % 使用 2 的幂次方作为 FFT 长度

    % 计算 FFT
    Y = fft(s, L);
    P2 = abs(Y/L);
    P1 = P2(1:L/2+1);
    P1(2:end-1) = 2*P1(2:end-1);

    % 频率轴
    f = fs*(0:(L/2))/L;

    % 清除之前的图形
    cla(app.signal_freq);

    if addNoise
        % 加噪信号：绘制功率谱
        [pxx, f_pxx] = periodogram(s, [], [], fs);
        pxx_dB = 10*log10(pxx);

        % 绘制功率谱
        plot(app.signal_freq, f_pxx, pxx_dB, 'b', 'LineWidth', 1.5);
        xlabel(app.signal_freq, '频率/Hz');
        ylabel(app.signal_freq, '功率谱密度 /dB');
        title(app.signal_freq, '功率谱分析');
        grid(app.signal_freq, 'on');
        box(app.signal_freq, 'on');

        % 设置横坐标范围为 0 到 fs/2
        xlim(app.signal_freq, [0, fs/2]);
        xticks(app.signal_freq,0:fs/20:fs/2)
        xticklabels(app.signal_freq,0:fs/20:fs/2)

        % 自动调整纵坐标范围
        y_vals = pxx_dB;
        if ~isempty(y_vals)
            y_min = min(y_vals);
            y_max = max(y_vals);
            y_range = y_max - y_min;
            ylim(app.signal_freq, [y_min - 0.1*y_range, y_max + 0.1*y_range]);
```

```
        end

        % 标注主要频率分量
        if ismember(signalType, {'正弦波', '方波', '混合正弦波'})
            % 寻找显著的峰值（幅度大于最大值的 10%）
            threshold = max(pxx) * 0.1;
            peaks = [];
            locs = [];

            % 简单的峰值检测
            for i = 2:length(pxx_dB)-1
                if pxx(i) > pxx(i-1) && pxx(i) > pxx(i+1) && pxx(i) > threshold
                    peaks = [peaks, pxx_dB(i)];
                    locs = [locs, i];
                end
            end

            % 按峰值大小排序
            [peaks_sorted, sort_idx] = sort(peaks, 'descend');
            locs_sorted = locs(sort_idx);

            % 标注前 3 个主要频率
            hold(app.signal_freq, 'on');
            for i = 1:min(3, length(peaks_sorted))
                plot(app.signal_freq, f_pxx(locs_sorted(i)), peaks_sorted(i), 'ro', ...
                    'MarkerSize', 5, 'LineWidth', 1.5);
                text(app.signal_freq, f_pxx(locs_sorted(i))+fs/120, peaks_sorted(i), ...
                    sprintf('%.1f Hz', f_pxx(locs_sorted(i))), ...
                    'FontSize', 10, 'HorizontalAlignment', 'left');
            end
            hold(app.signal_freq, 'off');
        end

    else
        % 不加噪信号：绘制 FFT 频谱
        plot(app.signal_freq, f, P1, 'b', 'LineWidth', 1.5);
        xlabel(app.signal_freq, '频率/Hz');
        ylabel(app.signal_freq, '幅度');
        title(app.signal_freq, 'FFT 频谱分析');
        grid(app.signal_freq, 'on');
        box(app.signal_freq, 'on');

```

```
        % 设置横坐标范围为 0 到 fs/2
        xlim(app.signal_freq, [0, fs/2]);
        xticks(app.signal_freq,0:fs/20:fs/2)
        xticklabels(app.signal_freq,0:fs/20:fs/2)

        % 自动调整纵坐标范围
        if ~isempty(P1)
            y_min = min(P1);
            y_max = max(P1);
            y_range = y_max - y_min;
            % 设置纵坐标范围，留出 10%的边距
            if y_min == y_max
                ylim(app.signal_freq, [y_min - 1, y_max + 1]);
            else
                ylim(app.signal_freq, [max(y_min - 0.1*y_range, 0), y_max + 0.1*y_range]);
            end
        end

        % 标注主要频率分量
        if ismember(signalType, {'正弦波', '方波', '混合正弦波'})
            % 寻找显著的峰值（幅度大于最大值的 10%）
            threshold = max(P1) * 0.1;
            peaks = [];
            locs = [];

            % 简单的峰值检测
            for i = 2:length(P1)-1
                if P1(i) > P1(i-1) && P1(i) > P1(i+1) && P1(i) > threshold
                    peaks = [peaks, P1(i)];
                    locs = [locs, i];
                end
            end

            % 按峰值大小排序
            [peaks_sorted, sort_idx] = sort(peaks, 'descend');
            locs_sorted = locs(sort_idx);

            % 标注前 3 个主要频率
            hold(app.signal_freq, 'on');
            for i = 1:min(3, length(peaks_sorted))
                plot(app.signal_freq, f(locs_sorted(i)), peaks_sorted(i), 'ro', ...
                    'MarkerSize', 5, 'LineWidth', 1.5);
```

```
                text(app.signal_freq, f(locs_sorted(i))+fs/120, peaks_sorted(i),
    ...
                    sprintf('%.1f Hz', f(locs_sorted(i))), ...
                    'FontSize', 10, 'HorizontalAlignment', 'left');
            end
            hold(app.signal_freq, 'off');
        end
    end
end
```

2.3. 滤波器设计模块

2.3.1. 基本原理

滤波器设计模块支持 IIR 和 FIR 两种类型的数字滤波器，覆盖低通、高通、带通、带阻四种通带类型，其核心原理如下：

对于 IIR 滤波器，可采用双线性变换法进行设计。首先将所设计滤波器的模拟截止频率ω_p、ω_s转换为数字频率Ω_p、Ω_s，然后按照以下公式进行频率预畸：

$$\omega = \frac{2}{T} tan\left(\frac{\Omega}{2}\right) \tag{7}$$

由此得到模拟原型滤波器的截止频率ω_p'、ω_s'。选择对应的模拟原型滤波器种类（本系统支持巴特沃斯型、切比雪夫 I/II 型、椭圆型模拟滤波器），根据预畸后的截止频率ω_p'和ω_s'、通带波纹和阻带衰减等参数设计模拟原型滤波器$H_a(s)$，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

$$H(z) = H_a\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\right) \tag{8}$$

对于 FIR 滤波器，采用窗函数法，通过对理想的滤波器频率响应$H_d\left(e^{j\Omega}\right)$加窗抑制频谱泄漏，然后用窗函数截取它的单位冲激响应$h_d(n)$，得到线性相位和因果的 FIR 滤波器。为了得到一个具有线性相位的因果系统，在通带内应满足

$$H_d\left(e^{j\Omega}\right) = e^{-j\alpha\Omega} \tag{9}$$

其中

$$\alpha = \frac{j\Omega(N-1)}{2} \tag{10}$$

由于在 FIR 滤波器设计过程中通带波纹和阻带衰减不参与运算，因此改变系统中相应参数的输入不会对滤波器的设计产生影响。这也反映出 FIR 滤波器设计的灵活性不如 IIR 滤波器。但为了尽可能满足阻带衰减的需求，可以在设计滤波器时按表 1 中各窗函数设计的滤波器最小阻带衰减的指标选择窗函数。为了满足过渡带宽的需求，需要根据表 1 中不同窗函数设计的滤波器过渡带宽的指标确定滤波器的长度 N，如果

阶数 N 过小。为保证设计效果，统一将其增加至 65。

表 1 常用窗函数的特性

窗函数	窗函数频率特性		加窗后滤波器指标	
	旁瓣峰值（dB）	主瓣宽度	过渡带宽	最小阻带衰减（dB）
矩形窗	-13	$4\pi/N$	$1.8\pi/N$	-21
汉宁窗	-31	$8\pi/N$	$6.2\pi/N$	-44
海明窗	-41	$8\pi/N$	$6.6\pi/N$	-53
布莱克曼窗	-57	$12\pi/N$	$11\pi/N$	-74

凯瑟窗是一种在实际中广泛应用的窗函数，它由下式给定

$$w(n)=\frac{I_0\left[\beta\sqrt{1-\left(1-\frac{2n}{N-1}\right)^2}\right]}{I_0[\beta]},0\le n\le N-1 \tag{11}$$

其中$I_0[x]$是修正的零阶贝塞尔（Bessel）函数，参数β控制最小阻带衰减，这种窗函数对于相同的N可以提供不同的过渡带宽。已知给定的指标Ω_p，Ω_{st}，R_p和A_s，滤波器长度N、凯瑟窗参数β和过渡带宽$\Delta\Omega$可以按如下凯瑟窗设计方程给出：

$$\Delta\Omega=\Omega_{st}-\Omega \tag{12}$$

$$N\approx\frac{A_s-7.95}{2.285\Delta\Omega}+1 \tag{13}$$

$$\beta=\begin{cases}0.1102(A_s-8.7) & A_s\ge 50\\ 0.5842(A_s-21)^{0.4}+0.07886(A_s-21) & 21<A_s<50\end{cases} \tag{14}$$

完成上述滤波器设计后，即可计算滤波器系统函数$H(z)$分母和分子的系数a_k和b_k，一般令$a_0=1$。当$k\neq 0$时，对于 IIR 滤波器a_k一般不为 0，而对于 FIR 滤波器，$a_k=0$。

2.3.2. 关键 MATLAB 函数

- buttord/butter：巴特沃斯滤波器阶数计算与系数设计；
- cheb1ord/cheby1：切比雪夫 I 型滤波器阶数计算与系数设计；
- cheb2ord/cheby2：切比雪夫 II 型滤波器阶数计算与系数设计；
- ellipord/ellip：椭圆滤波器阶数计算与系数设计；
- fir1：基于窗函数的 FIR 滤波器系数设计；
- rectwin/hann/hamming/blackman/kaiser：生成各类窗函数；
- freqz：计算滤波器频率响应；

2.3.3. 核心代码

滤波器设计的核心代码为函数 designFilter，如下所示：

```
1.  function [b, a] = designFilter(app)
```

```
    % 设计滤波器并返回系数
    % 返回值: b - 分子系数, a - 分母系数

    filterType = app.lvboType.Value;       % IIR 或 FIR
    passType = app.tdType.Value;           % 低通或高通
    Rp = app.tongdaidB.Value;                % 通带波纹
    Rs = app.zudaidB.Value;                  % 阻带衰减
    fs = app.generatedFs;                    % 采样频率

    % 初始化输出参数（防止未赋值错误）
    b = [];
    a = [];

    % 检查必要的参数
    if isempty(fs) || fs <= 0
        errordlg('采样频率无效，请先生成信号', '参数错误');
        return;
    end

    % 根据滤波器类型获取频率参数
    if strcmp(passType, '低通滤波器') || strcmp(passType, '高通滤波器')
        % 低通或高通滤波器
        fp = app.tongdaiHz.Value;                % 通带截止频率
        fst = app.zudaiHz.Value;              % 阻带截止频率

        % 归一化频率
        wp = fp / (fs/2);
        ws = fst / (fs/2);

        % 将中文通带类型转换为英文
        if strcmp(passType, '低通滤波器')
            passTypeEnglish = 'low';
        else
            passTypeEnglish = 'high';
        end
    else

        % 带通或带阻滤波器
        % 从文本字段读取频率参数
        fp_text = strtrim(app.dtztongdaiHz.Value);
        fst_text = strtrim(app.dtzzudaiHz.Value);

        % 分割字符串
        fp_strs = strsplit(fp_text, ',');
        fst_strs = strsplit(fst_text, ',');
```

```
47.
48.         % 转换为数值
49.         fp = str2double(fp_strs);
50.         fst_values = str2double(fst_strs);
51.
52.         % 归一化频率
53.         wp = fp / (fs/2);
54.         ws = fst_values / (fs/2);
55.
56.         % 将中文通带类型转换为英文
57.         if strcmp(passType, '带通滤波器')
58.             passTypeEnglish = 'bandpass';
59.         else
60.             passTypeEnglish = 'stop';
61.         end
62.     end
63.
64.     if strcmp(filterType, 'IIR 滤波器')
65.         % IIR 滤波器设计
66.         iirType = app.IIRType.Value;
67.
68.         try
69.             % 计算最小阶数
70.             switch iirType
71.                 case '巴特沃斯'
72.                     [N, Wn] = buttord(wp, ws, Rp, Rs);
73.                     [b, a] = butter(N, Wn, passTypeEnglish);
74.                 case '切比雪夫 I 型'
75.                     [N, Wn] = cheb1ord(wp, ws, Rp, Rs);
76.                     [b, a] = cheby1(N, Rp, Wn, passTypeEnglish);
77.                 case '切比雪夫 II 型'
78.                     [N, Wn] = cheb2ord(wp, ws, Rp, Rs);
79.                     [b, a] = cheby2(N, Rs, Wn, passTypeEnglish);
80.                 case '椭圆'
81.                     [N, Wn] = ellipord(wp, ws, Rp, Rs);
82.                     [b, a] = ellip(N, Rp, Rs, Wn, passTypeEnglish);
83.                 otherwise
84.                     [N, Wn] = buttord(wp, ws, Rp, Rs);
85.                     [b, a] = butter(N, Wn, passTypeEnglish);
86.             end
87.         catch ME
88.             errordlg(sprintf('IIR 滤波器设计失败: %s', ME.message), '设计错误');
89.             return;
90.         end
91.     else
```

```
92.         % FIR 滤波器设计
93.         firType = app.FIRType.Value;
94.         
95.         % 缩小过渡带，更好满足指标
96.         wp = ws/3 + 2*wp/3;
97.         ws = 2*ws/3 + wp/3;
98.         Delta_w = min(abs(ws - wp));      % 过渡带宽
99.         
100.        try
101.            % 设置窗函数
102.            switch firType
103.                case '矩形窗'
104.                    N = round(1.8/(Delta_w) + abs(mod(1.8/(Delta_w),2)-1));
105.                    if N <= 65
106.                        N = 65;
107.                    end
108.                    window = rectwin(N);
109.                case '汉宁窗'
110.                    N = round(6.2/(Delta_w) + abs(mod(6.2/(Delta_w),2)-1));
111.                    if N <= 65
112.                        N = 65;
113.                    end
114.                    window = hann(N);
115.                case '汉明窗'
116.                    N = round(6.6/(Delta_w) + abs(mod(6.6/(Delta_w),2)-1));
117.                    if N <= 65
118.                        N = 65;
119.                    end
120.                    window = hamming(N);
121.                case '布莱克曼窗'
122.                    N = round(11/(Delta_w) + abs(mod(11/(Delta_w),2)-1));
123.                    if N <= 65
124.                        N = 65;
125.                    end
126.                    window = blackman(N);
127.                case '凯瑟窗'
128.                    beta = 0.1102 * (Rs - 8.7);
129.                    N = ceil((Rs - 7.95)/(2.285 * Delta_w * pi) + 1);
130.                    N = round(N + abs(mod(N,2)-1));
131.                    if N <= 65
132.                        N = 65;
133.                    end
134.                    window = kaiser(N, beta);
135.                otherwise
136.                    window = hamming(65);
```

```
            end

            % 使用英文通带类型
            Wn = fp / (fs/2);
            b = fir1(N - 1, Wn, passTypeEnglish, window);
            a = 1; % FIR 滤波器分母系数为 1
        catch ME
            errordlg(sprintf('FIR 滤波器设计失败: %s', ME.message), '设计错误');
            return;
        end
    end
end
```

2.4. 滤波处理模块

2.4.1. 基本原理

滤波处理模块采用线性滤波方法，将设计好的滤波器系数应用于信号生成模块生成的信号，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：

$$y[n] = x[n] * h[n] \tag{15}$$

对于 IIR 滤波器一般有：

$$y[n] = -\sum_{k=1}^{M} a_k y[n-k] + \sum_{k=0}^{N} b_k x[n-k] \tag{16}$$

而对于 FIR 滤波器，由于当$k \neq 0$时$a_k = 0$，公式(16)可简化为：

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] \tag{17}$$

2.4.2. 关键 MATLAB 函数

- filter：执行线性滤波，输入滤波器系数 b,a 和信号 x，输出滤波后信号 y；
- freqz：计算滤波器频率响应，用于绘制幅频 / 相频特性曲线；
- angle：计算复数频率响应的相位角，得到相频响应；

2.4.3. 核心代码

滤波处理模块核心代码为“开始滤波”按钮的回调函数，如下所示：

```
function lvboButtonPushed(app, event)
    % 检查是否生成信号
    if isempty(app.generatedSignal)
        errordlg('请先生成信号', '错误');
        return;
    end

    % 检查是否已存储采样频率
    if isempty(app.generatedFs)
        errordlg('未找到采样频率信息', '错误');
        return;
```

```
    end

    passType = app.tdType.Value;           % 低通或高通

    % 根据滤波器类型获取频率参数
    if strcmp(passType, '低通滤波器') || strcmp(passType, '高通滤波器')
        % 低通或高通滤波器
        fp = app.tongdaiHz.Value;               % 通带截止频率
        fst = app.zudaiHz.Value;              % 阻带截止频率

        % 参数检查
        if fp <= 0 || fst <= 0
            errordlg('截止频率必须大于 0', '参数错误');
            return;
        end

        if strcmp(passType, '低通滤波器')
            if fp >= fst
                errordlg('低通滤波器：通带截止频率应小于阻带截止频率', '参数错误');
                return;
            end
        else % 高通滤波器
            if fp <= fst
                errordlg('高通滤波器：通带截止频率应大于阻带截止频率', '参数错误');
                return;
            end
        end
    else
        % 带通或带阻滤波器
        % 从文本字段读取频率参数
        fp_text = strtrim(app.dtztongdaiHz.Value);
        fst_text = strtrim(app.dtzzudaiHz.Value);

        % 分割字符串
        fp_strs = strsplit(fp_text, ',');
        fst_strs = strsplit(fst_text, ',');

        % 转换为数值
        fp_values = str2double(fp_strs);
        fst_values = str2double(fst_strs);

        % 检查参数有效性
        if any(isnan(fp_values)) || any(isnan(fst_values))
            errordlg('请输入有效的频率数值（用逗号分隔）', '参数错误');
            return;
```

```
        end

        if length(fp_values) ~= 2 || length(fst_values) ~= 2
            errordlg('带通/带阻滤波器需要两个频率参数', '参数错误');
            return;
        end

        if any(fp_values <= 0) || any(fst_values <= 0)
            errordlg('截止频率必须大于 0', '参数错误');
            return;
        end

        % 检查频率顺序
        if fp_values(1) >= fp_values(2)
            errordlg('通带截止频率必须由低到高', '参数错误');
            return;
        end

        if fst_values(1) >= fst_values(2)
            errordlg('阻带截止频率必须由低到高', '参数错误');
            return;
        end

        % 根据滤波器类型检查频率关系
        if strcmp(passType, '带通滤波器')
            % 带通滤波器：通带在中间，阻带在两边
            if not(fp_values(1) > fst_values(1) && fp_values(2) < fst_values(2))

                errordlg('带通滤波器：通带应在阻带之内', '参数错误');
                return;
            end
        else
            % 带阻滤波器：阻带在中间，通带在两边
            if not(fst_values(1) > fp_values(1) && fst_values(2) < fp_values(2))

                errordlg('带阻滤波器：阻带应在通带之内', '参数错误');
                return;
            end
        end
    end

    % 获取信号和采样频率
    signal = app.generatedSignal;
    fs = app.generatedFs;

```

```
    % 设计滤波器
    try
        [b, a] = designFilter(app);
    catch ME
        errordlg(sprintf('滤波器设计失败: %s', ME.message), '错误');
        return;
    end

    % 绘制滤波器响应
    plotFilterResponse(app, b, a, fs);

    % 滤波信号
    filteredSignal = filter(b, a, signal);

    % 绘制滤波后信号
    plotFilteredSignal(app, filteredSignal, fs);
end
```

三、 系统功能测试

为了验证上述设计的信号处理系统是否可以正常工作，需要对其进行系统功能测试。为了节省篇幅，在此只选取一些典型信号和滤波处理的测试案例进行说明，以此表明系统工作的有效性。

1. 典型信号生成和频谱分析

在这一部分的测试中，生成正弦波信号、方波信号、混合正弦波信号、从外部加载文件信号并进行加噪处理，观察信号波形和谱特性并与理论值作比较，判断系统是否正确工作。

1.1. 正弦波信号

为简单起见，按默认参数生成正弦波信号，信号波形和频谱如下所示：

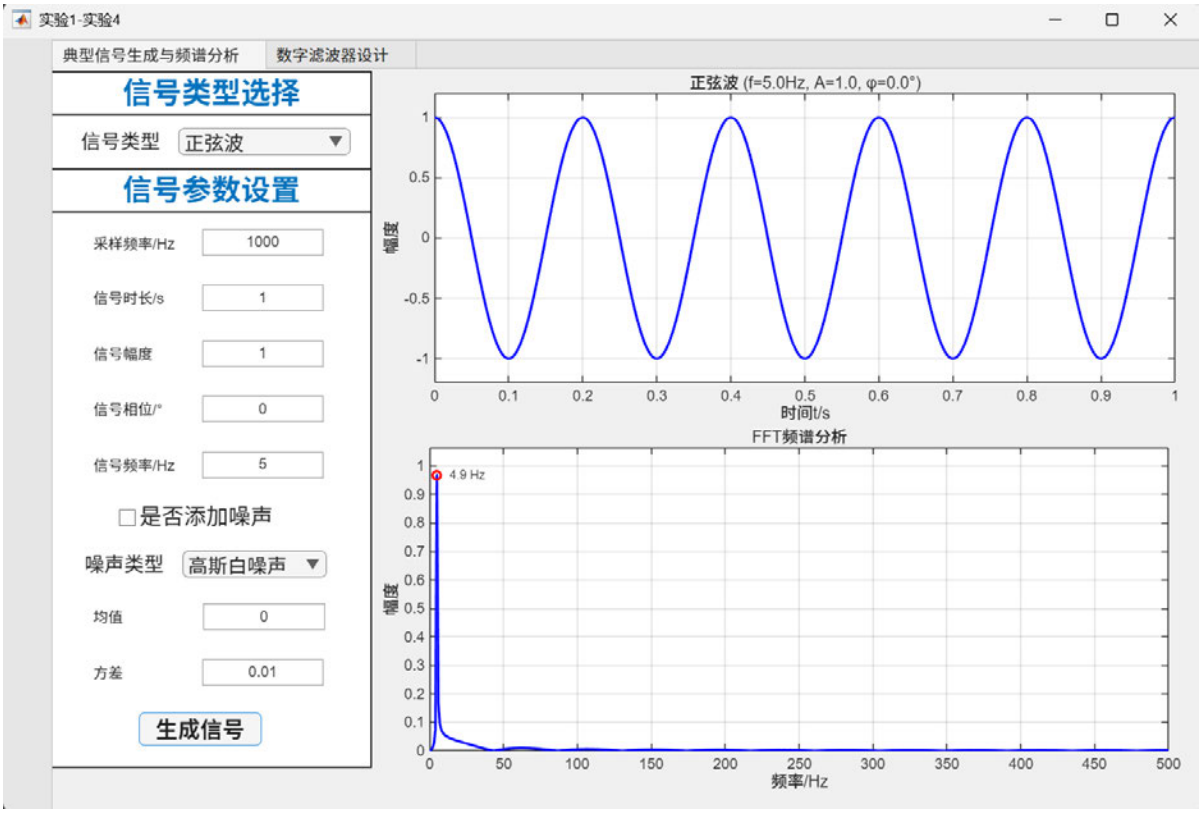

图 3：正弦波信号测试

观察测试结果可知，信号波形为标准的正弦波，频谱图最大频率分量出现在 5Hz 左右，这与输入参数基本相符。但信号的最大频率分量精确落在 5Hz 频率点上，而是出现了略微的偏差，同时信号的频谱图出现了其他的频率分量，这与正弦波信号的理论频谱存在差异。但这种差异是合理的，因为只有无限长正弦波信号的频谱才是理想的冲激函数，有限长的正弦波信号相当于对无限长信号进行了加窗处理，必然会发生频谱泄漏的现象，导致频谱能量的扩散以及频率分辨率的降低。由于后续所有产生的信号都是有限长信号，因此都会发生频谱泄漏现象，但只要其显著特征与理论值相符，即可认为系统进行了正确的谱分析。

注意输入采样频率，信号时长、频率必须大于 0，否则系统会提示错误。

1.2. 方波信号

为简单起见，按默认参数生成正弦波信号，信号波形和频谱如下所示：

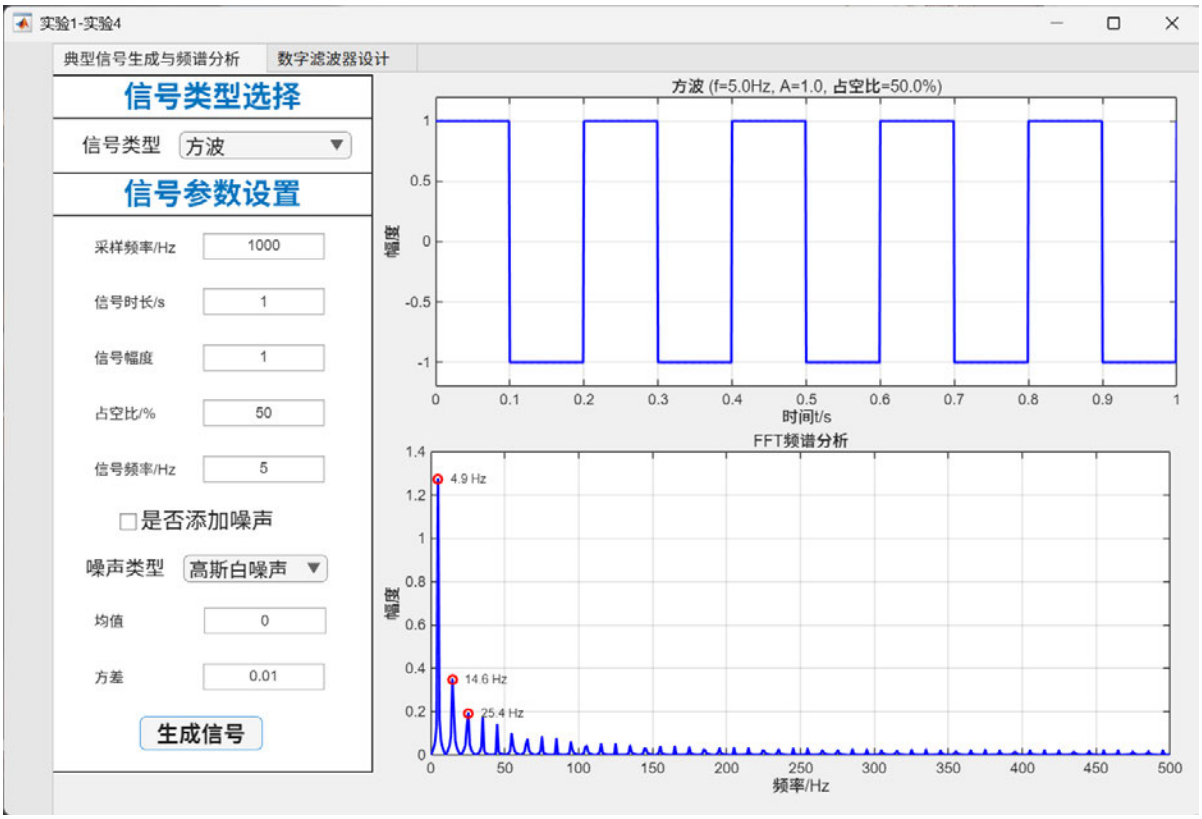

图 4：方波信号测试

观察测试结果可知，信号波形为标准的方波，且符合输入参数的要求。占空比为50%方波的傅里叶级数展开为：

$$f(t)=\frac{4A}{\pi}\sum_{n=1}^{\infty}\frac{\sin\left((2n-1)\omega t\right)}{2n-1} \tag{18}$$

信号的频谱图中三个主要频率分量分别落在频率点 4.9Hz、14.6Hz、25.4Hz，近似为基波频率 5Hz 的奇数倍，且随着频率的升高，频率分量的幅值逐渐变小，这与公式(18)的理论分析相符。注意输入信号占空比应在 0~100 之间，否则系统会提示错误。

1.3. 混合正弦波信号

首先按默认参数生成混合正弦波信号，如下图所示：

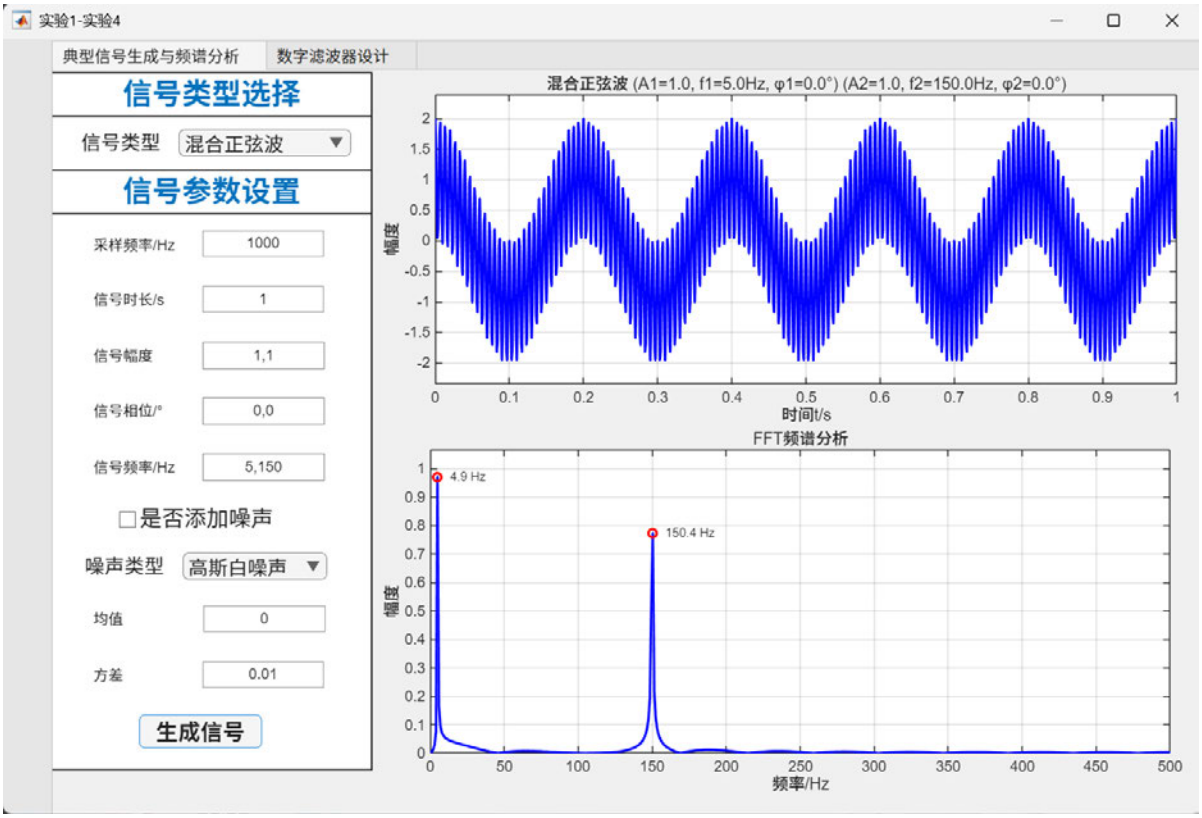

图 5：两个频率分量的混合正弦波信号测试

观察到信号频谱的主要频率分量为 4.9Hz、150.4Hz，这与输入基本相符。如果在在上述混合正弦波信号中再增加一个幅值为 2，初始相位0°的 300Hz 的频率分量后，结果如下图所示：

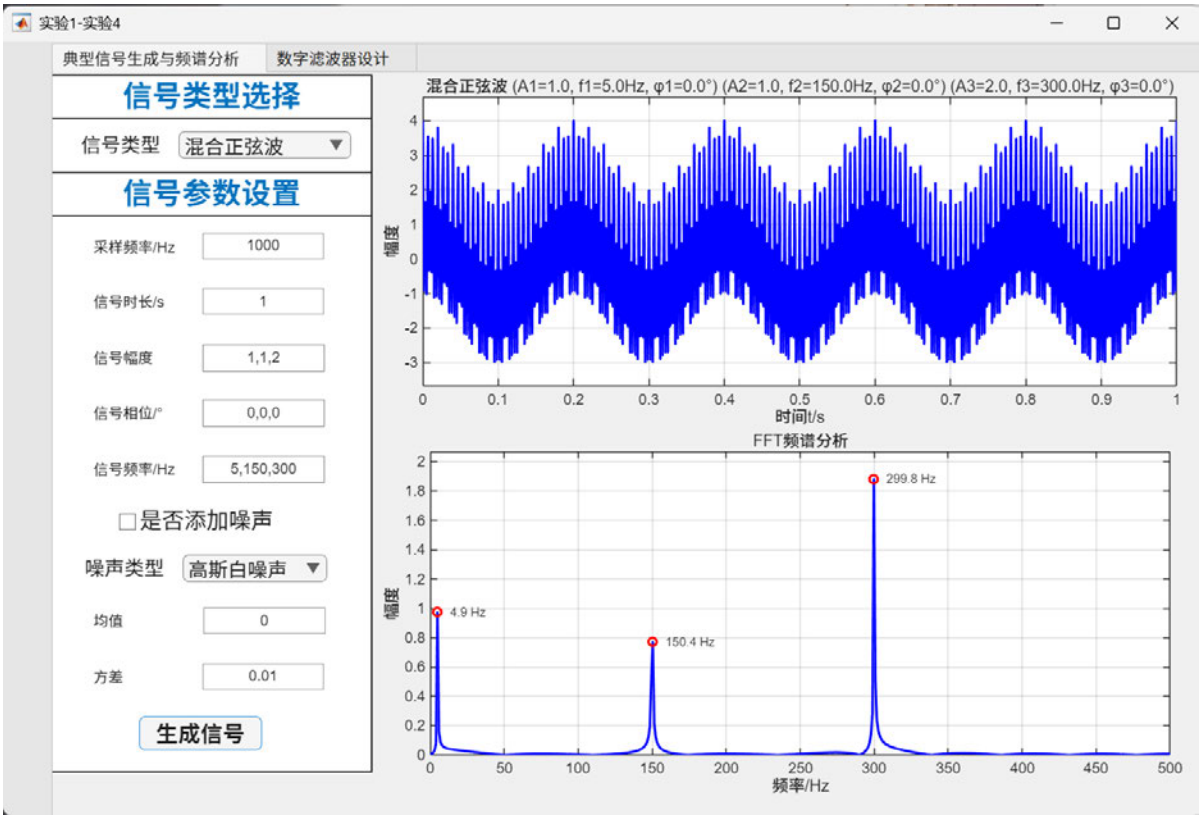

图 6：三个频率分量的混合正弦波信号测试

观察到信号频谱的主要频率分量为 4.9Hz、150.4Hz、299.8Hz，这与输入基本相符。有上面两个案例可以看出，所设计的信号处理系统可产生多个不同频率分量的混合正弦波信号。注意信号幅度、相位和频率参数的输入数量必须一致，否则系统会提示错误。

1.4. 加载文件信号

导入外部 ssvep_s10Hz.mat 信号文件后，信号波形和频谱如下所示：

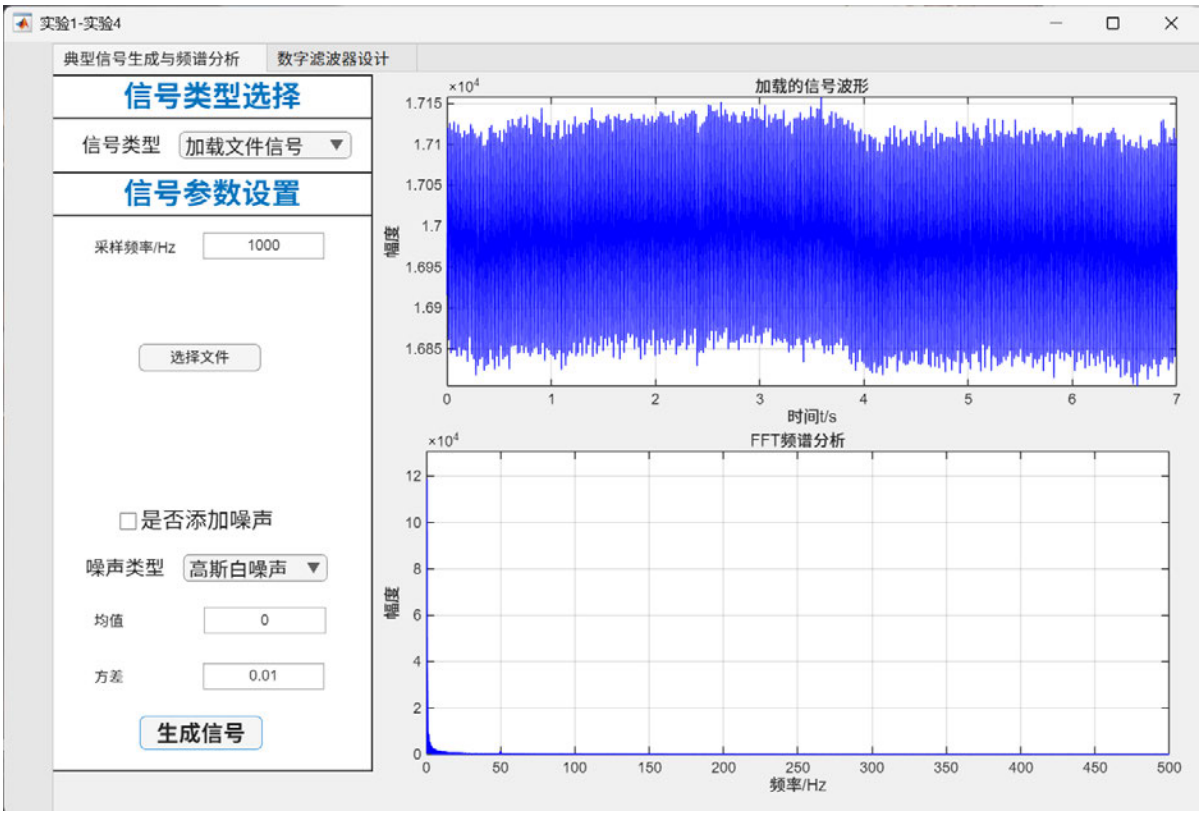

图 7：导入文件信号测试

加载文件信号必须遵循下面的操作流程：首先点击“选择文件”按钮选择对应的信号文件，若文件加载失败系统会提示错误。导入文件后点击“生成信号”按钮即可生成时域波形和谱分析结果。如果实现不导入文件，系统会提示请先导入文件。

观察上述文件信号，从时域角度可看出存在一个 17000 左右的直流偏量，从频谱图的角度可以再次印证这一特点，进一步表明了该系统工作的正确性。

1.5. 信号加噪处理

在完成确定信号的分析后，接下来需要对信号进行加噪处理。为简单起见，这里仅分析对第 1 部分的正弦波信号进行加噪的结果。噪声类型分别选择高斯白噪声和均匀白噪声，噪声均值和方差均保持默认值，此时信号具有较高的信噪比，加噪后的结果如下图所示：

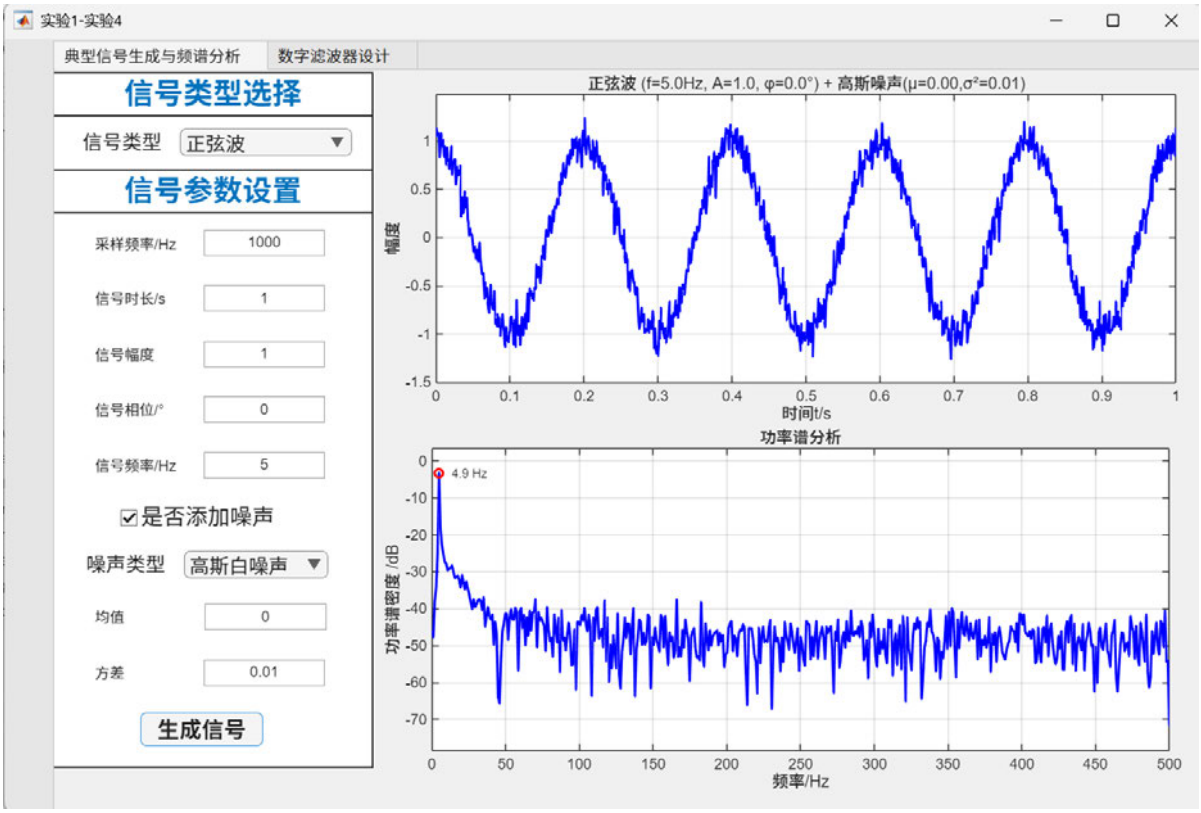

图 8：正弦波加高斯白噪声信号测试

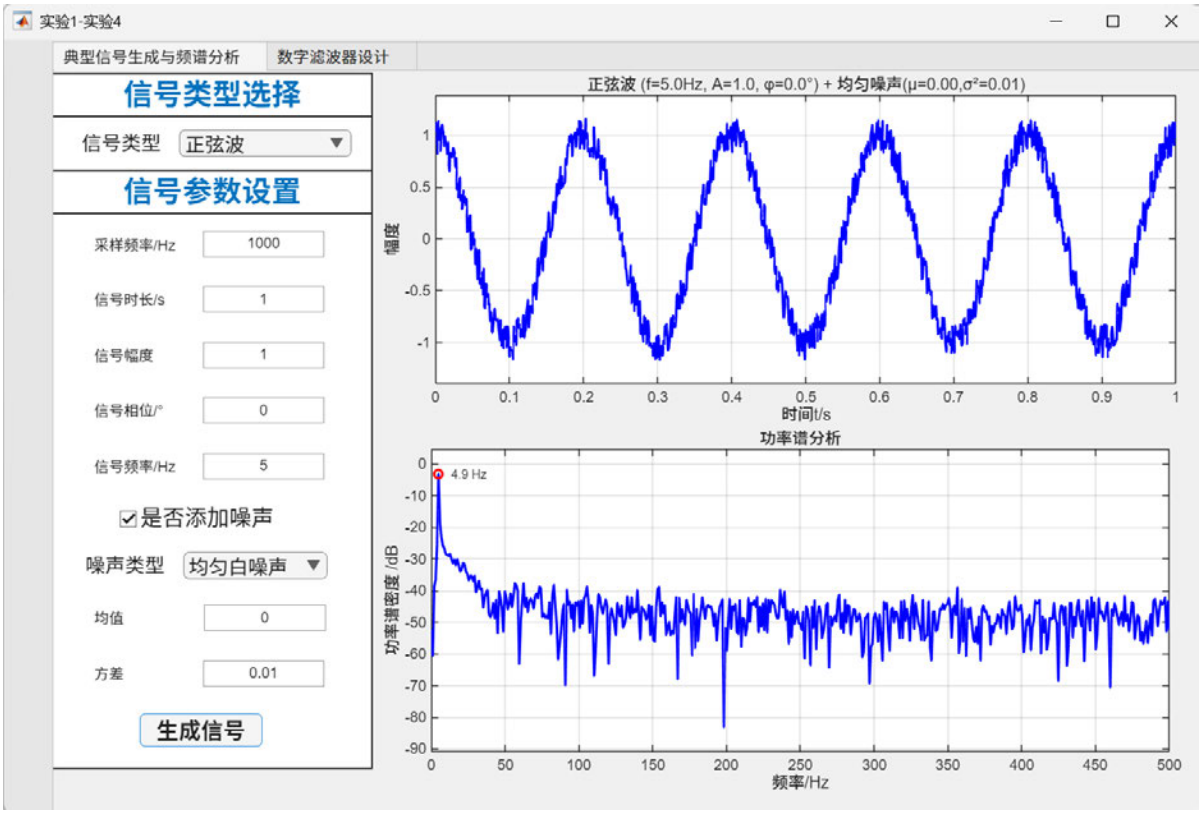

图 9：正弦波加均匀白噪声信号测试

观察上图可知，在信噪比较高时，信号仍保持原有的包络，但在噪声的影响下出现了一定的失真。对比高斯白噪声和均匀白噪声可发现，均匀白噪声的幅值分布较为

均匀，而高斯白噪声有时会出现较大的噪声分量。观察加噪后信号的功率谱密度可知，以上两张图的功率谱密度的主要频率分量均为 4.9Hz，这与原始无噪信号相符。而噪声的频谱范围很宽，从低频到高频一直都有分布。注意输入噪声方差应当大于 0，否则系统会提示错误。

作为对比，这里再次给出信噪比较低时正弦波加高斯白噪声的信号测试：

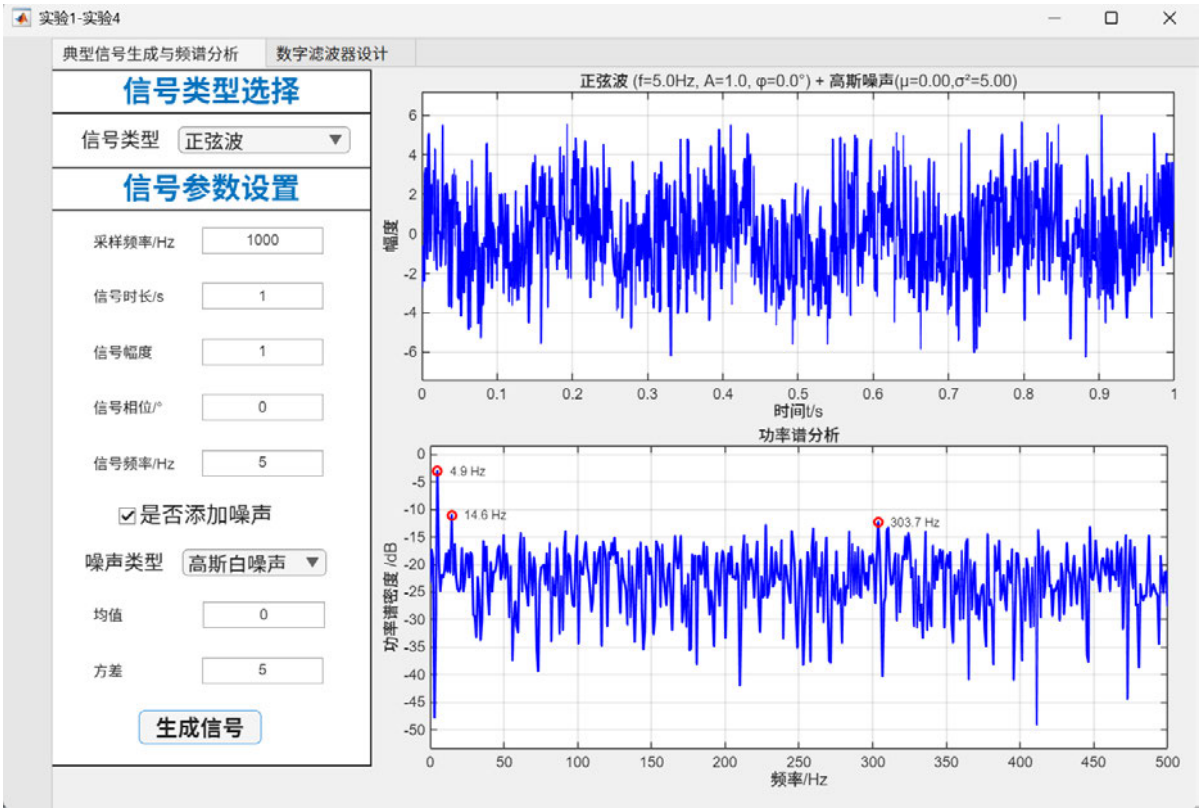

图 10：正弦波加高斯白噪声信号测试(低信噪比)

由于实际情况下均匀白噪声较为少见，因此在后续的对比分析和信号滤波降噪处理不再讨论均匀白噪声的情况，仅对高斯白噪声进行详细分析。可以看出当信噪比较低时，信号将出现比较严重的失真，原有包络将变得不再明显，与此同时原有信号 4.9Hz 的频率分量将变得不是特别明显，此时有用信号几乎完全淹没于噪声中。因此如果要提高信号的传输质量，一定要保证较高的信噪比。

2. 滤波器设计

本信号处理系统支持多种通带类型和设计方法的滤波器设计，存在多种排列组合方案，如果逐一进行功能测试工作量太大。在进行滤波器功能测试时，将从低通、高通、带通和带阻四种通带类型中分别选择不同的滤波器设计方案，力求覆盖所有 IIR 和 FIR 滤波器类型，有效验证该系统滤波处理的可靠性。所有的滤波器测试环节将选择默认参数生成的混合正弦波信号进行，便于观察滤波器对特定频率分量的滤除效果。

在完成功能测试后，利用该系统设计滤波器进行信号降噪分析和方波信号的滤波分析，从而体现滤波器设计的实际价值。

2.1. 滤波器功能测试

2.1.1. 低通滤波器

设置滤波器通带截止频率为 30Hz，阻带截止频率为 50Hz，通带波纹和阻带衰减保持默认数值。设计巴特沃斯型 IIR 滤波器对信号进行滤波处理，结果如下图所示：

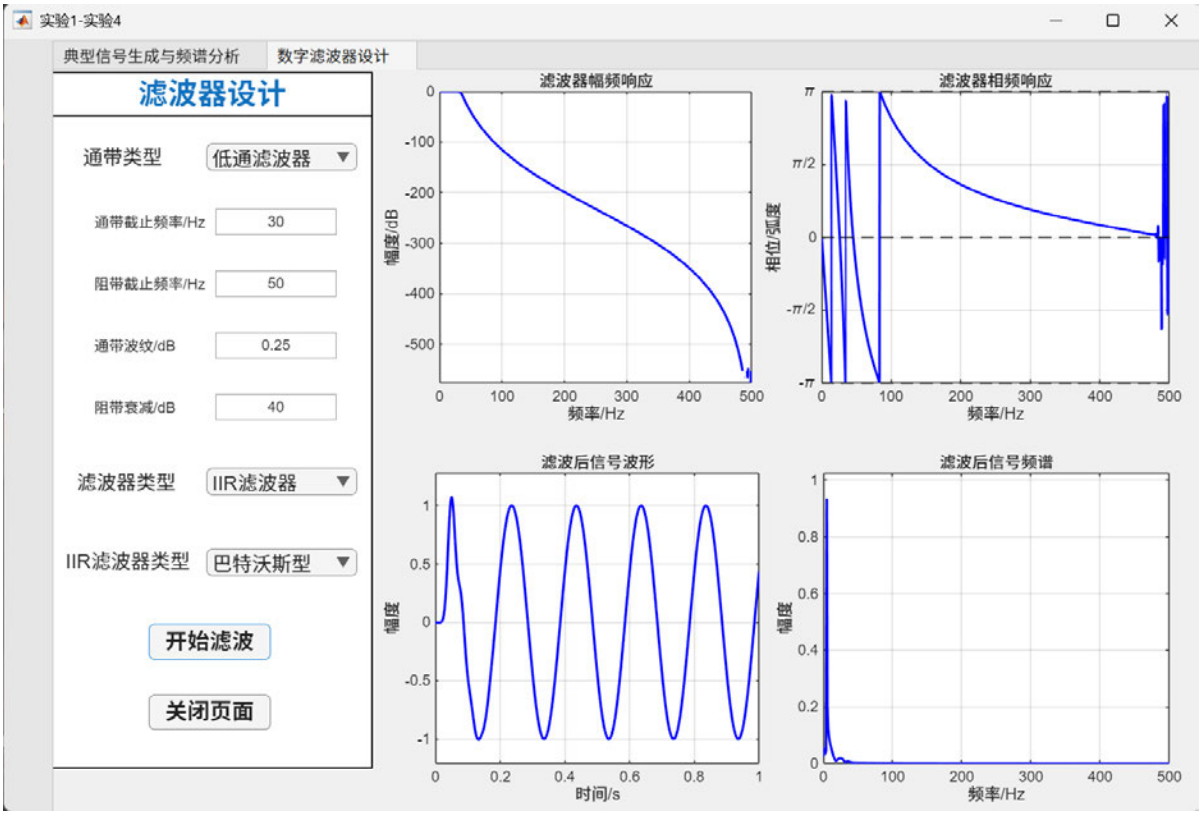

图 11：巴特沃斯低通滤波器设计

观察上图可以发现，设计的巴特沃斯低通滤波器在通带和阻带幅频响应不存在波纹，截止频率与输入参数相符。信号经滤波后仅保留 5Hz 的频率分量，150Hz 频率分量被滤除，这表明滤波器设计是合理的。

利用矩形窗和汉宁窗设计 FIR 滤波器对信号进行滤波处理结果如下图所示：

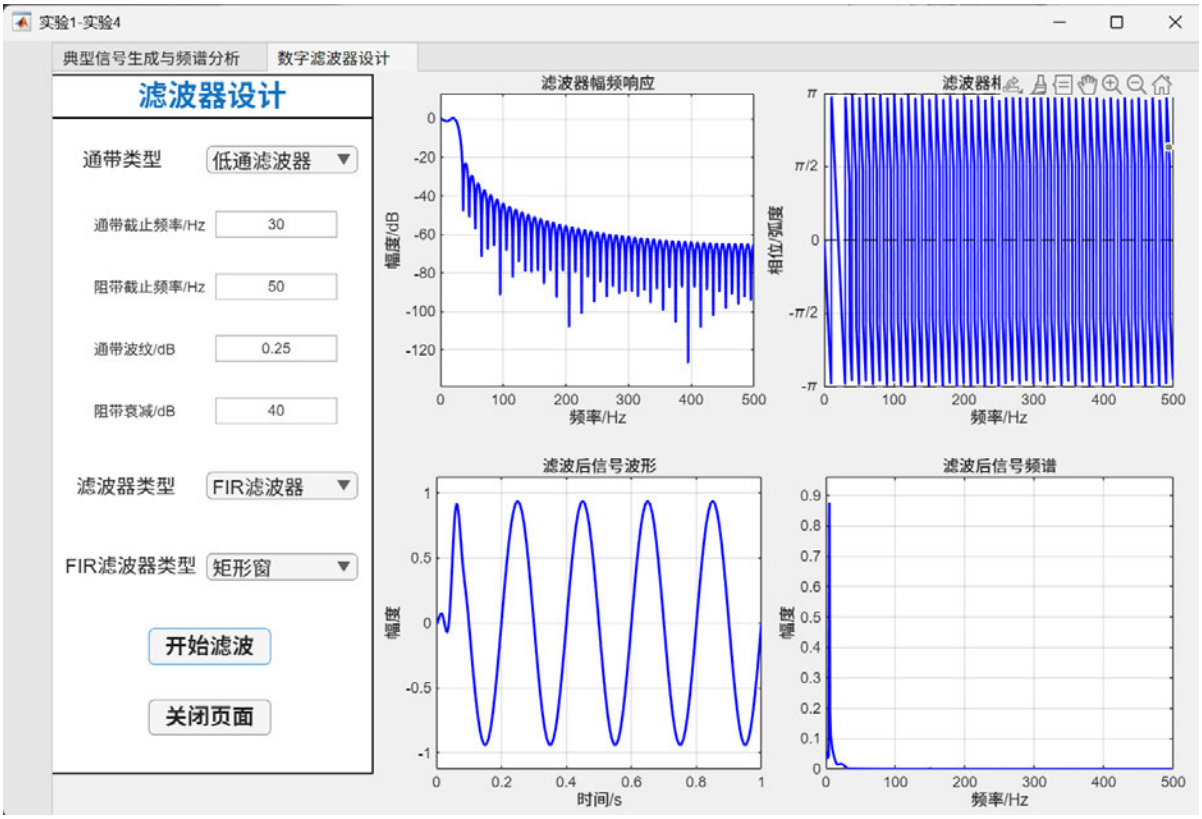

图 12：矩形窗低通滤波器设计

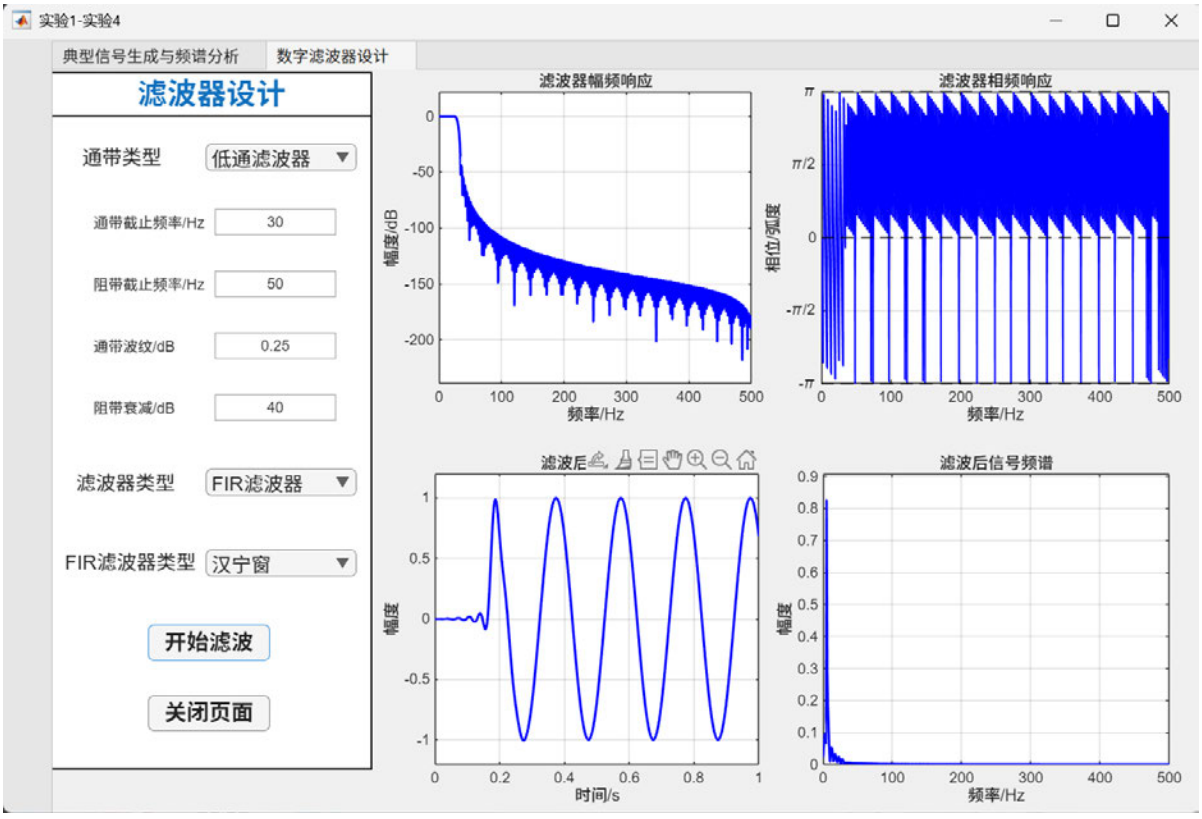

图 13：汉宁窗低通滤波器设计

观察上图可以发现，利用矩形窗和汉宁窗设计的低通滤波器最小阻带衰减分别在

-21dB 和-44dB 左右，与表 1 数据相符，同时设计的滤波器具有线性相位。滤波后的信号相比于原信号出现了一定的延迟，并且汉宁窗设计滤波器的延迟时间要显著长于矩形滤波器。但这是合理的，因为在滤波器的设计中为了保持线性相位加入了相移因子，这个相移因子会导致输出信号在时域上的延迟。同时为了满足过渡带宽的需求汉宁窗滤波器的长度要明显长于矩形窗滤波器，因此具有更长的时间延迟。观察后续所有的 FIR 滤波器滤波处理的结果，发现输出信号延迟的现象普遍存在。信号经两种 FIR 滤波器滤波后仅保留 5Hz 的频率分量，150Hz 频率分量被滤除，这进一步表明滤波器设计是合理的。

注意，低通滤波器输入的通带截止频率要小于阻带截止频率，否则系统会提示错误。

2.1.2. 高通滤波器

设置滤波器通带截止频率为 50Hz，阻带截止频率为 30Hz，通带波纹和阻带衰减保持默认数值。设计切比雪夫 I 型 IIR 滤波器对信号进行滤波处理，结果如下图所示：

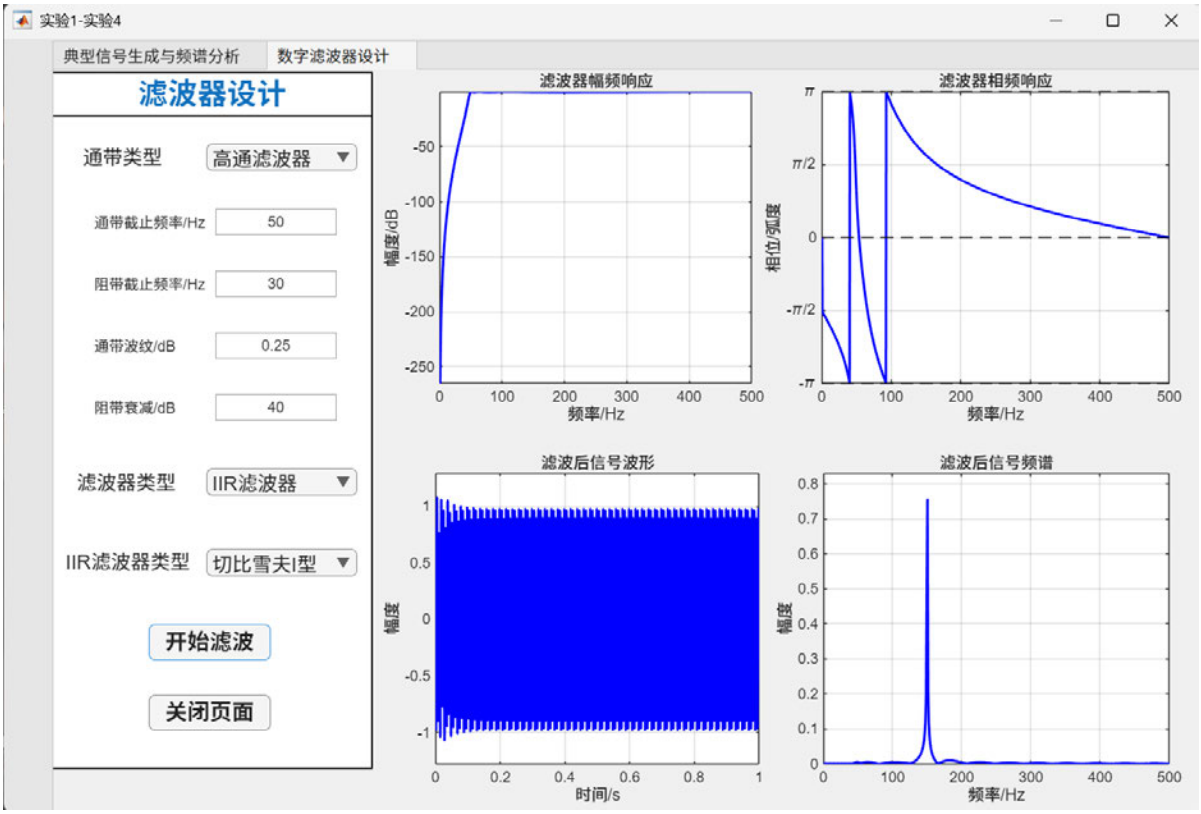

图 14：切比雪夫 I 型高通滤波器设计

观察上图可以发现，设计的切比雪夫 I 型高通滤波器幅频响应存在通带波纹，阻带单调递减，截止频率与输入数值相符。信号经滤波后仅保留 150Hz 的频率分量，5Hz 频率分量被滤除，这表明滤波器设计是合理的。

利用汉明窗设计 FIR 滤波器对信号进行滤波处理结果如下图所示：

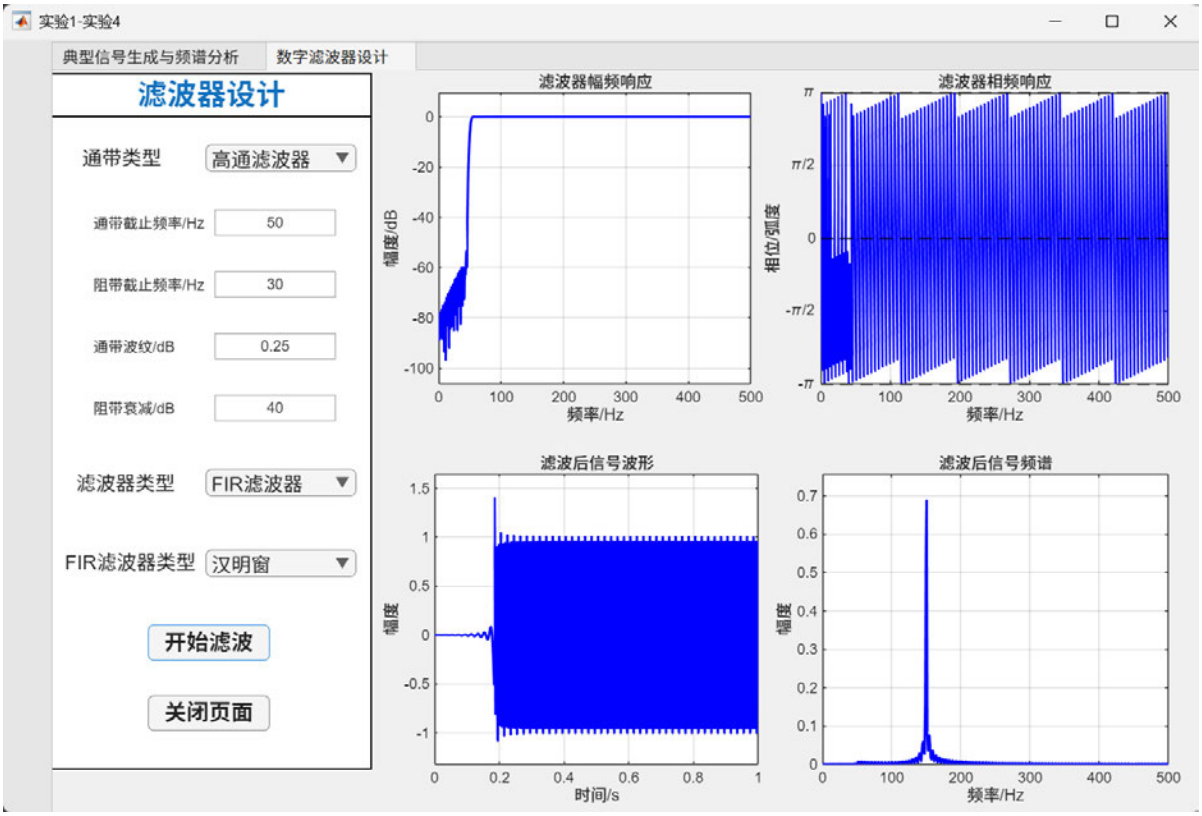

图 15：汉明窗高通滤波器设计

观察上图可以发现，利用汉明窗设计的高通滤波器最小阻带衰减在-53dB 左右，与表 1 数据相符，同时该滤波器具有线性相位，滤波后的信号相比于原信号出现了一定的延迟。信号经上述 FIR 滤波器滤波后仅保留 150Hz 的频率分量，5Hz 频率分量被滤除，这进一步表明滤波器设计是合理的。

注意，高通滤波器输入的通带截止频率要大于阻带截止频率，否则系统会提示错误。

2.1.3. 带通滤波器

滤波器通带截止频率、阻带截止频率、通带波纹和阻带衰减保持默认数值，设计切比雪夫 II 型 IIR 滤波器对信号进行滤波处理，结果如下图所示：

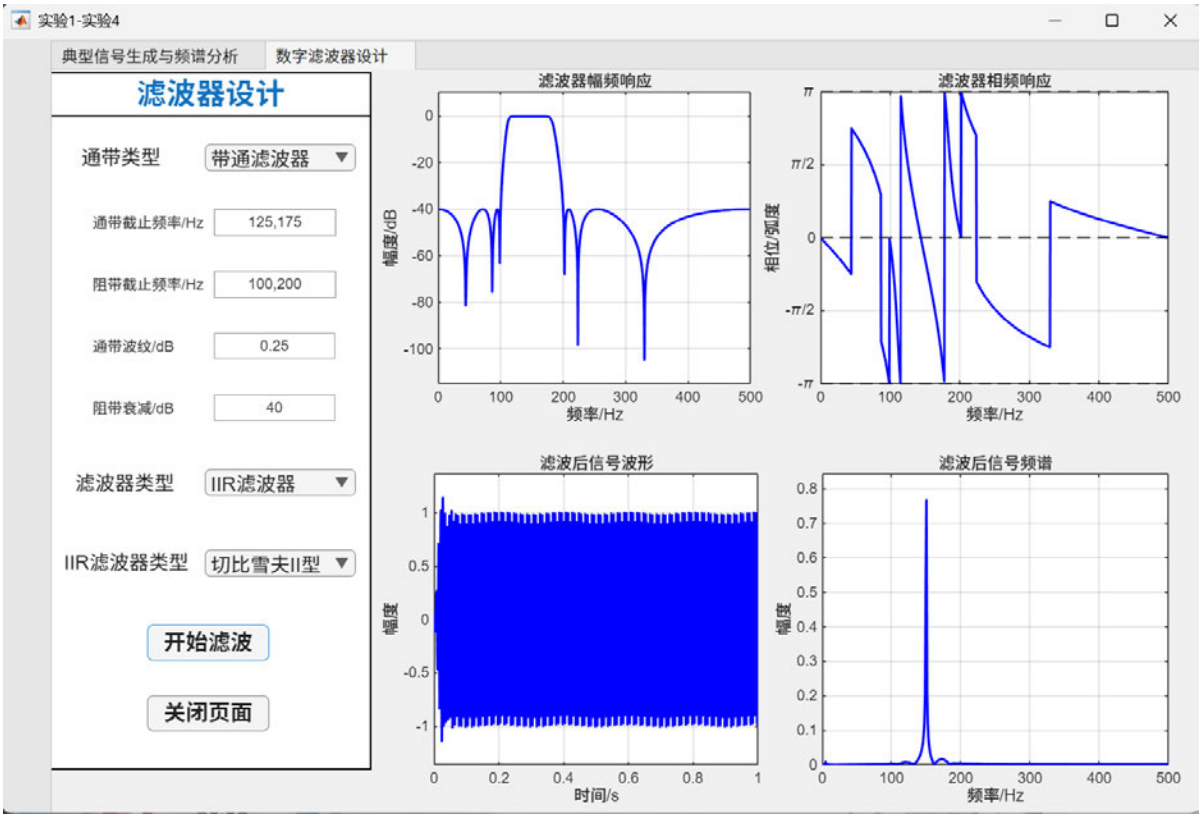

图 16：切比雪夫 II 型带通滤波器设计

观察上图可以发现，设计的切比雪夫 II 型带通滤波器幅频响应通带向两侧单调递减，阻带存在波纹，截止频率与输入数值相符。信号经滤波后仅保留 150Hz 的频率分量，5Hz 频率分量被滤除，这表明滤波器设计是合理的。

利用布莱克曼窗设计 FIR 滤波器对信号进行滤波处理结果如下图所示：

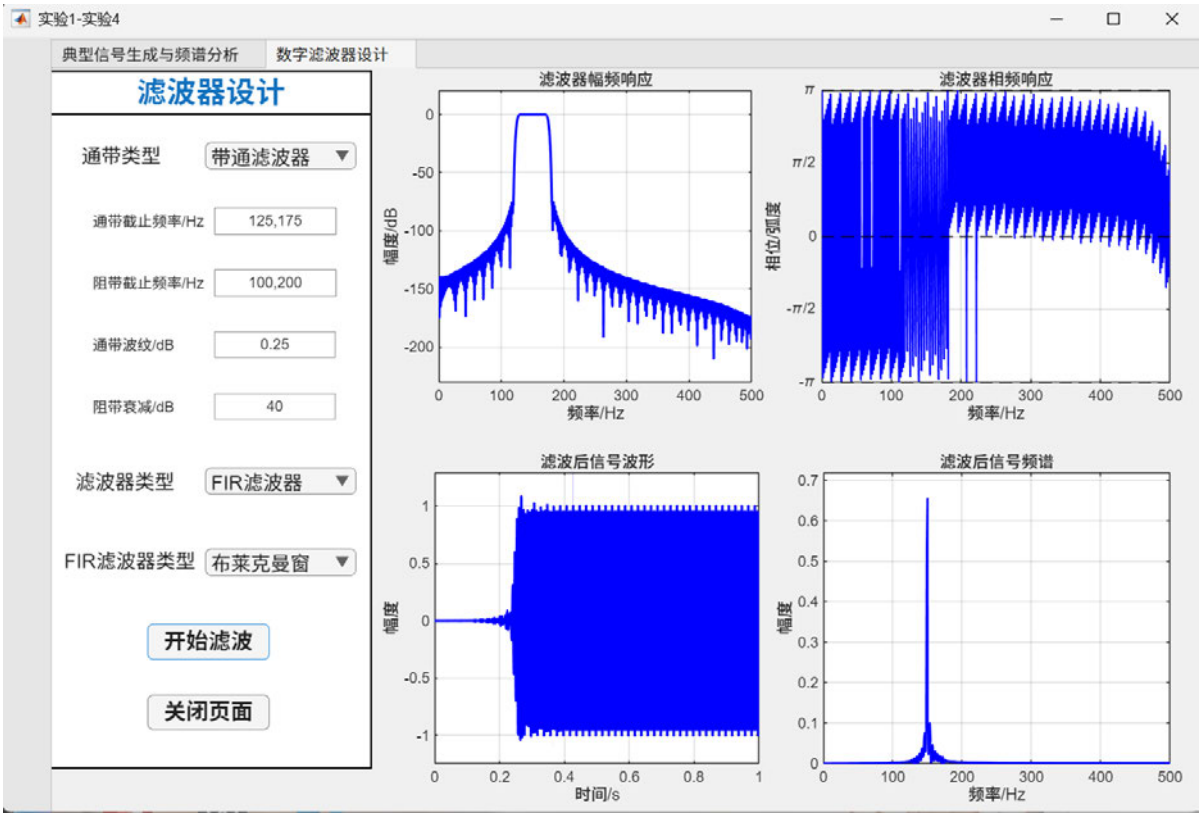

图 17：布莱克曼窗高通滤波器设计

观察上图可以发现，利用布莱克曼窗设计的带通滤波器最小阻带衰减在-74dB 左右，与表 1 数据相符，同时该滤波器具有线性相位。滤波后的信号相比于原信号出现了一定的延迟，信号经上述 FIR 滤波器滤波后仅保留 150Hz 的频率分量，5Hz 频率分量被滤除，这进一步表明滤波器设计是合理的。

注意，带通滤波器输入的通带截止频率要在阻带截止频率之内，否则系统会提示错误。

2.1.4. 带阻滤波器

将上述带通滤波器的通带和阻带的频率范围交换，通带波纹和阻带衰减保持默认数值，设计椭圆型 IIR 滤波器对信号进行滤波处理结果如下图所示：

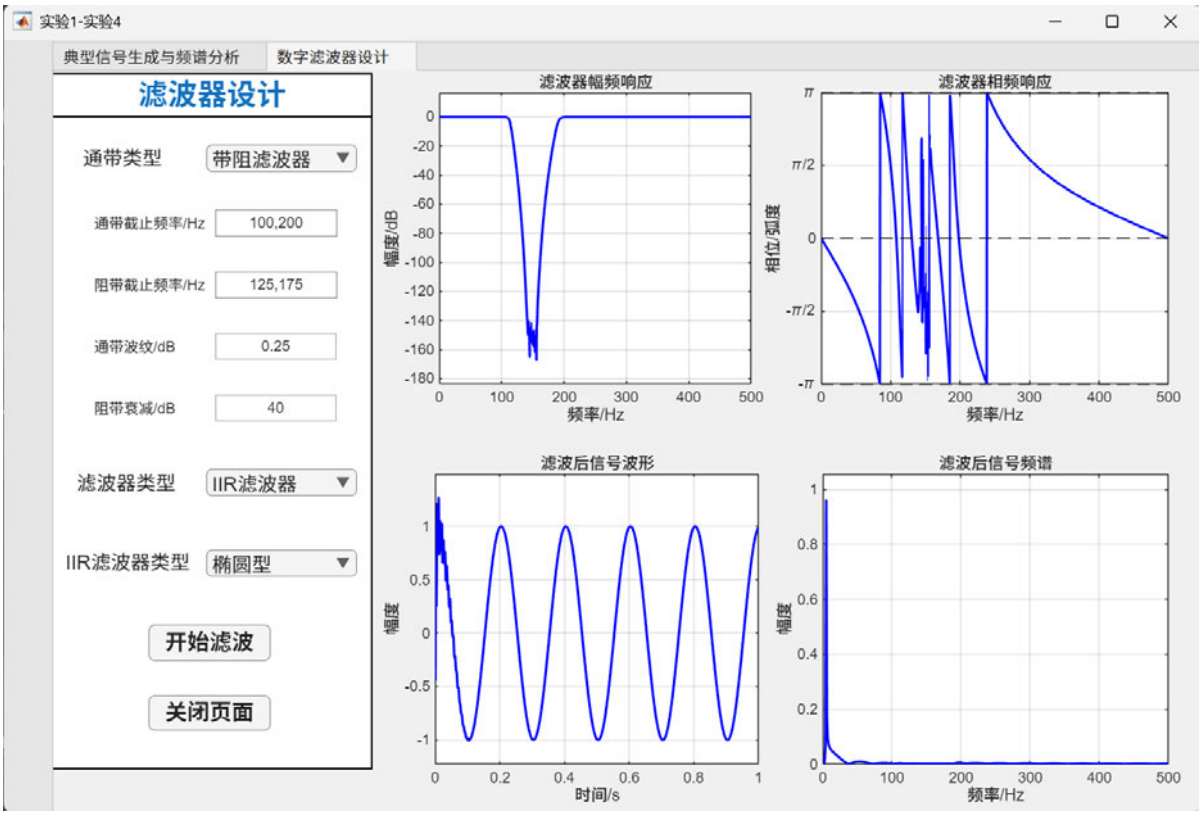

图 18：椭圆型带阻滤波器设计

观察上图可以发现，设计的椭圆型带阻滤波器幅频响应通带和阻带内均存在波纹，截止频率与输入数值相符。信号经滤波后仅保留 5Hz 的频率分量，150Hz 频率分量被滤除，这表明滤波器设计是合理的。

利用凯瑟窗设计 FIR 滤波器对信号进行滤波处理，结果如下图所示：

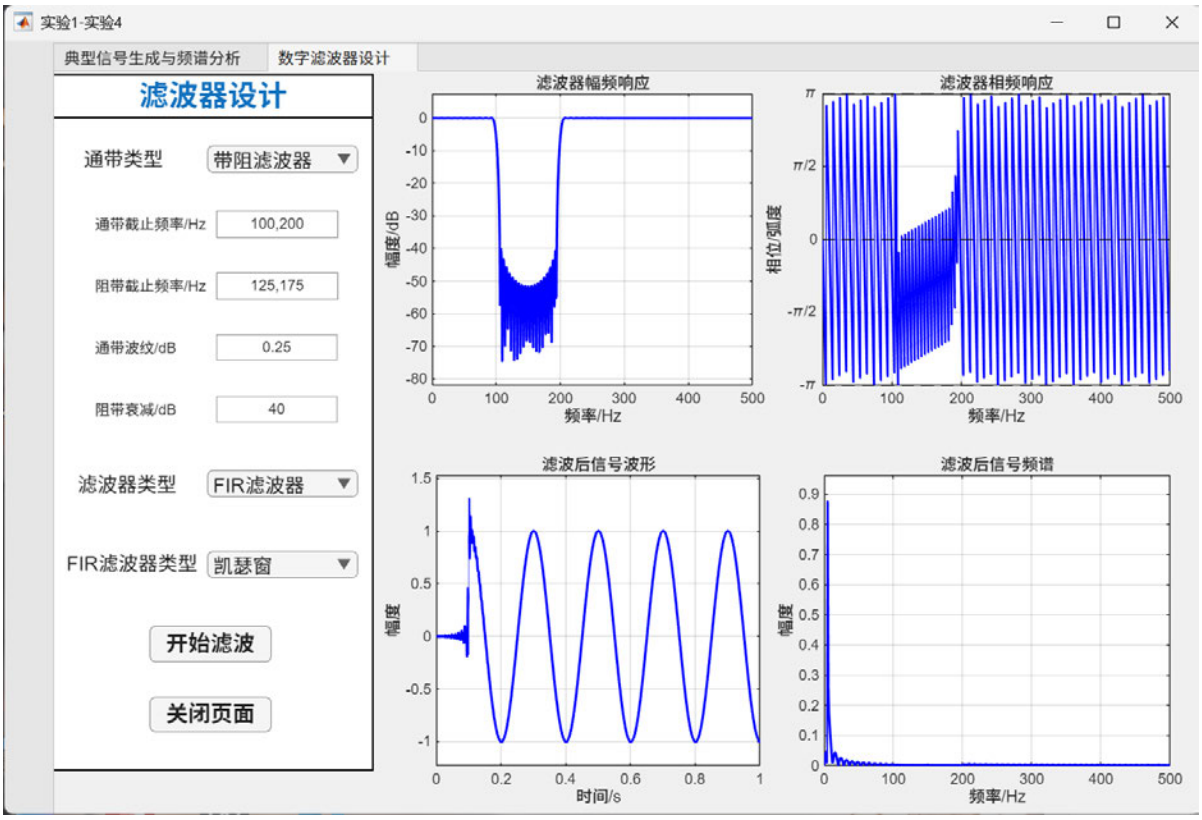

图 19：凯瑟窗带阻滤波器设计

观察上图可以发现，凯瑟窗设计的带阻滤波器满足阻带衰减需求，同时该滤波器具有线性相位。滤波后的信号相比于原信号出现了一定的延迟，信号经上述 FIR 滤波器滤波后仅保留 5Hz 的频率分量，150Hz 频率分量被滤除，这进一步表明滤波器设计是合理的。

注意，带阻滤波器输入的通带截止频率要在阻带截止频率之外，否则系统会提示错误。

2.2. 典型信号滤波分析

以上滤波器测试环节对不同通带类型的所有 IIR、FIR 滤波器类型进行了测试，均能很好地完成信号的滤波处理任务，这表明了该信号处理系统滤波器设计模块的合理性与正确性。为了体现滤波器设计的核心价值，现设计滤波器完成信号滤波处理和方波信号滤波分析的任务。后续的探究均使用切比雪夫 I 型 IIR 滤波器进行滤波处理，通带波纹和阻带衰减均保持默认值，实际应用中可根据需求改变滤波器种类与参数。

2.2.1. 有噪正弦波信号的降噪处理

通过 1.5 部分的分析可知，噪声的频带范围很宽，从低频到高频一直都有分布。而有用信号往往只分布于一段较窄的频带内，因此可以用滤波的方法滤除信号频带外的噪声。为简单起见并方便观察滤波降噪的效果，这里仅探究 1.5 部分中不同信噪比下正

弦波加高斯白噪声的滤波降噪。由于有用信号的频率较低，因此可直接使用低通滤波器进行滤波处理。

对于噪声方差为0.01的高信噪比信号，如果使用通带、阻带截止频率分别为100Hz、200Hz的低通滤波器进行滤波处理，结果如下图所示：

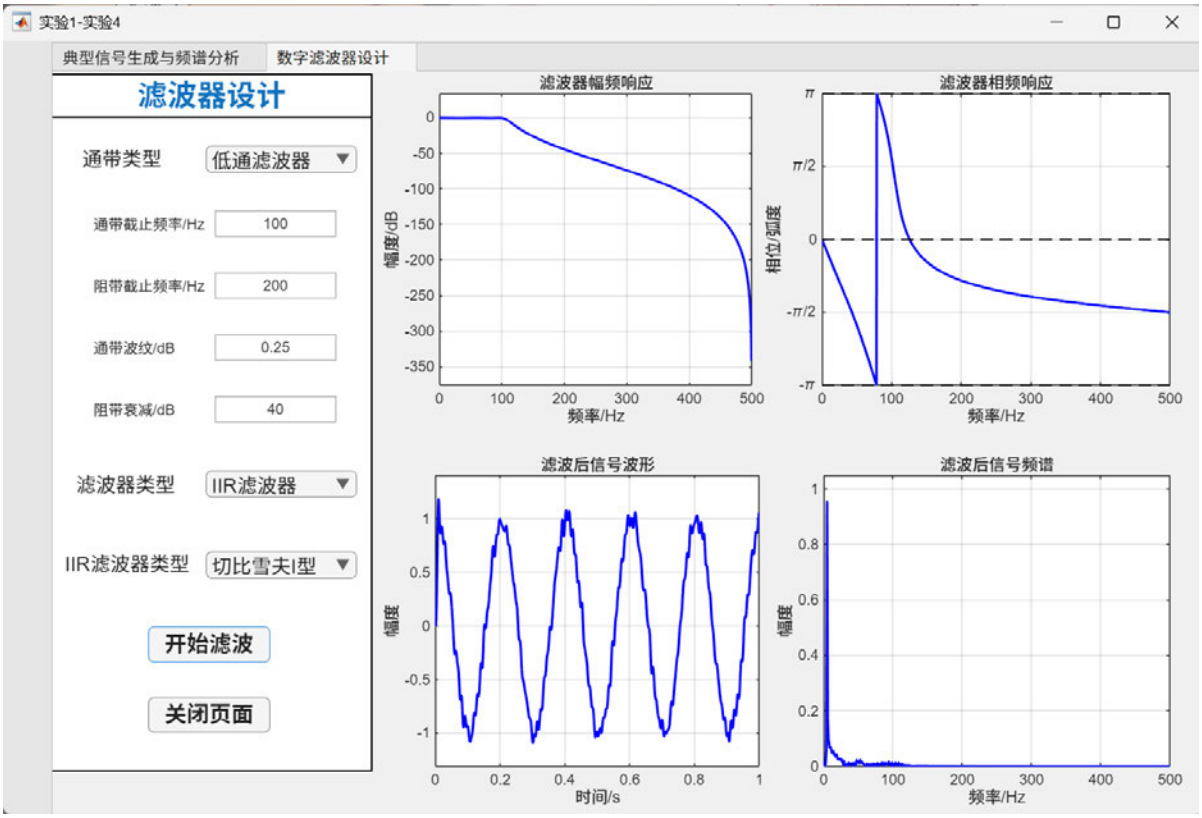

图 20：高信噪比正弦波信号的滤波降噪处理(截止频率较高)

将上述滤波处理后的结果与图 9 进行对比，可发现信号明显平滑了很多，说明滤波降噪是有效的。但信号的波形仍存在一定的毛刺，这是因为滤波器的通带带宽较大，并没有将信号的频带外噪声彻底滤除。如果进一步收缩低通滤波器的通带带宽，将通带、阻带截止频率分别调整为 30Hz、50Hz，滤波处理后的结果如下图所示：

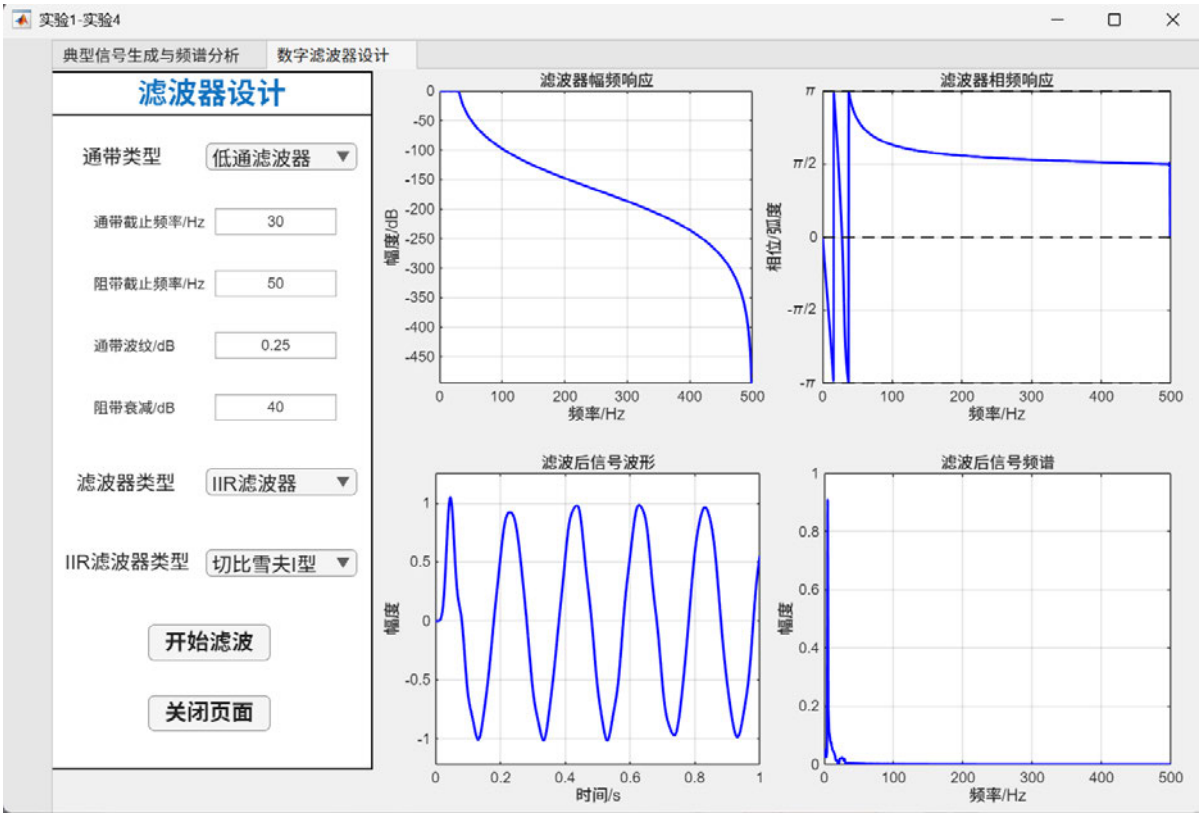

图 21：高信噪比正弦波信号的滤波降噪处理(截止频率较低)

可以发现此时信号的波形已经非常接近标准正弦波，信号波形几乎不存在毛刺，信号仅发生小幅度的失真。这说明要提高滤波降噪的效果，滤波器的通带范围应尽可能窄，以便有效滤除频带外的噪声分量，但同时要注意不要滤除有用信号的频率分量。以上是对 5Hz 低频信号的分析，而对于频带分布在高频区域的带限信号，一般使用带通滤波器进行滤波降噪处理，通带设置为有用信号的频带范围即可。

作为对比，再次使用刚才设计的低通滤波器对噪声方差为 5 的低信噪比信号进行滤波处理，结果如下图所示：

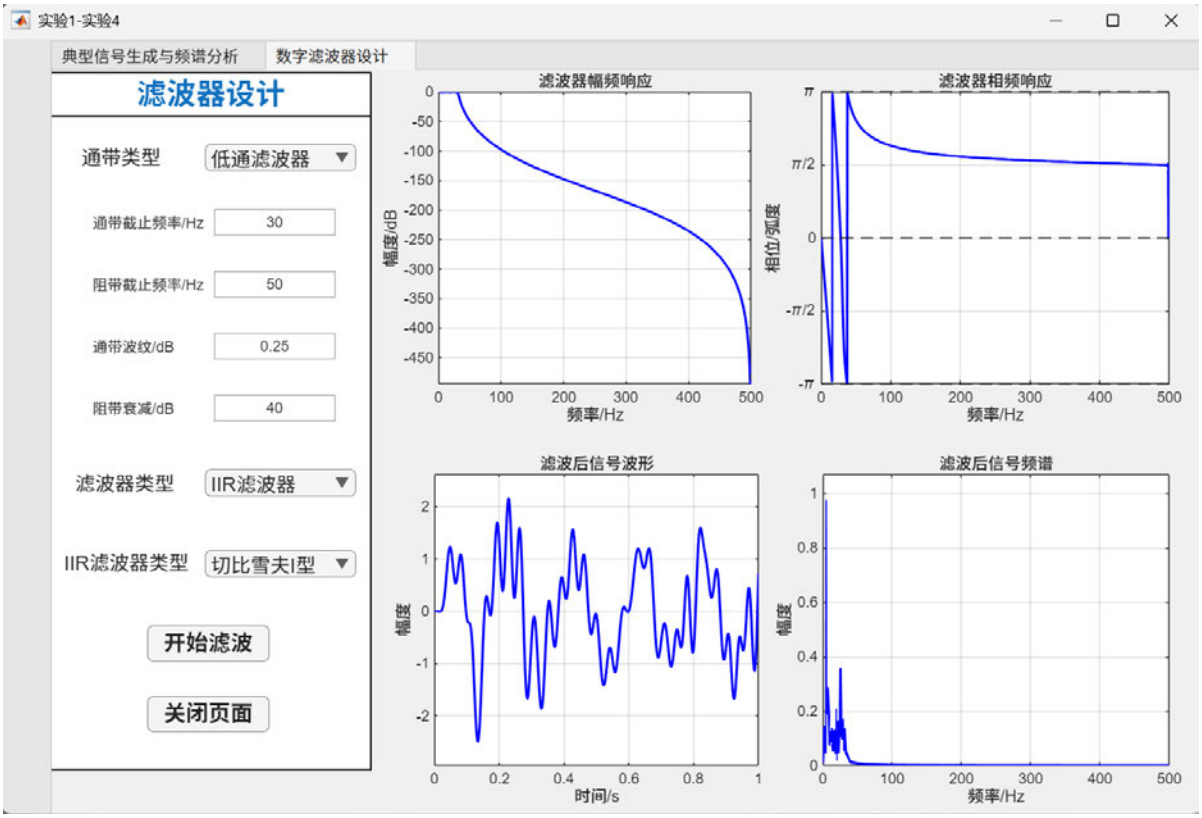

图 22：低信噪比正弦波信号的滤波降噪处理

观察上图可知，即使进行滤波降噪处理，信号波形也几乎完全失真。这说明对于低信噪比的信号，仅通过简单的滤波降噪无法有效地滤除噪声干扰，还必须使用其他更为先进的信号处理方法进行降噪处理。

2.2.2. 方波信号滤波分析

在接下来的实验中分别使用不同截止频率的滤波器对方波信号进行滤波处理，观察滤波后的效果。方波信号使用 1.2 部分的测试案例，如果使用通带、阻带截止频率分别为 30Hz、50Hz 的低通滤波器进行滤波处理，结果如下图所示：

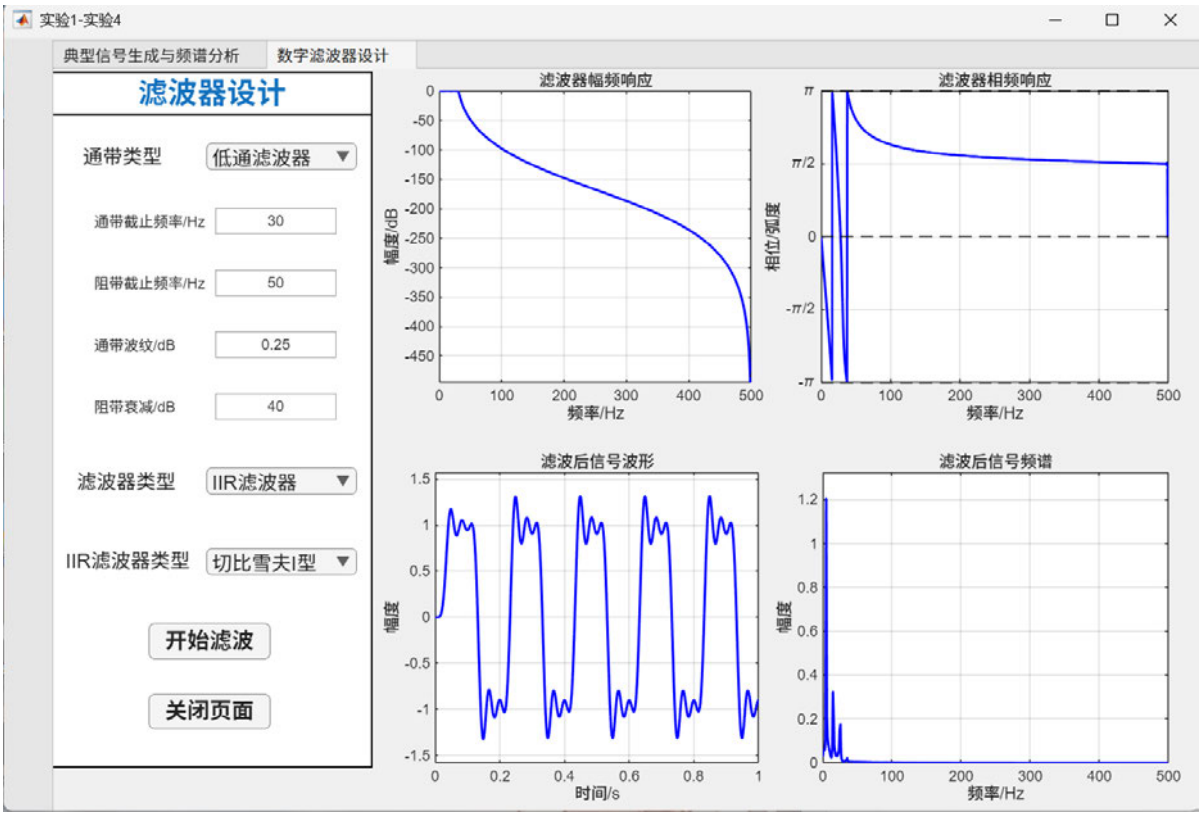

图 23：方波信号的滤波处理(截止频率较低)

可以发现滤波后的信号产生了明显的波纹，这是因为信号的高频分量被滤除，从而导致信号波形出现了失真。将低通滤波器通带、阻带截止频率分别调整为 100Hz、150Hz 的低通滤波器进行滤波处理，结果如下图所示：

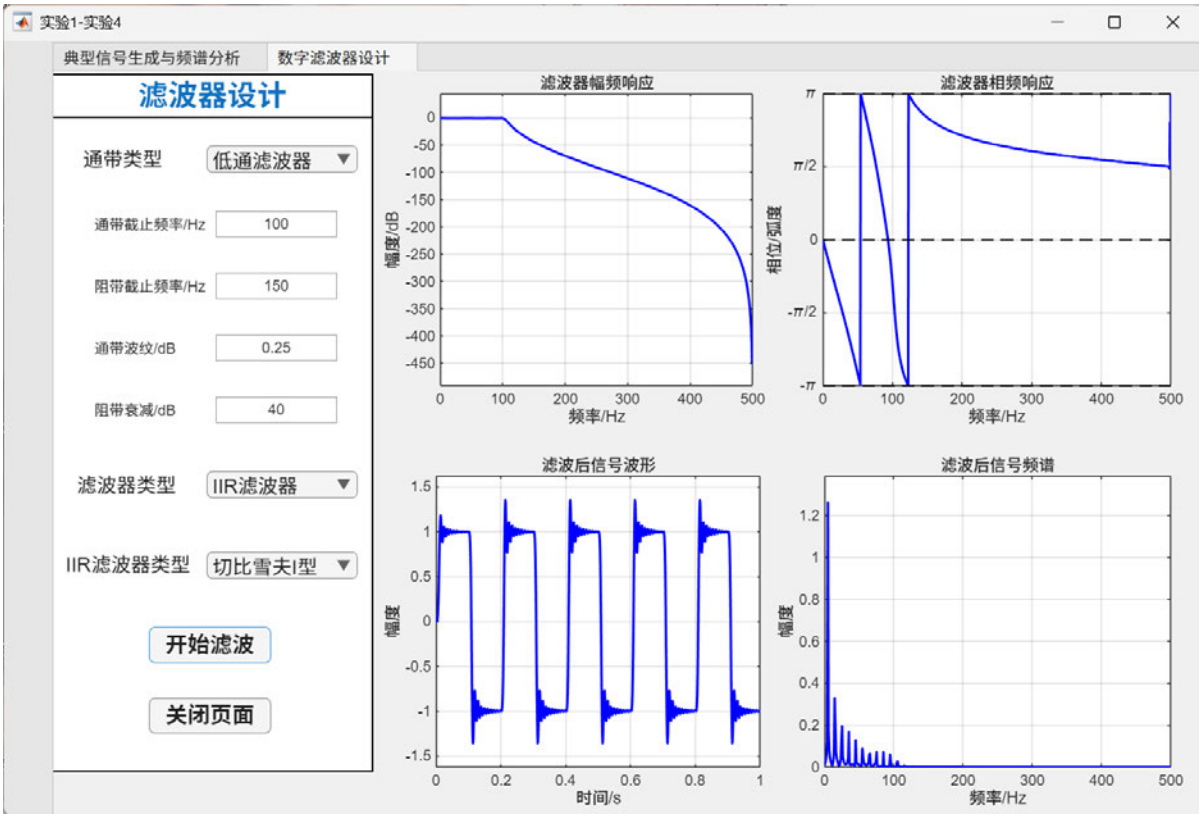

图 24：方波信号的滤波处理(截止频率较高)

可以发现此时信号已经比较接近原信号，但是在信号的跳变点处波纹存在过冲，且过冲的相对幅度随着滤波器截止频率的升高不发生改变，这就是吉布斯现象，与方波信号的傅里叶级数理论分析相符。

四、 总结与体会

通过本次典型信号生成、频谱分析及数字滤波器设计的实验，我收获了扎实的信号处理专业知识与宝贵的 MATLAB App 开发实践经验，对数字信号处理的核心原理有了更为透彻的理解。实验过程中，我不仅深入掌握了正弦波、方波、混合正弦波等典型信号的数学模型、频谱和功率谱分析原理，还熟练运用 MATLAB App Designer 完成了界面控件交互、动态参数显隐、信号波形和频谱的绘制及 IIR/FIR 滤波器设计等软件功能开发

其中，MATLAB APP 设计是我在这个实验中遇到的最大困难。第一个问题是不同控件的动态显隐出现了严重冲突。在开发初期因未梳理清楚不同信号专用参数的对应关系，导致在切换信号类型时出现专用参数编辑字段重叠和功能错乱的问题。最后我通过反复调试代码，明确不同信号的专用参数对应关系，在信号类型下拉框的回调函数中逐一设置信号类型切换时各部件的显隐关系，最终解决这个问题。第二个问题是跨功能模块的数据同步问题。在开发初期信号生成模块中的原始信号和采样频率无法

传递到信号滤波模块，导致无法对产生的信号进行滤波处理。为解决这一问题，我在APP 的私有属性中添加变量 generatedSignal 和 generatedFs，将信号生成模块的生成信号和采样频率分别存放在这两个变量中，通过这两个私有变量即可完成参数的传递，实现对生成信号的滤波处理。

本次实验让我深刻体会到理论与实践结合的重要意义和价值。之前我在课堂上学习的 FFT 频谱分析、滤波器设计等数字信号处理技术，通过这款信号处理系统的开发得到了全面应用。与此同时我还在实验中验证了数字滤波器对高斯白噪声的去除效果，观察到了频谱泄漏和吉布斯现象的发生，进一步加深了对相关信号处理理论的感受与理解，深刻体会到数字信号处理技术在实际中的重要应用价值。

最后，我要衷心感谢[redacted]的悉心指导。[redacted]对信号谱分析原理、IIR/FIR 滤波器设计原理和 MATLAB APP 开发流程的细致讲解，帮助我快速理清软件系统的开发思路。通过本次实验，我不仅夯实了数字信号处理的基础理论知识，积累了 MATLAB App 开发实践经验，更培养了科学严谨的实验态度和自主调试的实践思维。这些收获将为我后续深入学习数字信号处理进阶算法、开展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。