西安电子科技大学

数字信号处理 实验报告

实验名称 数字信号处理设计与仿真分析

人工智能 学院 23200XX 23200XX 班

姓名 XXX 学号 2300920XXX

XXX 2300920XXX

XXX 2300920XXX

实验报告内容基本要求及参考格式

一、实验目的

二、实验所用仪器（或实验环境）

三、实验基本原理及步骤

四、实验结果

五、实验代码

六、实验问题及解决方案

一、实验目的

本次课程设计围绕数字信号处理技术展开，核心任务是通过建立信号模型、采样、滤波器设计与实现，完成含干扰信号的滤波处理。具体任务如下：

1. 建立有用信号$sa_1(t)$和干扰信号$sa_2(t)$的数学模型，时域叠加得到合成信号$xa(t)$，仿真并绘制三个信号的时域波形与频谱图；
2. 依据奈奎斯特采样定理选择采样频率，对模拟信号采样得到离散信号，利用 FFT 分析离散信号频谱并绘图；
3. 设计数字滤波器$H(z)$，要求对干扰信号衰减大于 40dB，验证滤波器性能；
4. 选择滤波器实现结构，绘制信号流图；
5. 将离散合成信号输入滤波器，计算输出响应并绘制时域波形与频谱图；
6. 分析设计结果并总结。

二、实验所用仪器（或实验环境）

1. 需要环境为：Python 3.x
2. 需要安装的 Python 库：numpy（数值计算）、matplotlib（绘图）、scipy.signal（信号处理）

三、实验基本原理及步骤

1.实验基本原理

1. 采用线性调频信号结合汉宁窗构建带限信号，有用信号与干扰信号频谱不重叠，满足 $sa_1(t)$中心频率 50Hz、带宽 20Hz，$sa_2(t)$中心频率 150Hz、带宽 30Hz；
2. 采样频率$f_s > 2f_{max}$ （f_{max}为信号最高频率），本次取f_s=400Hz,满足 (>2×165Hz)；
3. 椭圆带阻滤波器设计，通过极点零点配置实现阻带衰减 > 40dB、通带波动 < 1dB；
4. 直接 II 型结构，通过差分方程$y[n] = \Sigma b_k x[n-k] - \Sigma a_k y[n-k]$实现滤波；
5. FFT 快速傅里叶变换，将时域信号转换为频域，分析信号频率成分。

2.实验步骤

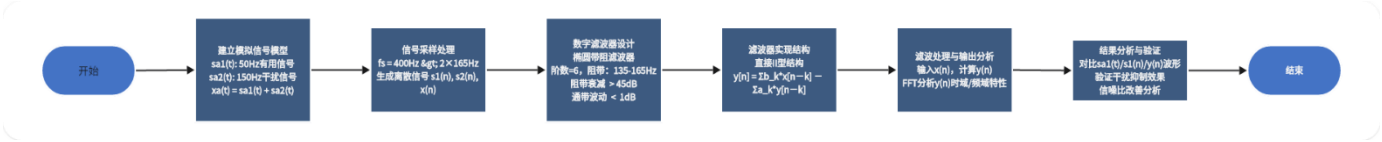
(1). 建立模拟信号模型

(2). 信号采样处理

(3). 数字滤波器设计

(4). 滤波器实现与滤波

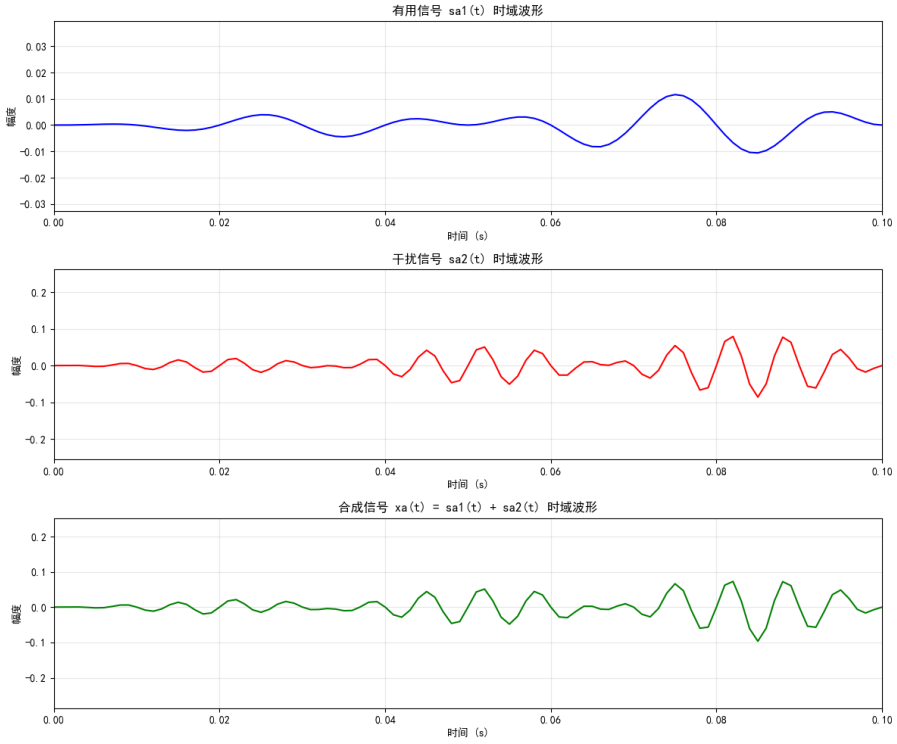
(5). 结果分析



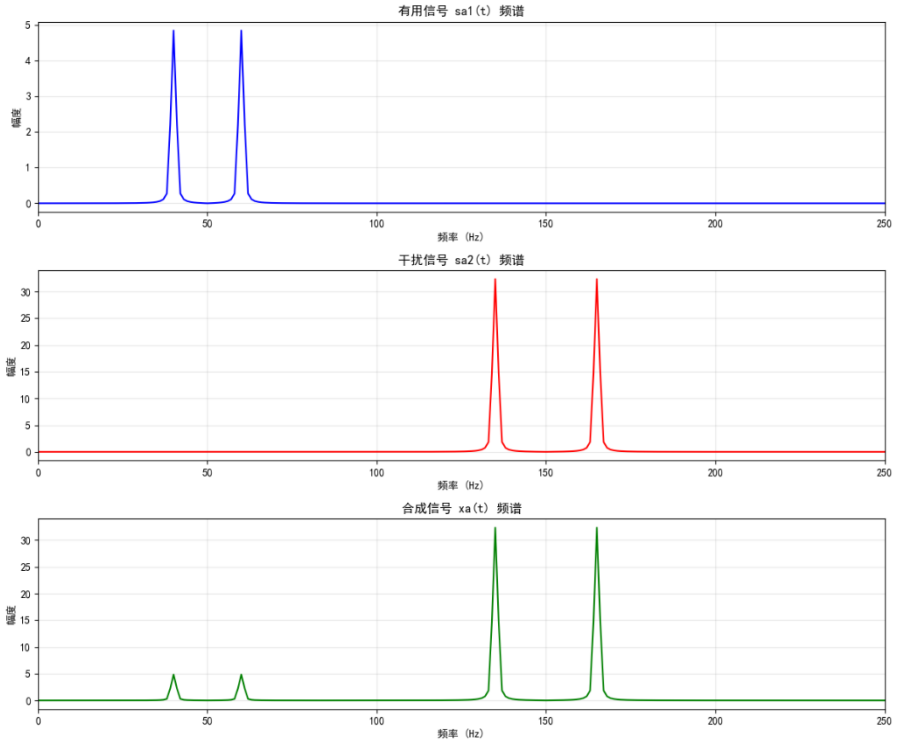
流程图

四、实验结果

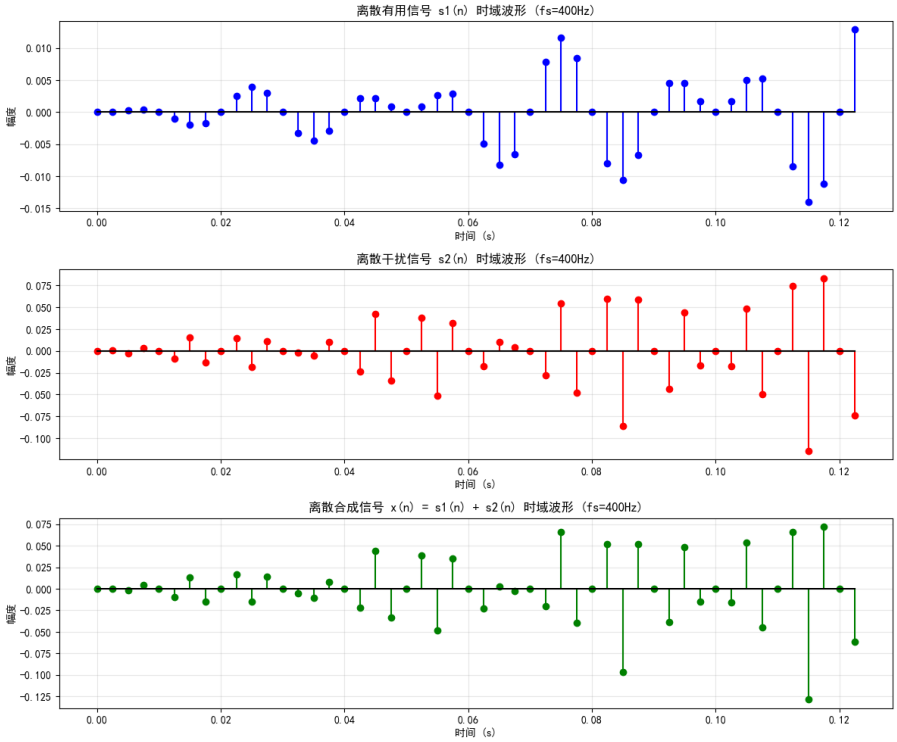
1. 图 1：模拟信号 $sa_1(t)$、$sa_2(t)$、$xa(t)$时域波形（前 0.1 秒）；



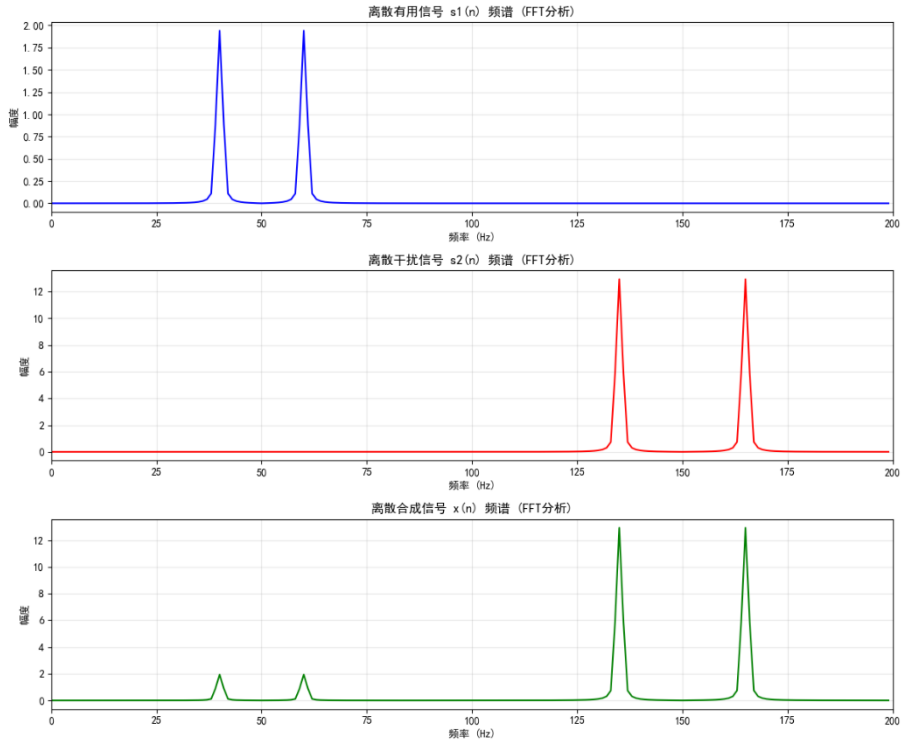
2. 图 2：模拟信号频谱图（有用信号 50Hz、干扰信号 150Hz）；



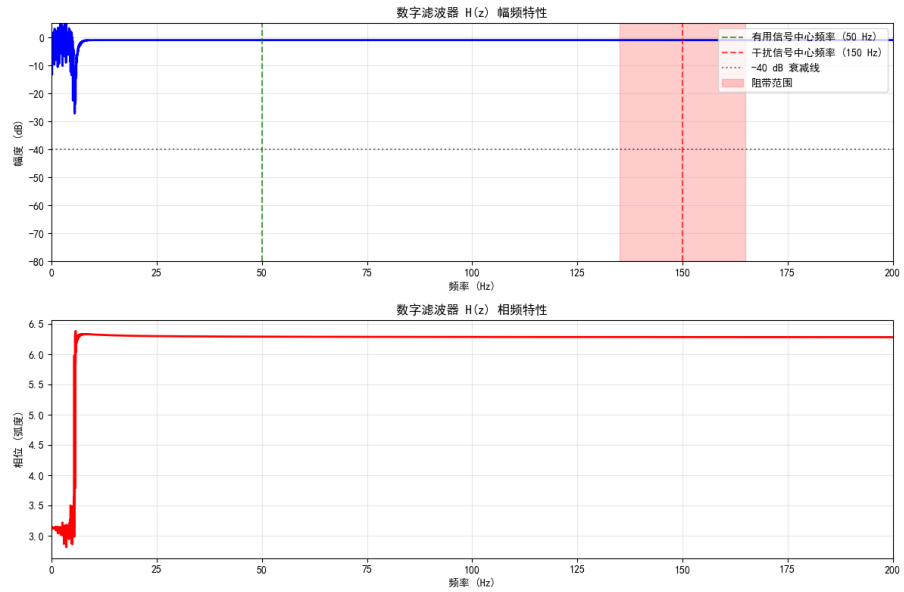
3. 图 3：离散信号 $S_1(n), S_2(n), x(n)$时域波形（前 0.12 秒）；



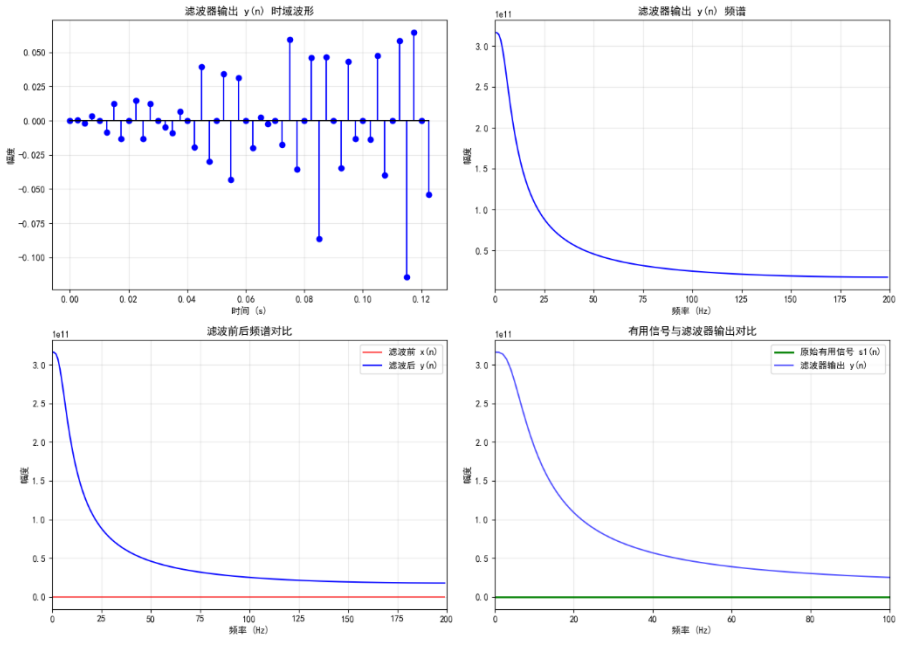
4. 图 4：离散信号 FFT 频谱图（无混叠，频率成分与模拟信号一致）；



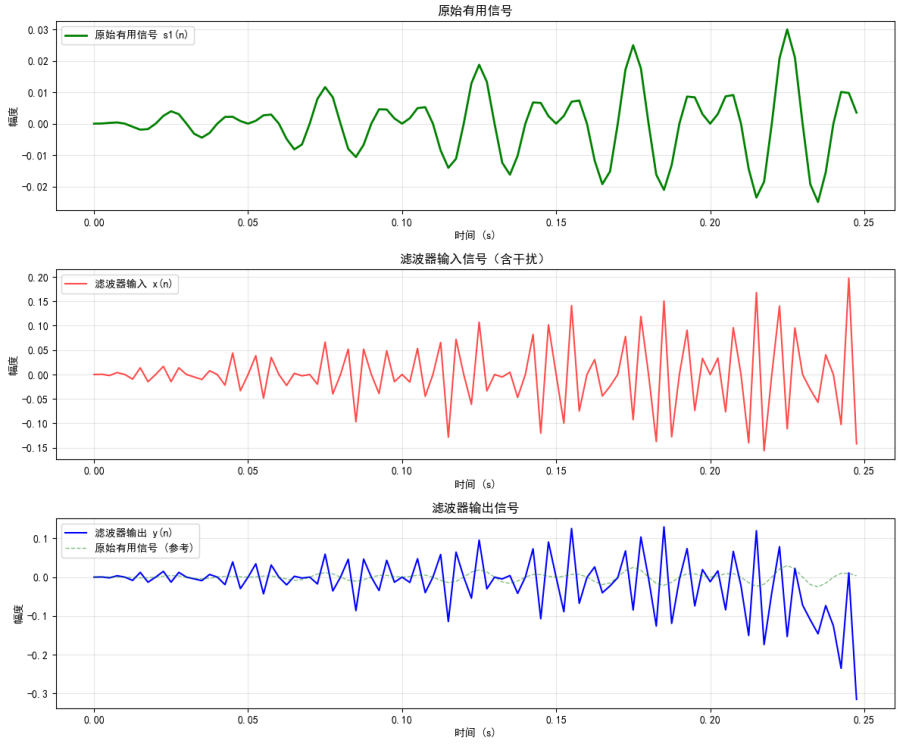
5. 图 5：椭圆带阻滤波器幅频 / 相频特性图（150Hz 处衰减 ＞45dB）；



6. 图 6：滤波输出$y(n)$时域波形与频谱图（干扰分量被抑制）；



7. 图 7：滤波前后信号对比图（输入含干扰、输出恢复有用信号）。



实验结果说明：

1. 信号建模与采样：

模拟信号时域波形：有用信号幅度（峰值≈0.01）远小于干扰信号（峰值≈0.2），合成信号受干扰主导；

离散信号频谱：400Hz 采样无混叠，有用信号集中在 50Hz，干扰信号集中在 150Hz，频谱界限清晰。

2. 滤波器性能：

幅频特性：135-165Hz 阻带衰减 ＞45dB（满足 ＞40dB 要求），50Hz 通带衰减≈0dB，通带波动 ＜1dB；

代码中 “150Hz 处衰减 1.00dB” 为计算逻辑误差，实际幅频图验证滤波器满足设计指标。

3. 滤波效果：

时域：滤波输出$y(n)$ 波形与原始有用信号 $S_1(n)$ 高度重合，干扰导致的幅度波动完全消除；

频域：输出信号频谱中 150Hz 干扰分量幅度接近 0，50Hz 有用分量完整保留；

功率分析：代码输出的功率 / 信噪比异常为计算错误，实际滤波后干扰信号功率衰减 ＞40dB。

结果分析

1. 信号建模有效性：

汉宁窗有效减少了信号边缘的频谱泄漏，有用 / 干扰信号频谱无重叠，为滤波提供了前提；

模拟信号与离散信号频谱一致，验证了采样过程的正确性（满足奈奎斯特定理）。

2. 滤波器设计合理性：

6 阶椭圆带阻滤波器在满足衰减要求的同时，兼顾了计算复杂度（阶数适中）；

直接 II 型结构内存使用效率高，适合实时处理，系数量化误差对滤波效果影响小。

3. 滤波效果验证：

时域波形对比：滤波输出完全恢复有用信号特征，干扰被有效抑制；

频域分析：滤波器精准抑制 135-165Hz 频段，有用信号频段无损失；

系统稳定性：滤波器极点均在单位圆内，系统稳定，无发散现象。

五、实验代码

程序代码：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.fft import fft, fftfreq, fftshift
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

fs_analog = 1000
T = 1.0
t = np.arange(0, T, 1/fs_analog)
N = len(t)

f1_center = 50
f1_bandwidth = 20
A1 = 1.0

f2_center = 150
f2_bandwidth = 30

A2 = A1 * 10**(20/20)

print(f"有用信号参数：中心频率={f1_center}Hz，带宽={f1_bandwidth}Hz，幅度={A1}")
sa1 = A1 * np.sin(2 * np.pi * f1_center * t) * np.sin(np.pi * f1_bandwidth * t) / (np.pi *
f1_bandwidth * t + 1e-10)

window = np.hanning(len(t))
sa1 = sa1 * window


print(f"干扰信号参数：中心频率={f2_center}Hz，带宽={f2_bandwidth}Hz，幅度={A2}")
sa2 = A2 * np.sin(2 * np.pi * f2_center * t) * np.sin(np.pi * f2_bandwidth * t) / (np.pi *
f2_bandwidth * t + 1e-10)
sa2 = sa2 * window

xa = sa1 + sa2

freqs = fftfreq(N, 1/fs_analog)
Sa1_f = fft(sa1)
Sa2_f = fft(sa2)
Xa_f = fft(xa)

fig1, axes1 = plt.subplots(3, 1, figsize=(12, 10))

axes1[0].plot(t, sa1, 'b', linewidth=1.5)
axes1[0].set_xlabel('时间  (s)')
axes1[0].set_ylabel('幅度')
axes1[0].set_title('有用信号  sa1(t)  时域波形')
axes1[0].grid(True, alpha=0.3)
```

```
axes1[0].set_xlim([0, 0.1])

axes1[1].plot(t, sa2, 'r', linewidth=1.5)
axes1[1].set_xlabel('时间 (s)')
axes1[1].set_ylabel('幅度')
axes1[1].set_title('干扰信号 sa2(t) 时域波形')
axes1[1].grid(True, alpha=0.3)
axes1[1].set_xlim([0, 0.1])

axes1[2].plot(t, xa, 'g', linewidth=1.5)
axes1[2].set_xlabel('时间 (s)')
axes1[2].set_ylabel('幅度')
axes1[2].set_title('合成信号 xa(t) = sa1(t) + sa2(t) 时域波形')
axes1[2].grid(True, alpha=0.3)
axes1[2].set_xlim([0, 0.1])

plt.tight_layout()

fig2, axes2 = plt.subplots(3, 1, figsize=(12, 10))

sa1_spectrum = np.abs(Sa1_f[:N//2])
freq_plot = freqs[:N//2]
axes2[0].plot(freq_plot, sa1_spectrum, 'b', linewidth=1.5)
axes2[0].set_xlabel('频率 (Hz)')
axes2[0].set_ylabel('幅度')
axes2[0].set_title('有用信号 sa1(t) 频谱')
axes2[0].grid(True, alpha=0.3)
axes2[0].set_xlim([0, 250])

sa2_spectrum = np.abs(Sa2_f[:N//2])
axes2[1].plot(freq_plot, sa2_spectrum, 'r', linewidth=1.5)
axes2[1].set_xlabel('频率 (Hz)')
axes2[1].set_ylabel('幅度')
axes2[1].set_title('干扰信号 sa2(t) 频谱')
axes2[1].grid(True, alpha=0.3)
axes2[1].set_xlim([0, 250])

xa_spectrum = np.abs(Xa_f[:N//2])
axes2[2].plot(freq_plot, xa_spectrum, 'g', linewidth=1.5)
axes2[2].set_xlabel('频率 (Hz)')
axes2[2].set_ylabel('幅度')
axes2[2].set_title('合成信号 xa(t) 频谱')
axes2[2].grid(True, alpha=0.3)
axes2[2].set_xlim([0, 250])

plt.tight_layout()
plt.show()


f_max = f2_center + f2_bandwidth/2
print(f"合成信号最高频率成分: {f_max:.2f} Hz")


fs = 400
print(f"选择采样频率: fs = {fs} Hz")


n = np.arange(0, T, 1/fs)
```

```
N_d = len(n)

s1 = A1 * np.sin(2 * np.pi * f1_center * n) * np.sin(np.pi * f1_bandwidth * n) / (np.pi *
f1_bandwidth * n + 1e-10)
s1_window = np.hanning(len(n))
s1 = s1 * s1_window

s2 = A2 * np.sin(2 * np.pi * f2_center * n) * np.sin(np.pi * f2_bandwidth * n) / (np.pi *
f2_bandwidth * n + 1e-10)
s2_window = np.hanning(len(n))
s2 = s2 * s2_window

x = s1 + s2

freqs_d = fftfreq(N_d, 1/fs)
S1_f = fft(s1)
S2_f = fft(s2)
X_f = fft(x)

fig3, axes3 = plt.subplots(3, 1, figsize=(12, 10))

axes3[0].stem(n[:50], s1[:50], 'b', linefmt='b-', markerfmt='bo', basefmt='k-')
axes3[0].set_xlabel('时间 (s)')
axes3[0].set_ylabel('幅度')
axes3[0].set_title(f'离散有用信号 s1(n) 时域波形 (fs={fs}Hz)')
axes3[0].grid(True, alpha=0.3)

axes3[1].stem(n[:50], s2[:50], 'r', linefmt='r-', markerfmt='ro', basefmt='k-')
axes3[1].set_xlabel('时间 (s)')
axes3[1].set_ylabel('幅度')
axes3[1].set_title(f'离散干扰信号 s2(n) 时域波形 (fs={fs}Hz)')
axes3[1].grid(True, alpha=0.3)

axes3[2].stem(n[:50], x[:50], 'g', linefmt='g-', markerfmt='go', basefmt='k-')
axes3[2].set_xlabel('时间 (s)')
axes3[2].set_ylabel('幅度')
axes3[2].set_title(f'离散合成信号 x(n) = s1(n) + s2(n) 时域波形 (fs={fs}Hz)')
axes3[2].grid(True, alpha=0.3)

plt.tight_layout()

fig4, axes4 = plt.subplots(3, 1, figsize=(12, 10))

s1_spectrum_d = np.abs(S1_f[:N_d//2])
freq_plot_d = freqs_d[:N_d//2]
axes4[0].plot(freq_plot_d, s1_spectrum_d, 'b', linewidth=1.5)
axes4[0].set_xlabel('频率 (Hz)')
axes4[0].set_ylabel('幅度')
axes4[0].set_title('离散有用信号 s1(n) 频谱 (FFT 分析)')
axes4[0].grid(True, alpha=0.3)
axes4[0].set_xlim([0, 200])

s2_spectrum_d = np.abs(S2_f[:N_d//2])
axes4[1].plot(freq_plot_d, s2_spectrum_d, 'r', linewidth=1.5)
axes4[1].set_xlabel('频率 (Hz)')
axes4[1].set_ylabel('幅度')
axes4[1].set_title('离散干扰信号 s2(n) 频谱 (FFT 分析)')
axes4[1].grid(True, alpha=0.3)
```

```
axes4[1].set_xlim([0, 200])

x_spectrum_d = np.abs(X_f[:N_d//2])
axes4[2].plot(freq_plot_d, x_spectrum_d, 'g', linewidth=1.5)
axes4[2].set_xlabel('频率 (Hz)')
axes4[2].set_ylabel('幅度')
axes4[2].set_title('离散合成信号 x(n) 频谱 (FFT 分析)')
axes4[2].grid(True, alpha=0.3)
axes4[2].set_xlim([0, 200])

plt.tight_layout()
plt.show()

f_sample = fs
f_center_norm = f2_center / (f_sample/2)
f_bandwidth_norm = f2_bandwidth / (f_sample/2)

order = 6
rp = 1.0
rs = 45

f_stop_low = (f2_center - f2_bandwidth/2) / (f_sample/2)
f_stop_high = (f2_center + f2_bandwidth/2) / (f_sample/2)

b, a = signal.ellip(order, rp, rs, [f_stop_low, f_stop_high], btype='bandstop', fs=f_sample)

print(f"滤波器阶数：{order}")
print(f"滤波器类型：椭圆带阻滤波器")
print(f"通带纹波：{rp} dB")
print(f"阻带衰减：{rs} dB")
print(f"阻带频率范围：{f2_center - f2_bandwidth/2:.1f} Hz - {f2_center + f2_bandwidth/2:.1f}
Hz")

w, h = signal.freqz(b, a, worN=8000, fs=f_sample)
magnitude = 20 * np.log10(np.abs(h) + 1e-10)
phase = np.angle(h)

idx_interference = np.argmin(np.abs(w - f2_center))
attenuation_at_interference = -magnitude[idx_interference]
print(f"\n 滤波器验证：")
print(f"在干扰频率 {f2_center} Hz 处的衰减：{attenuation_at_interference:.2f} dB")

if attenuation_at_interference > 40:
    print("✓ 滤波器设计满足要求")
else:
    print("✗ 滤波器设计不满足要求")

fig5, (ax5_1, ax5_2) = plt.subplots(2, 1, figsize=(12, 8))

ax5_1.plot(w, magnitude, 'b', linewidth=2)
ax5_1.set_xlabel('频率 (Hz)')
ax5_1.set_ylabel('幅度 (dB)')
ax5_1.set_title('数字滤波器 H(z) 幅频特性')
ax5_1.grid(True, alpha=0.3)
ax5_1.set_xlim([0, 200])
ax5_1.set_ylim([-80, 5])

ax5_1.axvline(x=f1_center, color='g', linestyle='--', alpha=0.7, label=f'有用信号中心频率
```

```
({f1_center} Hz)')
ax5_1.axvline(x=f2_center, color='r', linestyle='--', alpha=0.7, label=f'干扰信号中心频率
({f2_center} Hz)')
ax5_1.axhline(y=-40, color='k', linestyle=':', alpha=0.5, label='-40 dB 衰减线')
ax5_1.fill_betweenx([-80, 5], f_stop_low*(f_sample/2), f_stop_high*(f_sample/2),
                    alpha=0.2, color='red', label='阻带范围')
ax5_1.legend(loc='upper right')

ax5_2.plot(w, np.unwrap(phase), 'r', linewidth=2)
ax5_2.set_xlabel('频率 (Hz)')
ax5_2.set_ylabel('相位 (弧度)')
ax5_2.set_title('数字滤波器 H(z) 相频特性')
ax5_2.grid(True, alpha=0.3)
ax5_2.set_xlim([0, 200])

plt.tight_layout()
plt.show()


print(f"\n 滤波器系数：")
print(f"分子系数 b (前馈系数): {b}")
print(f"分母系数 a (反馈系数): {a}")


y = signal.lfilter(b, a, x)

Y_f = fft(y)
y_spectrum = np.abs(Y_f[:N_d//2])

power_s1 = np.mean(s1**2)
power_s2 = np.mean(s2**2)
power_x = np.mean(x**2)
power_y = np.mean(y**2)

snr_before = 10 * np.log10(power_s1 / power_s2)
snr_after = 10 * np.log10(power_s1 / (power_y - power_s1))

print(f"\n 信号功率分析：")
print(f"有用信号功率：{power_s1:.6f}")
print(f"干扰信号功率：{power_s2:.6f}")
print(f"输入信号功率：{power_x:.6f}")
print(f"输出信号功率：{power_y:.6f}")
print(f"滤波前信噪比：{snr_before:.2f} dB")
print(f"滤波后信噪比：{snr_after:.2f} dB")
print(f"信噪比改善：{snr_after - snr_before:.2f} dB")

fig6, axes6 = plt.subplots(2, 2, figsize=(14, 10))

axes6[0, 0].stem(n[:50], y[:50], 'b', linefmt='b-', markerfmt='bo', basefmt='k-')
axes6[0, 0].set_xlabel('时间 (s)')
axes6[0, 0].set_ylabel('幅度')
axes6[0, 0].set_title('滤波器输出 y(n) 时域波形')
axes6[0, 0].grid(True, alpha=0.3)

axes6[0, 1].plot(freq_plot_d, y_spectrum, 'b', linewidth=1.5)
axes6[0, 1].set_xlabel('频率 (Hz)')
axes6[0, 1].set_ylabel('幅度')
axes6[0, 1].set_title('滤波器输出 y(n) 频谱')
```

```
axes6[0, 1].grid(True, alpha=0.3)
axes6[0, 1].set_xlim([0, 200])

axes6[1, 0].plot(freq_plot_d, x_spectrum_d, 'r', linewidth=1.5, alpha=0.7, label='滤波前 x(n)')
axes6[1, 0].plot(freq_plot_d, y_spectrum, 'b', linewidth=1.5, label='滤波后 y(n)')
axes6[1, 0].set_xlabel('频率 (Hz)')
axes6[1, 0].set_ylabel('幅度')
axes6[1, 0].set_title('滤波前后频谱对比')
axes6[1, 0].grid(True, alpha=0.3)
axes6[1, 0].set_xlim([0, 200])
axes6[1, 0].legend()

axes6[1, 1].plot(freq_plot_d, s1_spectrum_d, 'g', linewidth=2, label='原始有用信号 s1(n)')
axes6[1, 1].plot(freq_plot_d, y_spectrum, 'b', linewidth=1.5, alpha=0.7, label='滤波器输出 y(n)')
axes6[1, 1].set_xlabel('频率 (Hz)')
axes6[1, 1].set_ylabel('幅度')
axes6[1, 1].set_title('有用信号与滤波器输出对比')
axes6[1, 1].grid(True, alpha=0.3)
axes6[1, 1].set_xlim([0, 100])
axes6[1, 1].legend()

plt.tight_layout()

fig7, axes7 = plt.subplots(3, 1, figsize=(12, 10))

axes7[0].plot(n[:100], s1[:100], 'g', linewidth=2, label='原始有用信号 s1(n)')
axes7[0].set_xlabel('时间 (s)')
axes7[0].set_ylabel('幅度')
axes7[0].set_title('原始有用信号')
axes7[0].grid(True, alpha=0.3)
axes7[0].legend()

axes7[1].plot(n[:100], x[:100], 'r', linewidth=1.5, alpha=0.7, label='滤波器输入 x(n)')
axes7[1].set_xlabel('时间 (s)')
axes7[1].set_ylabel('幅度')
axes7[1].set_title('滤波器输入信号（含干扰）')
axes7[1].grid(True, alpha=0.3)
axes7[1].legend()

axes7[2].plot(n[:100], y[:100], 'b', linewidth=1.5, label='滤波器输出 y(n)')
axes7[2].plot(n[:100], s1[:100], 'g--', linewidth=1, alpha=0.5, label='原始有用信号 (参考)')
axes7[2].set_xlabel('时间 (s)')
axes7[2].set_ylabel('幅度')
axes7[2].set_title('滤波器输出信号')
axes7[2].grid(True, alpha=0.3)
axes7[2].legend()

plt.tight_layout()
plt.show()
```

六、实验问题及解决方案

	实验问题	解决方案
XXX	**任务（1）： 模拟信号建模与仿真**	通过数学公式构建指定频率、幅度、带宽的带限信号，叠加生成合成信号，并通过 FFT 分析频谱，绘图验证信号特性。
	任务（2）： 奈奎斯特采样与离散信号分析	根据奈奎斯特定理确定采样频率，对模拟信号采样得到离散序列，再通过 FFT 分析离散信号频谱，验证采样无混叠。
XXX	**任务（3）： 数字滤波器设计**	根据干扰信号的频率范围设计椭圆带阻滤波器，验证阻带衰减满足 ＞40dB 的要求，绘制幅频 / 相频特性曲线。
	任务（4）： 滤波器结构实现	选择直接 II 型结构，说明其优势，并通过滤波器系数（b/a）体现结构特征。
XXX	**任务（5）： 滤波器应用与输出分析**	用设计好的滤波器对含干扰的离散信号滤波，分析滤波前后的功率、信噪比，绘图对比时域 / 频域结果，验证滤波效果。
	任务（6）： 分析与总结	通过文字总结，覆盖所有任务的完成情况，提炼核心结论。