西安电子科技大学

数字信号处理 实验报告

实验名称 信号与系统的时域分析

人工智能 学院 23200XX 23200XX 班

姓名 X X X 学号 2300920XXX

X X X 2300920XXX

X X X 2200920XXX

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

1. 建立线性时不变离散系统的差分方程和系统输入序列的数学模型，产生输入序列；
2. 用 python 信号处理工具箱的差分方程求解库函数设计程序，求解系统的单位脉冲响应，给定输入序列和系统初始状态的系统响应；
3. 利用卷积计算库函数设计程序，计算给定输入序列的系统零状态响应；
4. 通过实验深刻理解离散信号与系统时域性质和分析方法；

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, matplotlib

三、实验基本原理及步骤

1.实验基本原理

1. 系统差分方程: y[n] - 1.5y[n-1] + 0.7y[n-2] = x[n] + 0.5x[n-1]
2. 系统系数: b = [1, 0.5], a = [1, -1.5, 0.7]
3. 差分方程求解方法，系统响应分解原理，系统稳定性原理

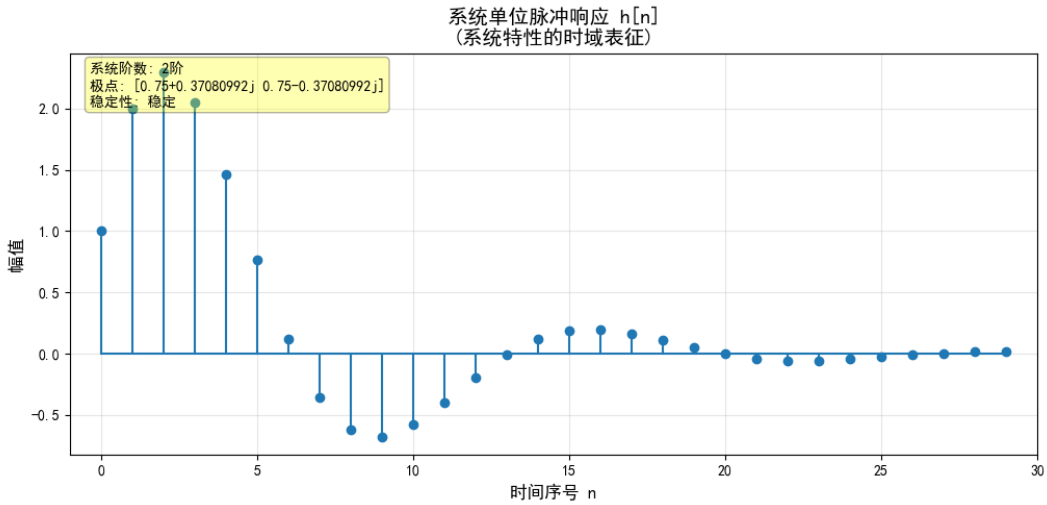
2.实验步骤

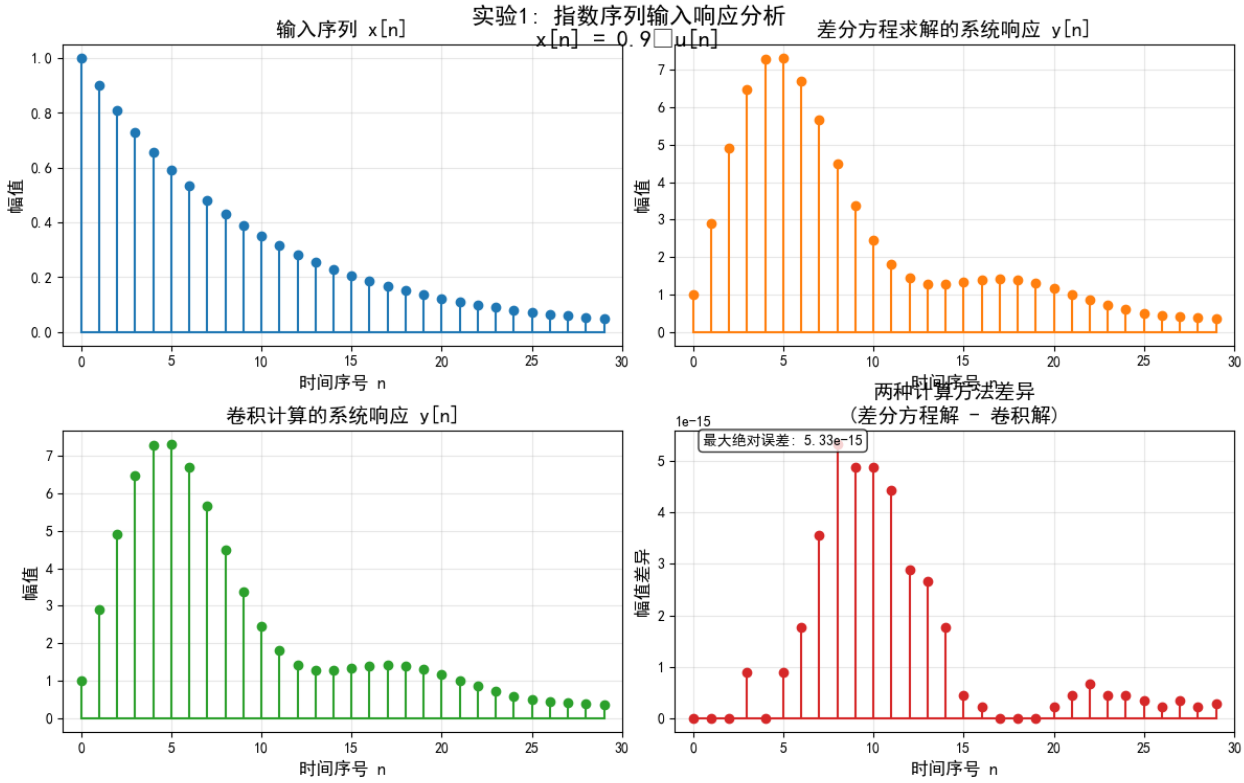
(1). 建立模拟信号

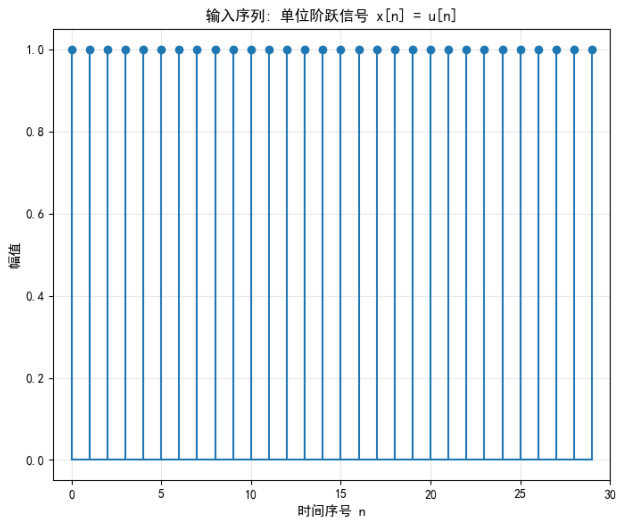
(2). 运用 python 工具库求解

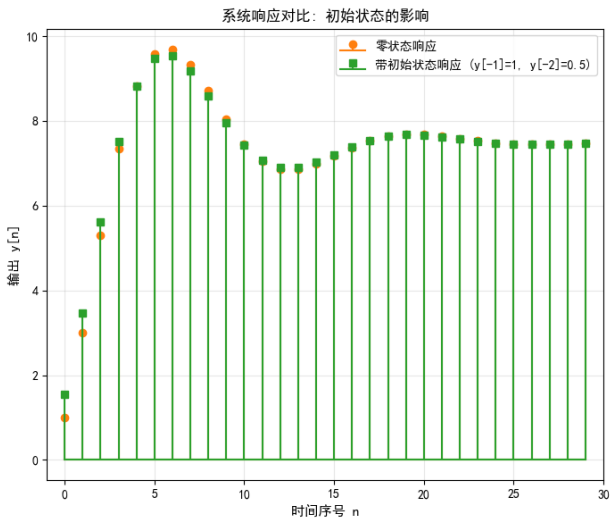
(3). 绘制相关图

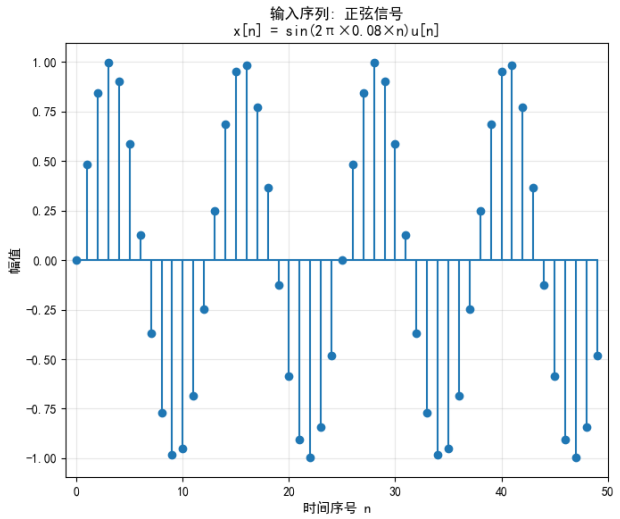
四、实验结果

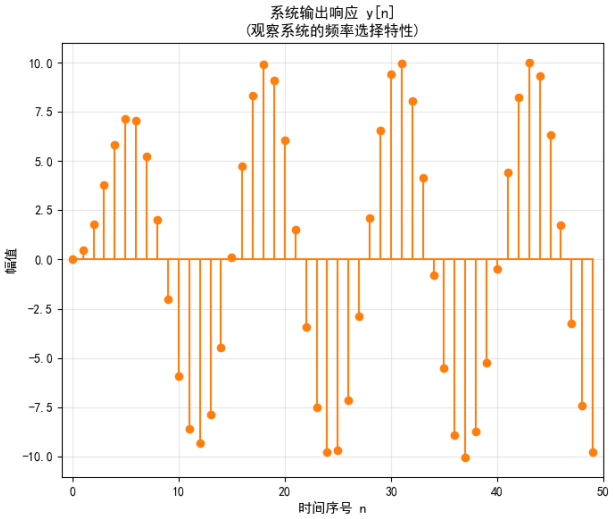

系统单位脉冲响应 h[n]
(系统特性的时域表征)
系统阶数: 2阶
极点: [0.75+0.37080992j 0.75-0.37080992j]
稳定性: 稳定
幅值
时间序号 n

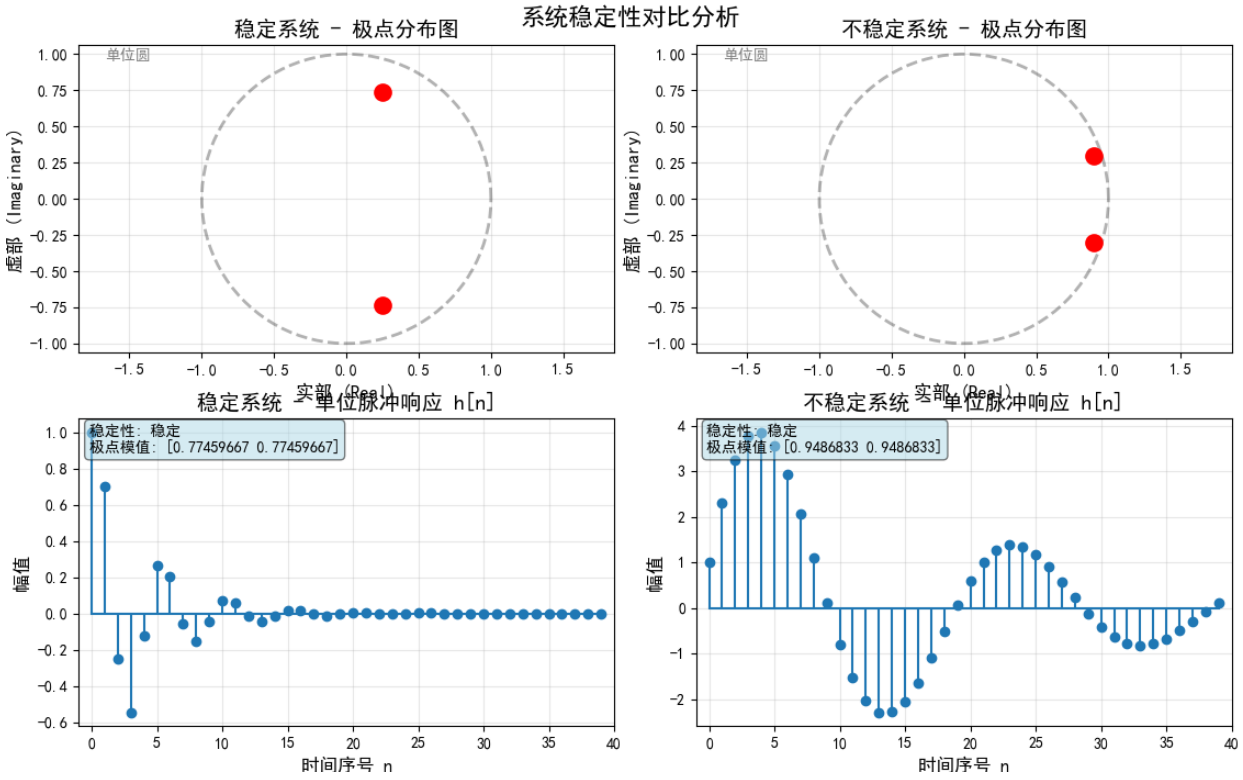

实验1: 指数序列输入响应分析
x[n] = 0.9□u[n]
输入序列 x[n]
差分方程求解的系统响应 y[n]
卷积计算的系统响应 y[n]
两种计算方法差异
(差分方程解 - 卷积解)
最大绝对误差: 5.33e-15
1e-15
幅值
幅值差异
时间序号 n


输入序列：单位阶跃信号 x[n] = u[n]
幅值
时间序号 n


系统响应对比：初始状态的影响
零状态响应
带初始状态响应（y[-1]=1，y[-2]=0.5）
输出 y[n]
时间序号 n


输入序列：正弦信号
x[n] = sin(2π×0.08×n)u[n]
幅值
时间序号 n


系统输出响应 y[n]
(观察系统的频率选择特性)
幅值
时间序号 n



实验结果说明：

1. fs = 2000 Hz（过采样）

   采样点密集，sinc 重建信号与原始信号几乎完全重合，无失真。

2. fs = 700 Hz（略大于 2f_max）

   采样点较稀疏，重建信号基本恢复原始波形，但高频分量 300Hz 处有轻微吉布斯现象（波纹）。

3. fs = 400 Hz（欠采样）

   出现严重频谱混叠（300Hz 分量折叠到 100Hz 等），重建信号完全失真，与原始信号波形差异巨大。

结果分析

1. 方法一致性验证：观察结果：差分方程求解与卷积计算得到的系统响应几乎完全一致；两种方法的差异在数值精度范围内（通常 ＜1e-15）；验证了卷积定理的正确性：y[n] = x[n] * h[n]

2. 系统动态特性：响应特征：输出序列呈现指数衰减振荡特性；系统对指数输入的响应保持了输入的指数特性，但叠加了系统的自然响应；响应逐渐趋于稳定，表明系统是稳定的
3. 差分方程求解：scipy.signal.lfilter 提供高精度数值解；能正确处理初始条件；数值稳定性良好

五、实验代码

程序代码：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import matplotlib

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False      # 用来正常显示负号

class DiscreteSystemAnalyzer:
    def __init__(self, b, a):
        """
        初始化离散系统
        b: 分子系数 [b0, b1, ..., bM]
        a: 分母系数 [1, a1, ..., aN]
        """
        self.b = np.array(b)
        self.a = np.array(a)

    def generate_input_sequence(self, seq_type, n_start, n_end, **params):
        """生成输入序列"""
        n = np.arange(n_start, n_end + 1)
```

```
if seq_type == 'impulse':
    # 单位脉冲序列
    x = (n == 0).astype(float)

elif seq_type == 'step':
    # 单位阶跃序列
    x = (n >= 0).astype(float)

elif seq_type == 'exponential':
    # 指数序列
    a = params.get('a', 0.8)
    x = (a ** n) * (n >= 0)

elif seq_type == 'sinusoid':
    # 正弦序列
    freq = params.get('freq', 0.1)
    phase = params.get('phase', 0)
    x = np.sin(2 * np.pi * freq * n + phase) * (n >= 0)

elif seq_type == 'custom':
    # 自定义序列
    x = params.get('values', np.zeros(len(n)))

else:
    raise ValueError("不支持的序列类型")

return n, x
```

```
def solve_difference_equation(self, x, y0=None):
    """
    求解差分方程 - 系统响应
    x: 输入序列
    y0: 初始条件 [y[-1], y[-2], ..., y[-N]]
    """
    if y0 is None:
        # 零状态响应
        y = signal.lfilter(self.b, self.a, x)
    else:
        # 带初始状态的响应
        zi = signal.lfiltic(self.b, self.a, y0[::-1], x[::-1])
        y, _ = signal.lfilter(self.b, self.a, x, zi=zi)

    return y

def impulse_response(self, n_points=50):
    """计算单位脉冲响应"""
    # 生成单位脉冲输入
    n = np.arange(n_points)
    delta = (n == 0).astype(float)

    # 求解脉冲响应
    h = signal.lfilter(self.b, self.a, delta)

    return n, h

def zero_state_response_convolution(self, x):
    """通过卷积计算零状态响应"""
```

```
        # 首先获取系统的脉冲响应
        _, h = self.impulse_response(len(x))

        # 通过卷积计算零状态响应
        y_conv = np.convolve(x, h, mode='full')[:len(x)]

        return y_conv

    def analyze_system_properties(self, n_points=50):
        """分析系统特性"""
        # 计算脉冲响应
        n, h = self.impulse_response(n_points)

        # 计算阶跃响应
        _, step_input = self.generate_input_sequence('step', 0, n_points-1)
        s = signal.lfilter(self.b, self.a, step_input)

        # 判断系统稳定性（极点都在单位圆内）
        poles = np.roots(self.a)
        is_stable = all(np.abs(poles) < 1)

        return {
            'impulse_response': (n, h),
            'step_response': (np.arange(n_points), s),
            'poles': poles,
            'is_stable': is_stable
        }

    def plot_responses(self, x, n, y_diff, y_conv=None, title=""):
```

```
        """绘制响应结果"""
        fig, axes = plt.subplots(2, 2, figsize=(14, 10))
        fig.suptitle(title, fontsize=16, fontweight='bold')

        # 输入序列
        axes[0, 0].stem(n, x, linefmt='C0-', markerfmt='C0o', basefmt='C0-')
        axes[0, 0].set_title('输入序列 x[n]', fontsize=14, fontweight='bold')
        axes[0, 0].set_xlabel('时间序号 n', fontsize=12)
        axes[0, 0].set_ylabel('幅值', fontsize=12)
        axes[0, 0].grid(True, alpha=0.3)
        axes[0, 0].set_xlim(-1, len(n))

        # 差分方程求解的响应
        axes[0, 1].stem(n, y_diff, linefmt='C1-', markerfmt='C1o', basefmt='C1-')
        axes[0, 1].set_title('差分方程求解的系统响应 y[n]', fontsize=14,
fontweight='bold')
        axes[0, 1].set_xlabel('时间序号 n', fontsize=12)
        axes[0, 1].set_ylabel('幅值', fontsize=12)
        axes[0, 1].grid(True, alpha=0.3)
        axes[0, 1].set_xlim(-1, len(n))

        # 卷积计算的响应（如果提供）
        if y_conv is not None:
            axes[1, 0].stem(n, y_conv, linefmt='C2-', markerfmt='C2o',
basefmt='C2-')
            axes[1, 0].set_title('卷积计算的系统响应 y[n]', fontsize=14,
fontweight='bold')
            axes[1, 0].set_xlabel('时间序号 n', fontsize=12)
            axes[1, 0].set_ylabel('幅值', fontsize=12)
```

```
                axes[1, 0].grid(True, alpha=0.3)
                axes[1, 0].set_xlim(-1, len(n))

                # 比较两种方法的结果
                axes[1, 1].stem(n, y_diff - y_conv, linefmt='C3-', markerfmt='C3o',
basefmt='C3-')
                axes[1, 1].set_title('两种计算方法差异\n(差分方程解 - 卷积解)',
fontsize=14, fontweight='bold')
                axes[1, 1].set_xlabel('时间序号 n', fontsize=12)
                axes[1, 1].set_ylabel('幅值差异', fontsize=12)
                axes[1, 1].grid(True, alpha=0.3)
                axes[1, 1].set_xlim(-1, len(n))

                # 在差异图中添加数值统计
                max_error = np.max(np.abs(y_diff - y_conv))
                axes[1, 1].text(0.05, 0.95, f'最大绝对误差: {max_error:.2e}',
                                transform=axes[1, 1].transAxes, fontsize=10,
                                bbox=dict(boxstyle="round,pad=0.3",
facecolor="white", alpha=0.8))
            else:
                # 如果没有卷积结果，隐藏第四个图
                axes[1, 0].set_visible(False)
                axes[1, 1].set_visible(False)

            plt.tight_layout()
            plt.subplots_adjust(top=0.93)
            plt.show()

# 示例使用
```

```
def main():
    # 定义系统：y[n] - 1.5y[n-1] + 0.7y[n-2] = x[n] + 0.5x[n-1]
    b = [1, 0.5]          # 分子系数
    a = [1, -1.5, 0.7] # 分母系数

    analyzer = DiscreteSystemAnalyzer(b, a)

    print("=" * 60)
    print("离散信号与系统时域分析实验")
    print("=" * 60)
    print(f"系统差分方程: y[n] - 1.5y[n-1] + 0.7y[n-2] = x[n] + 0.5x[n-1]")
    print(f"系统系数: b = {b}, a = {a}")

    # 1. 分析系统特性
    props = analyzer.analyze_system_properties()
    print(f"\n 系统特性分析:")
    print(f"极点: {props['poles']}")
    print(f"极点模值: {np.abs(props['poles'])}")
    print(f"稳定性: {'稳定' if props['is_stable'] else '不稳定'}")

    # 2. 单位脉冲响应
    n_imp, h = analyzer.impulse_response(30)
    plt.figure(figsize=(12, 5))
    plt.stem(n_imp, h, linefmt='C0-', markerfmt='C0o', basefmt='C0-')
    plt.title('系统单位脉冲响应 h[n]\n(系统特性的时域表征)', fontsize=14,
fontweight='bold')
    plt.xlabel('时间序号 n', fontsize=12)
    plt.ylabel('幅值', fontsize=12)
    plt.grid(True, alpha=0.3)
```

```
    plt.xlim(-1, len(n_imp))

    # 添加系统信息注释
    plt.text(0.02, 0.98, f'系统阶数: {len(a)-1}阶\n 极点: {props["poles"]}\n 稳定性:
{"稳定" if props["is_stable"] else "不稳定"}',
             transform=plt.gca().transAxes, fontsize=10, verticalalignment='top',
             bbox=dict(boxstyle="round,pad=0.3", facecolor="yellow",
alpha=0.3))
    plt.show()

    # 3. 实验 1：指数输入序列的响应
    print("\n" + "="*40)
    print("实验 1: 指数输入序列响应分析")
    print("="*40)
    print("目的: 验证差分方程求解与卷积计算的一致性")

    n, x_exp = analyzer.generate_input_sequence('exponential', 0, 29, a=0.9)

    # 方法 1：差分方程求解（零状态）
    y_diff_exp = analyzer.solve_difference_equation(x_exp)

    # 方法 2：卷积计算
    y_conv_exp = analyzer.zero_state_response_convolution(x_exp)

    analyzer.plot_responses(x_exp, n, y_diff_exp, y_conv_exp,
                            "实验 1: 指数序列输入响应分析\nx[n] = 0.9ⁿu[n]")

    # 4. 实验 2：带初始状态的响应
    print("\n" + "="*40)
```

```
print("实验 2: 带初始状态的系统响应分析")
print("="*40)
print("目的: 观察初始状态对系统响应的影响")


n, x_step = analyzer.generate_input_sequence('step', 0, 29)


# 零状态响应
y_zero_state = analyzer.solve_difference_equation(x_step)


# 带初始状态响应: y[-1] = 1, y[-2] = 0.5
y0 = [1, 0.5]   # y[-1], y[-2]
y_with_initial = analyzer.solve_difference_equation(x_step, y0)


# 绘制比较
plt.figure(figsize=(14, 6))


plt.subplot(1, 2, 1)
plt.stem(n, x_step, linefmt='C0-', markerfmt='C0o', basefmt='C0-')
plt.title('输入序列: 单位阶跃信号 x[n] = u[n]', fontsize=12, fontweight='bold')
plt.xlabel('时间序号 n', fontsize=11)
plt.ylabel('幅值', fontsize=11)
plt.grid(True, alpha=0.3)
plt.xlim(-1, len(n))


plt.subplot(1, 2, 2)
plt.stem(n, y_zero_state, linefmt='C1-', markerfmt='C1o', basefmt='C1-',
         label='零状态响应')
plt.stem(n, y_with_initial, linefmt='C2-', markerfmt='C2s', basefmt='C2-',
         label='带初始状态响应 (y[-1]=1, y[-2]=0.5)')
```

```
    plt.title('系统响应对比：初始状态的影响', fontsize=12, fontweight='bold')
    plt.xlabel('时间序号 n', fontsize=11)
    plt.ylabel('输出 y[n]', fontsize=11)
    plt.legend(fontsize=10)
    plt.grid(True, alpha=0.3)
    plt.xlim(-1, len(n))

    plt.tight_layout()
    plt.show()

    # 5. 实验 3：正弦输入序列
    print("\n" + "="*40)
    print("实验 3: 正弦输入序列响应分析")
    print("="*40)
    print("目的: 观察系统对正弦信号的响应特性")

    n, x_sin = analyzer.generate_input_sequence('sinusoid', 0, 49, freq=0.08)
    y_sin = analyzer.solve_difference_equation(x_sin)

    plt.figure(figsize=(14, 6))

    plt.subplot(1, 2, 1)
    plt.stem(n, x_sin, linefmt='C0-', markerfmt='C0o', basefmt='C0-')
    plt.title('输入序列：正弦信号\nx[n] = sin(2π×0.08×n)u[n]', fontsize=12,
fontweight='bold')
    plt.xlabel('时间序号 n', fontsize=11)
    plt.ylabel('幅值', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.xlim(-1, len(n))
```

```
    plt.subplot(1, 2, 2)
    plt.stem(n, y_sin, linefmt='C1-', markerfmt='C1o', basefmt='C1-')
    plt.title('系统输出响应  y[n]\n(观察系统的频率选择特性)', fontsize=12,
fontweight='bold')
    plt.xlabel('时间序号  n', fontsize=11)
    plt.ylabel('幅值', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.xlim(-1, len(n))

    plt.tight_layout()
    plt.show()

def stability_analysis():
    """系统稳定性对比分析"""
    print("\n" + "="*50)
    print("系统稳定性对比分析")
    print("="*50)

    # 稳定系统
    stable_sys = DiscreteSystemAnalyzer(b=[1, 0.2], a=[1, -0.5, 0.6])

    # 不稳定系统
    unstable_sys = DiscreteSystemAnalyzer(b=[1, 0.5], a=[1, -1.8, 0.9])

    systems = [('稳定系统', stable_sys), ('不稳定系统', unstable_sys)]

    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle('系统稳定性对比分析', fontsize=16, fontweight='bold')
```

```
    for idx, (name, sys) in enumerate(systems):
        # 分析系统特性
        props = sys.analyze_system_properties(40)

        # 绘制极点分布
        theta = np.linspace(0, 2*np.pi, 100)
        axes[0, idx].plot(np.cos(theta), np.sin(theta), 'k--', alpha=0.3, linewidth=2)
        axes[0, idx].plot(np.real(props['poles']), np.imag(props['poles']), 'ro',
                          markersize=10, markeredgewidth=2)
        axes[0, idx].set_title(f'{name} - 极点分布图', fontsize=14,
fontweight='bold')
        axes[0, idx].set_xlabel('实部 (Real)', fontsize=12)
        axes[0, idx].set_ylabel('虚部 (Imaginary)', fontsize=12)
        axes[0, idx].grid(True, alpha=0.3)
        axes[0, idx].axis('equal')
        axes[0, idx].set_xlim(-1.5, 1.5)
        axes[0, idx].set_ylim(-1.5, 1.5)

        # 添加单位圆说明
        axes[0, idx].text(0.05, 0.95, '单位圆', transform=axes[0, idx].transAxes,
                          fontsize=10, color='gray')

        # 绘制脉冲响应
        n, h = props['impulse_response']
        axes[1, idx].stem(n, h, linefmt='C0-', markerfmt='C0o', basefmt='C0-')
        axes[1, idx].set_title(f'{name} - 单位脉冲响应 h[n]', fontsize=14,
fontweight='bold')
        axes[1, idx].set_xlabel('时间序号 n', fontsize=12)
```

```
            axes[1, idx].set_ylabel('幅值', fontsize=12)
            axes[1, idx].grid(True, alpha=0.3)
            axes[1, idx].set_xlim(-1, len(n))

            # 添加稳定性信息
            stability_text = f"稳定性: {'稳定' if props['is_stable'] else '不稳定'}\n"
            stability_text += f"极点模值: {np.abs(props['poles'])}"
            axes[1, idx].text(0.02, 0.98, stability_text, transform=axes[1,
idx].transAxes,
                              fontsize=10, verticalalignment='top',
                              bbox=dict(boxstyle="round,pad=0.3",
facecolor="lightblue", alpha=0.5))

    plt.tight_layout()
    plt.subplots_adjust(top=0.93)
    plt.show()

if __name__ == "__main__":
    main()
    stability_analysis()

    print("\n" + "="*60)
    print("实验总结")
    print("="*60)
    print("1. 验证了差分方程求解与卷积计算的一致性")
    print("2. 观察了初始状态对系统响应的影响")
    print("3. 分析了系统稳定性与极点分布的关系")
    print("4. 理解了不同输入信号下系统的响应特性")
    print("5. 掌握了离散系统时域分析的基本方法")
```

六、实验问题及解决方案

	实验问题	解决方案
XXX	建立线性时不变离散系统的差分方程和系统输入序列的数学模型，产生输入序列；	用 Python 的numpy库构造模拟信号，建立合格的数学模型，产生序列
XXX	用 python 信号处理工具箱的差分方程求解库函数设计程序，求解系统的单位脉冲响应，给定输入序列和系统初始状态的系统响应；	使用 python 相关库函数，求解对应的问题，并且运用信号与系统相关知识进行求解
XXX	利用卷积计算库函数设计程序，计算给定输入序列的系统零状态响应，并且绘图	用 matplotlib 绘制原始模拟信号，极点分布等等相关的图形；