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

《数字信号处理课程设计》报告

运用 MATLAB 的卷积演示系统

学　　院: 人工智能学院

专　　业: 人工智能

学　生 1: 杨跃浙 WA2214014

学　生 2: 杨昀川 WA2214022

指导老师: 郑曼

课程编号: SJ52021

课程学分: 1

提交日期: 24.10.25

目　录

1 实验目的和要求

1.1 实验目的

通过本次课程设计，使得学生能够充分了解卷积的相关定义、计算和定理；使得学生能够使用 GUI/app design 进行图形化界面设计。本实验的主要目的是使学生能够深入理解卷积的基本定义、计算方法以及相关定理，从而掌握其在信号处理中的应用。学生将通过详细学习卷积的数学原理和性质，深化对如何将两个独立信号融合成一个新信号的理解，进一步认识到卷积在数据分析和工程实践中的应用价值。

1.2 实验要求

1.2.1 基础设计要求

1. 用 MATLAB 完成线性卷积的计算过程，并绘图。
2. 使用 GUI/app design 设计一个线性卷积的基本演示系统，要求：
   - 两个卷积信号的参数可以自由给定。
   - 设计框图中直接包含“计算按钮”，通过点击该按钮直接计算卷积结果。
3. 分析卷积计算结果。

1.2.2 提高设计要求

1. 设计线性卷积计算过程的演示系统界面；
2. 完成圆周卷积的计算过程及演示系统，分为时域计算过程和频域计算过程；
3. 验证线性卷积与圆周卷积的关系，并设计演示系统界面；
4. 使用 matlab 编程验证其他关于卷积的计算原理或定理。

2 实验原理

2.1 线性卷积

设两个序列 $x(n)$ 和 $h(n)$，其线性卷积[1] 定义为：

$$y(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m) \tag{1}$$

线性卷积计算过程[2] 可以分为以下几步：

1. **翻褶**: 将 $h(m)$ 进行时间翻转，得到 $h(-m)$。
2. **移位**: 将翻转后的 $h(-m)$ 进行平移，得到 $h(n-m)$。
3. **相乘**: 对每个 m，计算 $x(m) \cdot h(n-m)$。
4. **相加**: 对所有的 m 求和，得到卷积结果 $y(n)$。

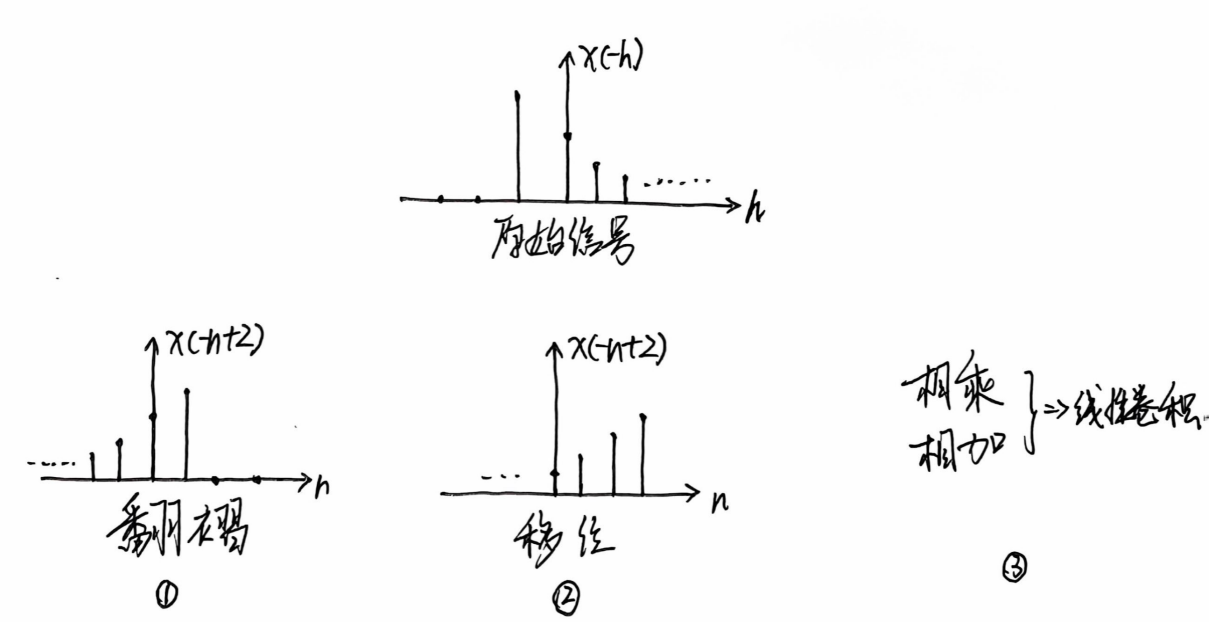


图 1: 线性卷积原理图

2.2 圆周卷积

2.2.1 圆周卷积的时域过程

设 $x(n)$ 和 $h(n)$ 为 N 点有限长序列，其圆周卷积[3] 定义为:

$$y(n) = \left[\sum_{m=0}^{N-1} x_1(m)x_2((n-m))_N\right] R_N(n) \tag{2}$$

圆周卷积计算过程包括以下步骤:

1. **补零**: 将序列 $x_1(n)$ 和 $x_2(n)$ 补零至 N 点长度。
2. **周期延拓**: $x_2(m)$ 进行周期延拓，得到 $x_2((m))_N$。
3. **圆周翻褶**: 将 $x_2((m))_N$ 进行翻转，得到 $x_2((-m))_N$。
4. **圆周移位**: 将 $x_2((-m))_N$ 按照 n 进行圆周平移，得到 $x_2((n-m))_N R_N(m)$。
5. **相乘相加**: 计算 $x_1(m)x_2((n-m))_N R_N(m)$ 并对所有 m 求和。

2.2.2 圆周卷积的频域过程

圆周卷积在频域中也有相应的计算方法。通过离散傅里叶变换（DFT）将时域卷积转换为频域乘积:

$$DFT[x_1(n)] = X_1(k), \quad DFT[x_2(n)] = X_2(k) \tag{3}$$

卷积的频域乘积为:

$$Y(k) = X_1(k) \cdot X_2(k) \tag{4}$$

通过逆傅里叶变换得到时域卷积结果：

$$y(n) = IDFT[Y(k)] = \left[\sum_{m=0}^{N-1} x_1(m)x_2((n-m))_N R_N(n)\right] \tag{5}$$

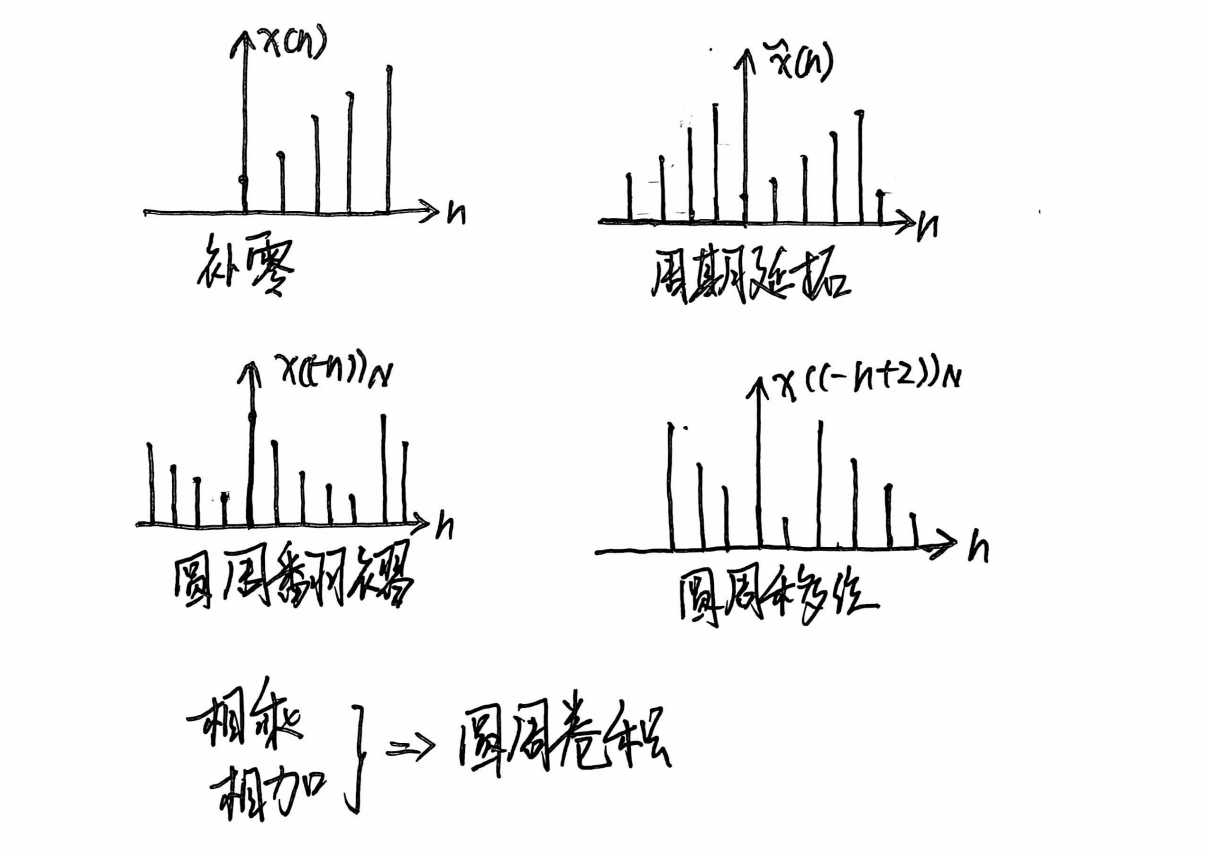


图 2: 圆周卷积原理图

2.3 圆周卷积和线性卷积的关系

圆周卷积与线性卷积的关系可以表示为：

- 当圆周卷积长度 $L \geq N + M - 1$ 时，没有混叠，圆周卷积的前 $N + M - 1$ 个值为有效值，且与线性卷积相同，其余为 0。
- 当圆周卷积长度 $L < N + M - 1$ 时，产生混叠，圆周卷积的前 $(N + M - 1) - L$ 个值有混叠。

2.4 Matlab 实现

2.4.1 线性卷积

在 MATLAB [4] 中，可以使用 `conv` 函数来实现线性卷积[5]。下面的代码应居中显示：

```
y = conv(x, h)
```

2.4.2 圆周卷积

在 MATLAB 中，可以使用 cconv 函数来实现圆周卷积，代码如下：

```
y = cconv(x, h, N)
```

或通过快速傅里叶变换（FFT）来实现高效计算[6]，代码也应居中显示：

```
y = ifft(fft(x, N) .* fft(h, N))
```

3 实验方法与内容

3.1 需求分析

在本实验中，学生将通过 MATLAB 和图形用户界面设计工具深入学习和应用卷积运算。实验的主要需求包括：

- **理解卷积运算：**学生需要理解卷积的数学基础，包括线性卷积和圆周卷积的理论和计算方法。
- **实现卷积计算：**通过 MATLAB 编程实现线性卷积和圆周卷积，包括时域和频域方法。
- **设计用户界面：**创建一个交互式的 GUI 应用程序，允许用户输入自定义信号，执行卷积运算，并直观地展示结果。
- **分析和比较：**对比线性卷积和圆周卷积的结果，分析两者在不同条件下的行为和结果。
- **验证卷积原理：**使用具体例子验证卷积定理，如卷积的交换律、结合律等。

3.2 算法设计思路

本实验的算法设计将遵循以下步骤，确保可以全面地实现卷积运算和设计需求：

1. **线性卷积的算法实现：**
   - 使用 MATLAB 中的 conv 函数直接计算两个信号的线性卷积。
   - 开发函数验证卷积的性质，如交换律和结合律。
2. **圆周卷积的实现：**
   - 实现时域的圆周卷积计算，使用 MATLAB 的 cconv 函数。
   - 实现频域的圆周卷积计算，通过 FFT 和 IFFT 来演示卷积的频域等价性。
3. **GUI 设计：**
   - 设计一个简洁的用户界面，包括输入框、计算按钮和图形显示区域。
   - 界面应允许用户选择卷积类型（线性或圆周），并输入自定义的信号参数。
4. **结果分析：**
   - 在 GUI 中展示卷积结果的图形，包括原始信号和卷积后的信号。
   - 提供选项比较线性卷积和圆周卷积的结果，以及在不同参数下的表现。

3.3 流程图

图 3: 系统设计流程图

4 实验记录

4.1 利用 Matlab 实现线性卷积

```
% 定义时间轴
n1 = 0:10;
n2 = 0:5;

% 定义信号 x[n]
x = sin(0.2*pi*n1);

% 定义信号 h[n]
h = cos(0.4*pi*n2);
% 计算线性卷积 y[n]
y = conv(x, h);
% 定义卷积结果的时间轴
ny = (n1(1)+n2(1)):(n1(end)+n2(end));

% 绘制信号 x[n]
subplot(3,1,1);
stem(n1, x, 'filled');
title('信号 x[n]');
xlabel('n');
ylabel('x[n]');

% 绘制信号 h[n]
subplot(3,1,2);
stem(n2, h, 'filled');
title('信号 h[n]');
xlabel('n');
ylabel('h[n]');

% 绘制卷积结果 y[n]
```

```
subplot(3,1,3);
stem(ny, y, 'filled');
title('线性卷积 y[n] = x[n] * h[n]');
xlabel('n');
ylabel('y[n]');
```

4.2 App Design 实现输入任意序列的卷积运算

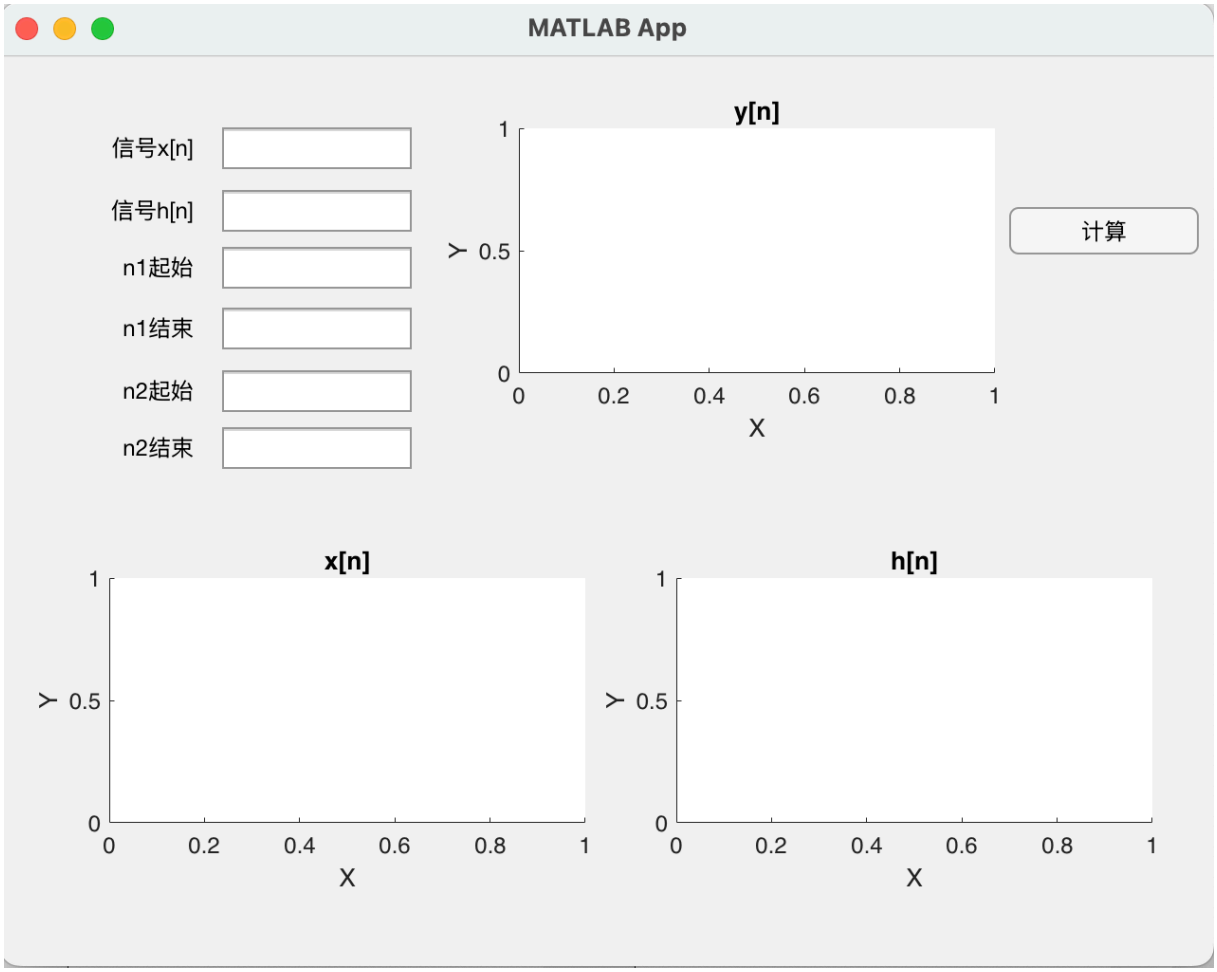


图 4: 卷积计算设计布局图

该处受篇幅影响，仅有核心交互代码

```
% Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: CalculateButton
        function CalculateButtonPushed(app, event)

            % 获取用户输入的表达式
            x_expr = app.xExpressionEditField.Value;
```

```
            h_expr = app.hExpressionEditField.Value;

            % 获取时间轴范围
            n1_start = str2double(app.n1StartEditField.Value);
            n1_end = str2double(app.n1EndEditField.Value);
            n2_start = str2double(app.n2StartEditField.Value);
            n2_end = str2double(app.n2EndEditField.Value);

            n1 = n1_start:n1_end;
            n2 = n2_start:n2_end;

            % 计算信号 x[n]
            n = n1; % 为了在表达式中使用
            x = eval(x_expr);

            % 计算信号 h[n]
            n = n2;
            h = eval(h_expr);

            % 计算线性卷积 y[n]
            y = conv(x, h);
            ny = (n1_start + n2_start):(n1_end + n2_end);

            % 绘制信号 x[n]
            stem(app.UIAxes1, n1, x, 'filled');
            title(app.UIAxes1, '信号 x[n]');
            xlabel(app.UIAxes1, 'n');
            ylabel(app.UIAxes1, 'x[n]');

            % 绘制信号 h[n]
            stem(app.UIAxes2, n2, h, 'filled');
            title(app.UIAxes2, '信号 h[n]');
            xlabel(app.UIAxes2, 'n');
            ylabel(app.UIAxes2, 'h[n]');

            % 绘制卷积结果 y[n]
            stem(app.UIAxes3, ny, y, 'filled');
            title(app.UIAxes3, '线性卷积 y[n] = x[n] * h[n]');
            xlabel(app.UIAxes3, 'n');
            ylabel(app.UIAxes3, 'y[n]');

        end
    end
```

4.3 计算结果分析

本节将分析卷积运算的结果，探讨卷积对信号特性的影响以及信号处理中卷积的应用意义。

- 卷积结果的长度: 卷积结果 $y[n]$ 的长度为 $L = L_x + L_h - 1$，其中 L_x 和 L_h 分别是信号 $x[n]$ 和 $h[n]$ 的长度。这表明卷积结果的时间轴比原始信号更长，体现了信号的延展性。
- 信号叠加效果: 卷积运算本质上是对信号的叠加和累积。当两个信号在某些区域有重叠时，卷积结果在这些区域的幅值会出现增加，这反映了信号之间的相互作用。
- 滤波特性: 如果将 $h[n]$ 视为系统的脉冲响应，则卷积结果 $y[n]$ 表示输入信号 $x[n]$ 通过该系统后的输出。这一点在信号处理和滤波器设计中极为重要。
- 频率特性变化: 卷积在时域上对应于频域的乘积。通过分析卷积结果的频率成分，可以了解输入信号经过系统处理后的频率特性变化。
- 具体示例分析: 在一个具体的示例中，假设 $x[n]$ 是一个频率为 0.2π 的正弦信号，$h[n]$ 是一个频率为 0.4π 的余弦信号。卷积结果 $y[n]$ 将包含这两个频率分量的组合，可能出现频率的和与差，即频率混叠。

  从绘图结果可以观察到：

  - 信号 $x[n]$ 和 $h[n]$ 的波形：分别呈现周期性的正弦和余弦曲线。
  - 卷积结果 $y[n]$ 的波形：波形更加复杂，幅度和形状都发生了变化，反映了两个信号的卷积叠加效果。
- 结论: 通过线性卷积计算和结果分析，可以加深对信号处理的理解，特别是信号通过线性时不变系统时的行为。卷积是分析和设计系统响应的基本工具，对于滤波、系统辨识和信号分析都具有重要意义。

4.4 卷积计算过程可视化

```
    % Callbacks that handle component events
methods (Access = private)

    % Code that executes after component creation
    function updateConvolutionProcess(app)

    persistent x_signal h_signal n1_range n2_range n2_flip h_flip h_shift
        n1_start n1_end n2_start n2_end; % 持久化变量
    % 第一步：绘制 x[n] 和 h[n]
    if app.currentStep == 1
        % 获取时间范围
        n1_start = str2double(app.n1StartEditField.Value);
        n1_end = str2double(app.n1EndEditField.Value);
        n2_start = str2double(app.n2StartEditField.Value);
        n2_end = str2double(app.n2EndEditField.Value);

        % 定义时间轴
        n1_range = n1_start:n1_end;
        n2_range = n2_start:n2_end;
```

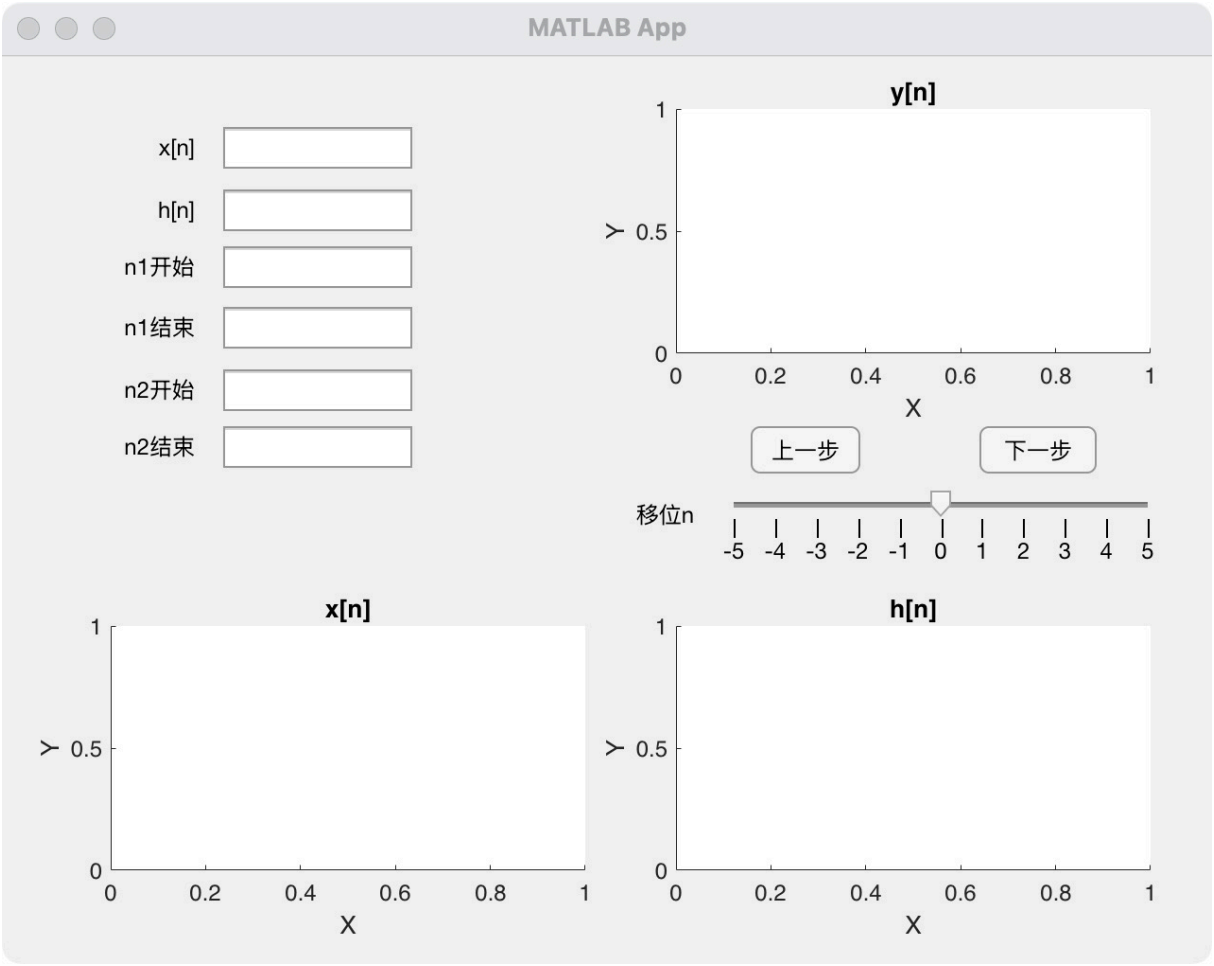


图 5: 卷积计算过程设计布局图

```
% 计算信号 x[n] 和 h[n]
n = n1_range;
x_signal = eval(app.xExpressionEditField.Value);
n = n2_range;
h_signal = eval(app.hExpressionEditField.Value);
% 绘制 x[n] 和 h[n]
cla(app.UIAxes_x);
stem(app.UIAxes_x, n1_range, x_signal, 'b', 'filled');
title(app.UIAxes_x, '信号 x[n]');
xlabel(app.UIAxes_x, 'n');
ylabel(app.UIAxes_x, 'x[n]');

cla(app.UIAxes_h);
stem(app.UIAxes_h, n2_range, h_signal, 'r', 'filled');
title(app.UIAxes_h, '信号 h[n]');
xlabel(app.UIAxes_h, 'n');
ylabel(app.UIAxes_h, 'h[n]');

% 清空卷积结果图表
```

```
        cla(app.UIAxes_y);
        title(app.UIAxes_y, '卷积过程');
        xlabel(app.UIAxes_y, 'n');
        ylabel(app.UIAxes_y, '结果');

    % 第二步: 翻折 h[n]
    elseif app.currentStep == 2
        h_flip = fliplr(h_signal); % 翻折 h[n]
        n2_flip = -fliplr(n2_range); % 翻折后的时间轴
        cla(app.UIAxes_y);
        stem(app.UIAxes_y, n2_flip, h_flip, 'r', 'filled');
        title(app.UIAxes_y, '翻折后的 h[-k]');
        xlabel(app.UIAxes_y, 'n');
        ylabel(app.UIAxes_y, '幅值');

    % 第三步: 移位 h[-k + n] (使用滑块控制移位量)
    elseif app.currentStep == 3
        shift = round(app.nSlider.Value); % 获取滑块的移位量
        h_shift = n2_flip + shift;
        cla(app.UIAxes_y);
        stem(app.UIAxes_y, h_shift, h_flip, 'r', 'filled');
        title(app.UIAxes_y, ['移位后的 h[-k + n], n = ', num2str(shift)]);
        xlabel(app.UIAxes_y, 'n');
        ylabel(app.UIAxes_y, '幅值');

    % 第四步: 相乘并显示相乘结果
    elseif app.currentStep == 4
        shift = round(app.nSlider.Value); % 获取滑块的移位量
        h_shift = n2_flip + shift;
        [n_overlap, idx_x, idx_h] = intersect(n1_range, h_shift); % 找到重叠区
            域
        x_overlap = x_signal(idx_x); % 获取 x 的重叠部分
        h_overlap = h_flip(idx_h);   % 获取 h 的重叠部分
        product = x_overlap .* h_overlap; % 相乘结果
        cla(app.UIAxes_y);
        stem(app.UIAxes_y, n_overlap, product, 'g', 'filled');
        title(app.UIAxes_y, ['相乘结果, 移位 n = ', num2str(shift)]);
        xlabel(app.UIAxes_y, 'n');
        ylabel(app.UIAxes_y, '幅值');

    % 第五步: 显示最终卷积结果
    elseif app.currentStep == 5
        y_conv = conv(x_signal, h_signal); % 计算卷积
        ny = (n1_start + n2_start):(n1_end + n2_end); % 卷积的时间范围
        cla(app.UIAxes_y);
        stem(app.UIAxes_y, ny, y_conv, 'm', 'filled'); % 显示卷积结果
        title(app.UIAxes_y, '卷积结果 y[n]');
        xlabel(app.UIAxes_y, 'n');
```

```
            ylabel(app.UIAxes_y, 'y[n]');
        end
        end

        % Value changed function: nSlider
        function StepSliderValueChanged2(app, event)
            app.updateConvolutionProcess();
        end

        % Value changed function: Button_pre
        function Button_preValueChanged(app, event)
            if app.currentStep > 0
                app.currentStep = app.currentStep - 1;
                app.updateConvolutionProcess();
            end
        end

        % Value changed function: Button_next
        function ButtonValueChanged(app, event)
            if app.currentStep < 5
                app.currentStep = app.currentStep + 1;
                app.updateConvolutionProcess();
            end
        end
    end
```

4.5 圆周卷积计算可视化

```
      % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function updateConvolutionProcess(app)

        persistent x_signal h_signal n1_range n2_range h_flip h_shift n1_start
            n1_end n2_start n2_end; % 持久化变量

        % 获取 N 的值（圆周长度），来自 NEditField
        N = str2double(app.NEditField.Value);

        % 初始状态，什么都不显示
        if app.currentStep == 0
            cla(app.UIAxes_x);
            cla(app.UIAxes_h);
            cla(app.UIAxes_y);
            title(app.UIAxes_x, '等待开始');
            title(app.UIAxes_h, '等待开始');
```

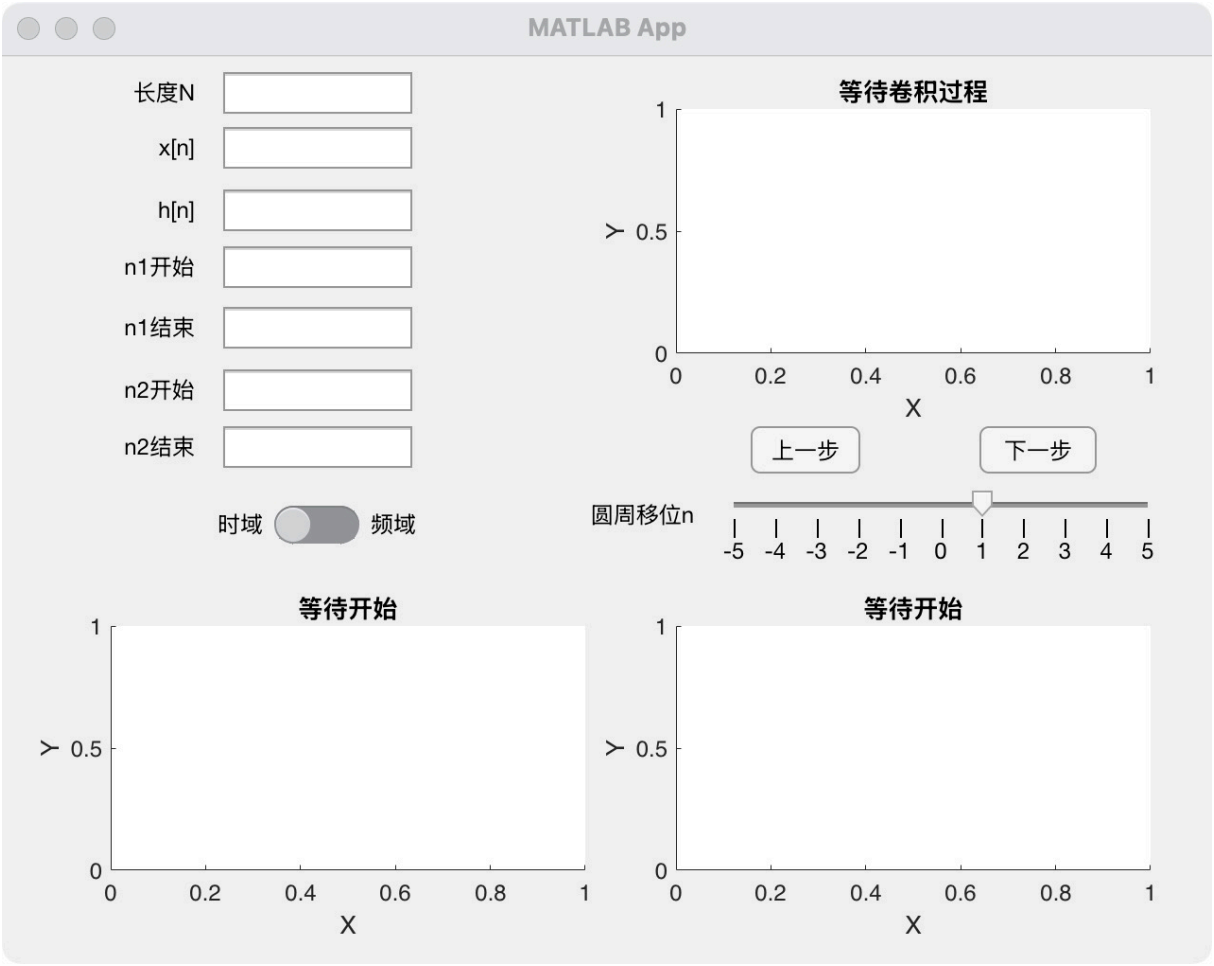


图 6: 圆周卷积计算过程设计布局图

```
        title(app.UIAxes_y, '等待卷积过程');

    % 第一步: 绘制原始信号
    elseif app.currentStep == 1
        % 获取时间范围和信号
        n1_start = str2double(app.n1StartEditField.Value);
        n1_end = str2double(app.n1EndEditField.Value);
        n2_start = str2double(app.n2StartEditField.Value);
        n2_end = str2double(app.n2EndEditField.Value);
        n1_range = n1_start:n1_end;
        n2_range = n2_start:n2_end;

        % 计算信号 x[n] 和 h[n]
        n = n1_range;
        x_signal = eval(app.xExpressionEditField.Value); % 计算 x[n] 的表达式
        n = n2_range;
        h_signal = eval(app.hExpressionEditField.Value); % 计算 h[n] 的表达式

        % 绘制原始信号
```

```
        cla(app.UIAxes_x);
        stem(app.UIAxes_x, n1_range, x_signal, 'b', 'filled');
        title(app.UIAxes_x, '原始信号 x[n]');
        xlabel(app.UIAxes_x, 'n');
        ylabel(app.UIAxes_x, 'x[n]');

        cla(app.UIAxes_h);
        stem(app.UIAxes_h, n2_range, h_signal, 'r', 'filled');
        title(app.UIAxes_h, '原始信号 h[n]');
        xlabel(app.UIAxes_h, 'n');
        ylabel(app.UIAxes_h, 'h[n]');

    % 第二步: 时域卷积 - 补零/截断+翻褶 或 频域卷积 - 计算 DFT
    elseif app.currentStep == 2
        if app.inTimeDomain
            % 时域卷积 - 先补零或截断, 再翻褶
            % 对 x_signal 和 h_signal 进行补零或截断到长度 N
            if length(x_signal) > N
                x_signal = x_signal(1:N);  % 截断 x_signal 到长度 N
            else
                x_signal = [x_signal zeros(1, N - length(x_signal))];  % 补零
            end

            if length(h_signal) > N
                h_signal = h_signal(1:N);  % 截断 h_signal 到长度 N
            else
                h_signal = [h_signal zeros(1, N - length(h_signal))];  % 补零
            end

            % 翻褶 h[n]
            h_flip = fliplr(h_signal);

            % 绘制补零/截断后的 x[n] 和 h[n]
            cla(app.UIAxes_x);
            stem(app.UIAxes_x, 0:N-1, x_signal, 'b', 'filled');
            title(app.UIAxes_x, '补零/截断后的 x[n]');
            xlabel(app.UIAxes_x, 'n');
            ylabel(app.UIAxes_x, 'x[n]');

            cla(app.UIAxes_h);
            stem(app.UIAxes_h, 0:N-1, h_flip, 'r', 'filled');
            title(app.UIAxes_h, '补零/截断并翻褶后的 h[n]');
            xlabel(app.UIAxes_h, 'n');
            ylabel(app.UIAxes_h, 'h[n]');
        else
            % 频域卷积 - 计算 DFT
            X = fft(x_signal, N);
            H = fft(h_signal, N);
```

```
            % 绘制 DFT 结果
            cla(app.UIAxes_x);
            stem(app.UIAxes_x, 0:length(X)-1, abs(X), 'b', 'filled');  % 显示频
                域的幅度
            title(app.UIAxes_x, 'DFT of x[n]');
            xlabel(app.UIAxes_x, 'k');
            ylabel(app.UIAxes_x, '|X[k]|');

            cla(app.UIAxes_h);
            stem(app.UIAxes_h, 0:length(H)-1, abs(H), 'r', 'filled');  % 显示频
                域的幅度
            title(app.UIAxes_h, 'DFT of h[n]');
            xlabel(app.UIAxes_h, 'k');
            ylabel(app.UIAxes_h, '|H[k]|');
        end

    % 第三步: 时域卷积 - 圆周移位 或 频域卷积 - 点乘
    elseif app.currentStep == 3
        if app.inTimeDomain
            % 时域卷积 - 圆周移位
            shift = round(app.nSlider.Value);
            h_shift = circshift(h_flip, shift);

            % 绘制移位后的 h[n]
            cla(app.UIAxes_y);
            stem(app.UIAxes_y, 0:length(h_shift)-1, h_shift, 'r', 'filled');
            title(app.UIAxes_y, ['圆周移位后的 h[n], 移位 n = ', num2str(shift)
                ]);
            xlabel(app.UIAxes_y, 'n');
            ylabel(app.UIAxes_y, 'h[n]');
        else
            % 频域卷积 - 点乘
            X = fft(x_signal, N);
            H = fft(h_signal, N);
            Y_freq = X .* H;

            % 绘制频域中的点乘结果
            cla(app.UIAxes_y);
            stem(app.UIAxes_y, 0:length(Y_freq)-1, abs(Y_freq), 'm', 'filled');
                  % 显示频域乘积的幅度
            title(app.UIAxes_y, '频域点乘结果 |Y[k]|');
            xlabel(app.UIAxes_y, 'k');
            ylabel(app.UIAxes_y, '|Y[k]|');
        end

    % 第四步: 时域卷积 - 相乘 或 频域卷积 - IFFT
    elseif app.currentStep == 4
```

```
        if app.inTimeDomain
            % 时域卷积 - 相乘
            shift = round(app.nSlider.Value);
            h_shift = circshift(h_flip, shift);

            % 检查 x_signal 和 h_shift 的长度是否相等
            if length(x_signal) == length(h_shift)
                product = x_signal .* h_shift;
            else
                error('x_signal 和 h_shift 的长度不一致, 无法相乘');
            end

            % 绘制相乘结果
            cla(app.UIAxes_y);
            stem(app.UIAxes_y, 0:length(product)-1, product, 'g', 'filled');
            title(app.UIAxes_y, 'x[n] 和 h[n] 的相乘结果');
            xlabel(app.UIAxes_y, 'n');
            ylabel(app.UIAxes_y, '乘积');
        else
            % 频域卷积 - 计算 IFFT
            X = fft(x_signal, N);
            H = fft(h_signal, N);
            Y_freq = X .* H;
            y_circular = ifft(Y_freq, N);

            % 绘制 IFFT 结果
            cla(app.UIAxes_y);
            stem(app.UIAxes_y, 0:length(y_circular)-1, y_circular, 'm', 'filled
                ');
            title(app.UIAxes_y, '圆周卷积结果 (频域转换回时域) ');
            xlabel(app.UIAxes_y, 'n');
            ylabel(app.UIAxes_y, 'y[n]');
        end
    % 第五步: 显示卷积结果
    elseif app.currentStep == 5
        if app.inTimeDomain
            % 时域卷积
            y_circular = cconv(x_signal, h_signal, N);

            cla(app.UIAxes_y);
            stem(app.UIAxes_y, 0:length(y_circular)-1, y_circular, 'm', 'filled
                ');
            title(app.UIAxes_y, '圆周卷积结果 (时域) ');
            xlabel(app.UIAxes_y, 'n');
            ylabel(app.UIAxes_y, 'y[n]');
        else
            % 频域卷积
            X = fft(x_signal, N);
```

```
            H = fft(h_signal, N);
            Y_freq = X .* H;
            y_circular = ifft(Y_freq, N);

            cla(app.UIAxes_y);
            stem(app.UIAxes_y, 0:length(y_circular)-1, y_circular, 'm', 'filled
                ');
            title(app.UIAxes_y, '圆周卷积结果 (频域) ');
            xlabel(app.UIAxes_y, 'n');
            ylabel(app.UIAxes_y, 'y[n]');
        end
    end

    end

    % Value changed function: nSlider
    function StepSliderValueChanged2(app, event)
        app.updateConvolutionProcess();
    end

    % Value changed function: Button_pre
    function Button_preValueChanged(app, event)
        if app.currentStep > 0
            app.currentStep = app.currentStep - 1;
            app.updateConvolutionProcess();
        end
    end

    % Value changed function: Button_next
    function ButtonValueChanged(app, event)
        if app.currentStep < 5
            app.currentStep = app.currentStep + 1;
            app.updateConvolutionProcess();
        end
    end

    % Value changed function: Switch
    function SwitchValueChanged(app, event)
        value = app.Switch.Value;    % 根据 Switch 的值更新 inTimeDomain 变量
        if strcmp(value, '时域')  % 如果开关设置为时域
            app.inTimeDomain = true;  % 设置为时域
        elseif strcmp(value, '频域')  % 如果开关设置为频域
            app.inTimeDomain = false; % 设置为频域
        end
        app.updateConvolutionProcess(); % 切换后更新卷积过程
    end
end
```

4.6 线性卷积和圆周卷积的对比

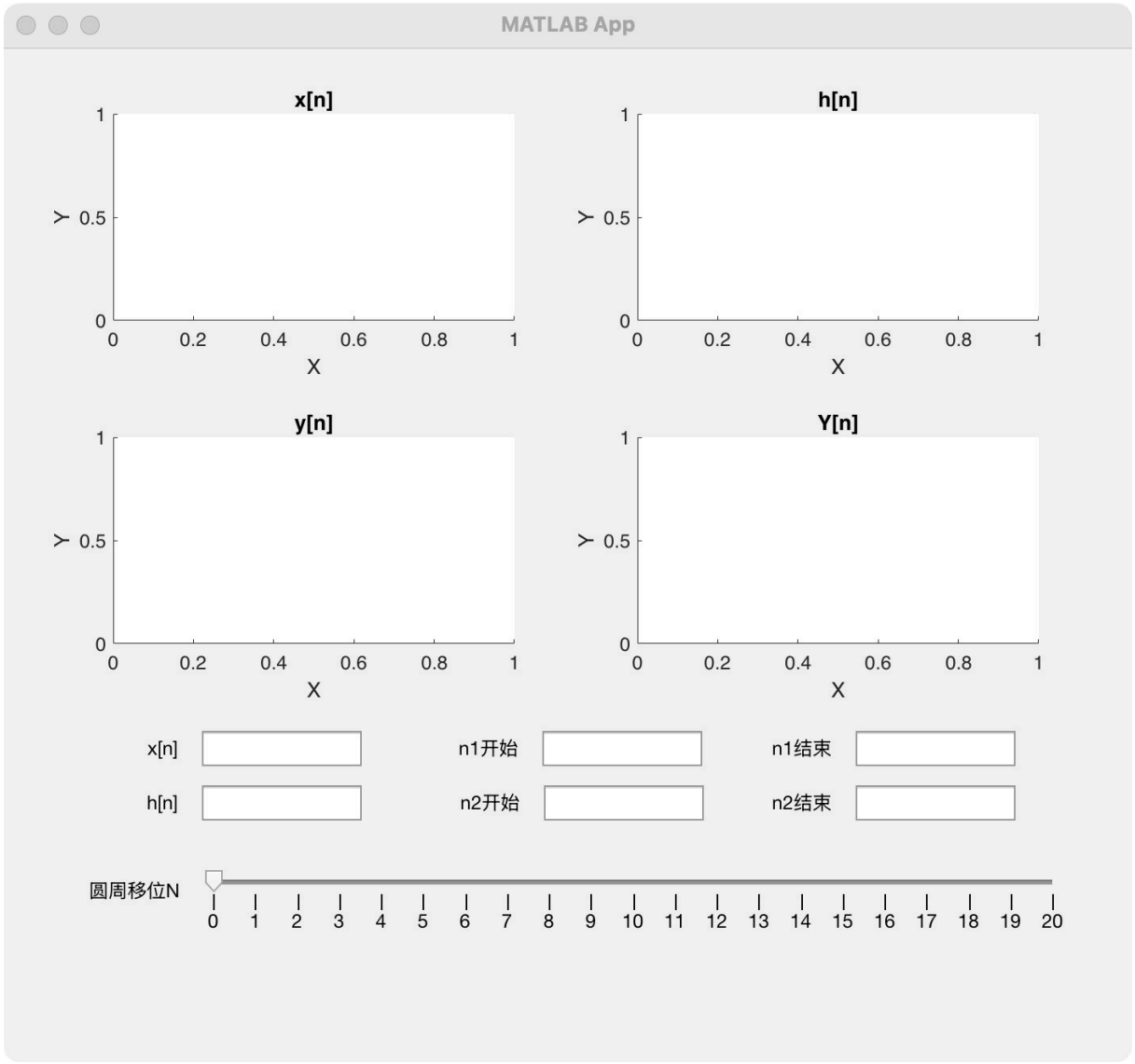


图 7: 两种卷积设计布局图

```
        % Code that executes after component creation
    function updateConvolutionProcess(app)
        persistent x h n1 n2; % 持久化变量

        % 获取 N 的值 (圆周卷积长度)
        N = round(app.NSlider.Value);

        % 如果 N = 1, 绘制线性卷积和圆周卷积
        if N == 1
            % 获取用户输入的表达式
            x_expr = app.xExpressionEditField.Value;
            h_expr = app.hExpressionEditField.Value;
```

```
% 获取时间轴范围
n1_start = str2double(app.n1StartEditField.Value);
n1_end = str2double(app.n1EndEditField.Value);
n2_start = str2double(app.n2StartEditField.Value);
n2_end = str2double(app.n2EndEditField.Value);

n1 = n1_start:n1_end;
n2 = n2_start:n2_end;

% 计算信号 x[n]
n = n1; % 为了在表达式中使用
x = eval(x_expr);

% 计算信号 h[n]
n = n2;
h = eval(h_expr);

% 计算线性卷积 y[n]
y_linear = conv(x, h);
ny = (n1_start + n2_start):(n1_end + n2_end);
cla(app.UIAxes1);  % 清除先前的图形
stem(app.UIAxes1, n1, x, 'b', 'filled');
title(app.UIAxes1, '信号 x[n]');
xlabel(app.UIAxes1, 'n');
ylabel(app.UIAxes1, 'x[n]');

cla(app.UIAxes2);  % 清除先前的图形
stem(app.UIAxes2, n2, h, 'r', 'filled');
title(app.UIAxes2, '信号 h[n]');
xlabel(app.UIAxes2, 'n');
ylabel(app.UIAxes2, 'h[n]');
% 绘制线性卷积
cla(app.UIAxes3);  % 清除之前的图形
stem(app.UIAxes3, ny, y_linear, 'b', 'filled');
title(app.UIAxes3, '线性卷积结果 y[n]');
xlabel(app.UIAxes3, 'n');
ylabel(app.UIAxes3, 'y[n]');

% 对 x 和 h 进行补零以执行圆周卷积
if length(x) > N
    x_circ = x(1:N);  % 截断
else
    x_circ = [x zeros(1, N - length(x))];  % 补零
end

if length(h) > N
    h_circ = h(1:N);  % 截断
```

```
                else
                    h_circ = [h zeros(1, N - length(h))];  % 补零
                end

                % 计算圆周卷积
                y_circ = cconv(x_circ, h_circ, N);
                ny_circ = 0:(N-1);

                % 绘制圆周卷积
                cla(app.UIAxes4);  % 清除之前的图形
                stem(app.UIAxes4, ny_circ, y_circ, 'r', 'filled');
                title(app.UIAxes4, '圆周卷积结果 y_c[n]');
                xlabel(app.UIAxes4, 'n');
                ylabel(app.UIAxes4, 'y_c[n]');

            % 如果 N > 1，仅绘制圆周卷积
            elseif N > 1
                % 对 x 和 h 进行补零以执行圆周卷积
                if length(x) > N
                    x_circ = x(1:N);  % 截断
                else
                    x_circ = [x zeros(1, N - length(x))];  % 补零
                end

                if length(h) > N
                    h_circ = h(1:N);  % 截断
                else
                    h_circ = [h zeros(1, N - length(h))];  % 补零
                end

                % 计算圆周卷积
                y_circ = cconv(x_circ, h_circ, N);
                ny_circ = 0:(N-1);

                % 绘制圆周卷积
                cla(app.UIAxes4);  % 清除之前的图形
                stem(app.UIAxes4, ny_circ, y_circ, 'r', 'filled');
                title(app.UIAxes4, ['圆周卷积结果 y_c[n] (N = ', num2str(N), ')']);
                xlabel(app.UIAxes4, 'n');
                ylabel(app.UIAxes4, 'y_c[n]');
            end
        end
```

4.7 Matlab 验证其他定理

```
    % 定义三个信号 x[n], h[n] 和 g[n]
x = [1, 2, 3];
```

```
h = [4, 5, 6];
g = [7, 8];

% 定义标准冲激响应函数 [n]
n_delta = 0:10; % 时间范围
delta = [1, zeros(1, length(n_delta)-1)]; % 冲激响应

% 设置图形窗口
figure;

%% 1. 显示原始信号 x[n]
subplot(4, 2, 1); % 图1: 显示信号 x[n]
stem(0:length(x)-1, x, 'b', 'filled');
title('信号 x[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

%% 2. 显示原始信号 h[n]
subplot(4, 2, 2); % 图2: 显示信号 h[n]
stem(0:length(h)-1, h, 'r', 'filled');
title('信号 h[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

%% 3. 显示原始信号 g[n]
subplot(4, 2, 3); % 图3: 显示信号 g[n]
stem(0:length(g)-1, g, 'g', 'filled');
title('信号 g[n]');
xlabel('n');
ylabel('Amplitude');
grid on;

%% 4. 显示标准的冲激响应函数 [n]
subplot(4, 2, 4); % 图4: 显示标准冲激响应 [n]
stem(n_delta, delta, 'm', 'filled');
title('标准冲激响应 [n]');
xlabel('n');
ylabel('Amplitude');
grid on;

%% 5. 交换律验证
subplot(4, 2, 5); % 图5: 验证 x[n] * h[n] 与 h[n] * x[n] 的交换律
conv_xh = conv(x, h);
conv_hx = conv(h, x);
stem(0:length(conv_xh)-1, conv_xh, 'b', 'filled');
hold on;
```

```
stem(0:length(conv_hx)-1, conv_hx, 'r--');
title('交换律验证');
xlabel('n');
ylabel('Amplitude');
legend('x[n] * h[n]', 'h[n] * x[n]');
grid on;
hold off;

%% 6. 结合律验证
subplot(4, 2, 6);  % 图6: 验证结合律 (x[n] * h[n]) * g[n] = x[n] * (h[n] * g[n])
conv_xh = conv(x, h);
conv_xh_g = conv(conv_xh, g);
conv_hg = conv(h, g);
conv_x_hg = conv(x, conv_hg);
stem(0:length(conv_xh_g)-1, conv_xh_g, 'b', 'filled');
hold on;
stem(0:length(conv_x_hg)-1, conv_x_hg, 'r--');
title('结合律验证');
xlabel('n');
ylabel('Amplitude');
legend('(x[n] * h[n]) * g[n]', 'x[n] * (h[n] * g[n])');
grid on;
hold off;

%% 7. 分配律验证
subplot(4, 2, 7);  % 图7: 验证分配律 x[n] * (h[n] + g[n]) = x[n] * h[n] + x[n] * g[
    n]
% 将 g[n] 进行零填充, 使其与 h[n] 长度一致
g_padded = [g, zeros(1, length(h) - length(g))];  % g[n] 零填充
conv_xh = conv(x, h);  % x[n] * h[n]
conv_xg = conv(x, g_padded);  % x[n] * g[n]
conv_x_h_plus_g = conv(x, h + g_padded);  % x[n] * (h[n] + g[n])
conv_xh_plus_xg = conv_xh + conv_xg;  % x[n] * h[n] + x[n] * g[n]
stem(0:length(conv_x_h_plus_g)-1, conv_x_h_plus_g, 'b', 'filled');
hold on;
stem(0:length(conv_xh_plus_xg)-1, conv_xh_plus_xg, 'r--');
title('分配律验证');
xlabel('n');
ylabel('Amplitude');
legend('x[n] * (h[n] + g[n])', 'x[n] * h[n] + x[n] * g[n]');
grid on;
hold off;

%% 8. 冲激响应验证
subplot(4, 2, 8);  % 图8: h[n] 与  [n] 的卷积验证
% 将 h[n] 进行零填充, 以匹配  [n] 的长度
h_padded = [h, zeros(1, length(delta) - length(h))];  % h[n] 零填充
conv_h_delta = conv(h_padded, delta);  % h[n] 与  [n] 的卷积
```

```
stem(0:length(conv_h_delta)-1, conv_h_delta, 'g', 'filled');
hold on;
stem(0:length(h)-1, h, 'r--');
title('h[n] 与  [n] 的卷积验证');
xlabel('n');
ylabel('Amplitude');
legend('h[n] *  [n]', 'h[n]');
grid on;
hold off;

% 调整图像布局
sgtitle('卷积定理验证');
```

5 实验结果和分析

5.1 4.1 实验结果

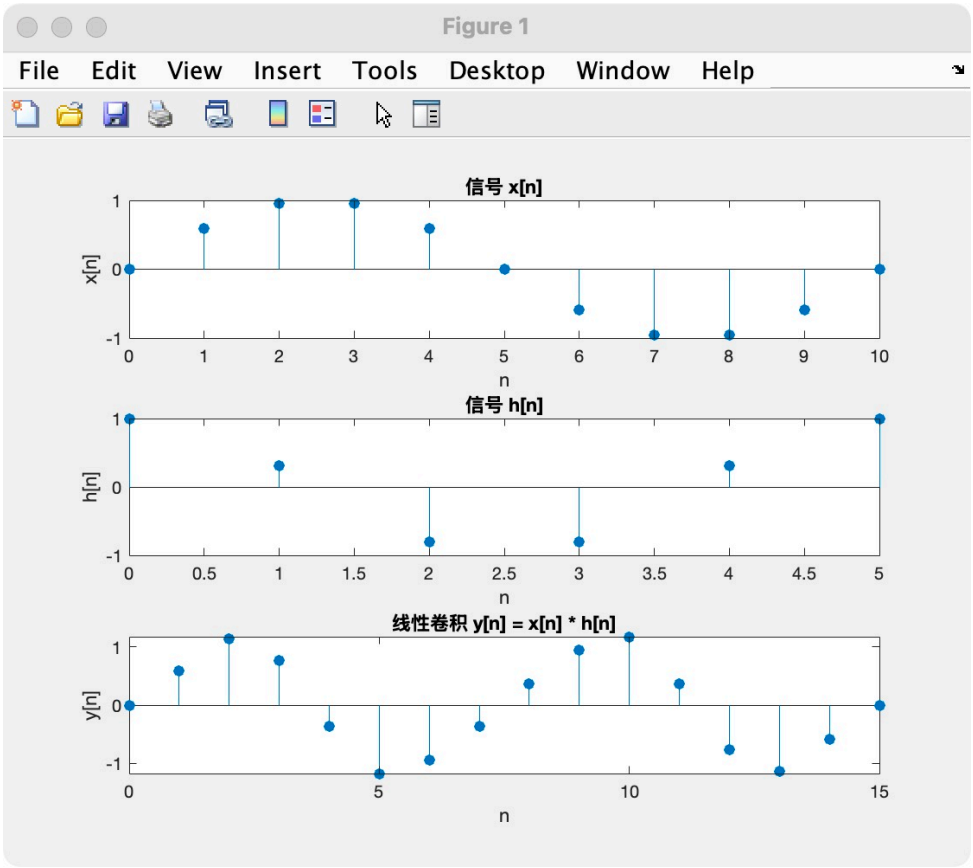
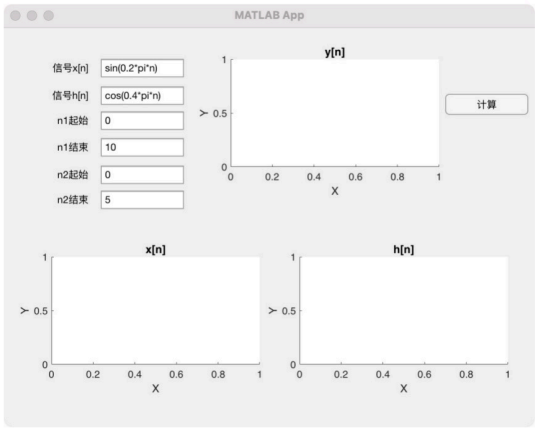
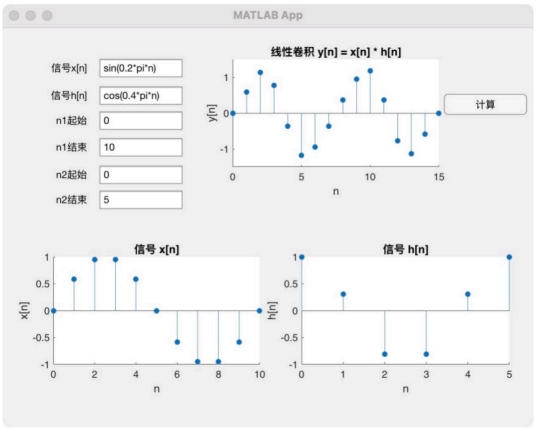


图 8: Matlab 卷积结果

5.2 4.2 实验结果



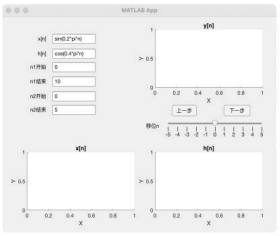
Step 1:输入对应值



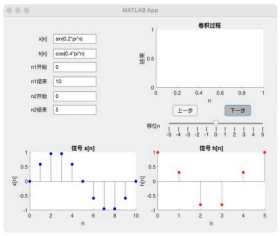
Step 2:单击计算按钮

图 9: 卷积计算结果

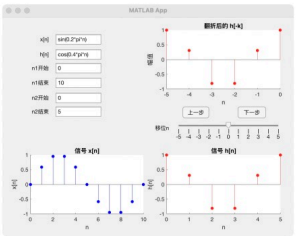
5.3 4.4 实验结果



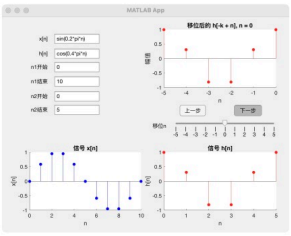
Step 1:输入对应值



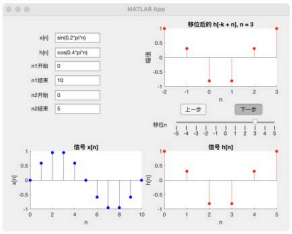
Step 2:单击下一步



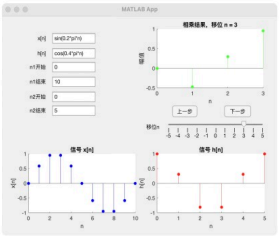
Step 3:单击下一步



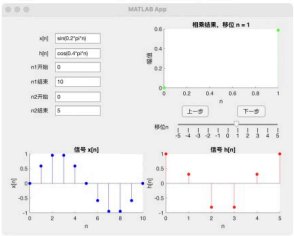
Step 4:单击下一步



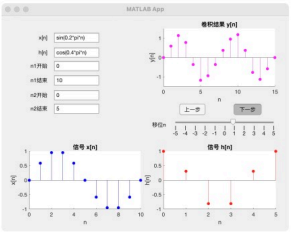
Step 5:移动滑块
显示不同n下的结果



Step 6:单击下一步



Step :7移动滑块
显示不同n下的结果



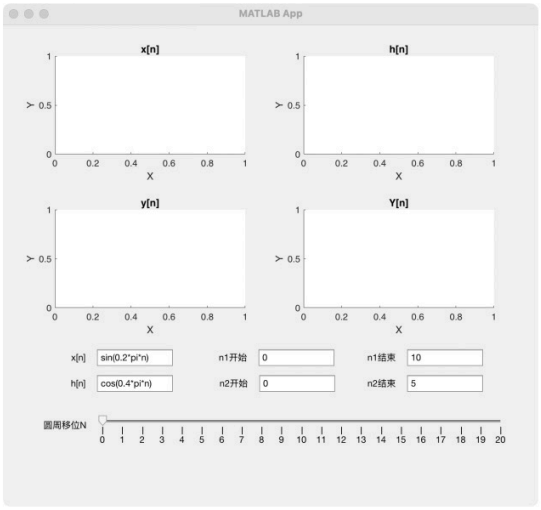
Step 8:单击下一步

图 10: 卷积计算过程结果

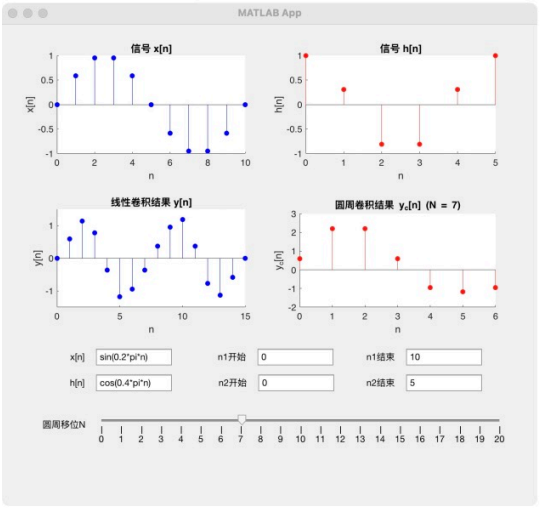
5.4 4.5 实验结果

图 11: 圆周卷积计算过程结果

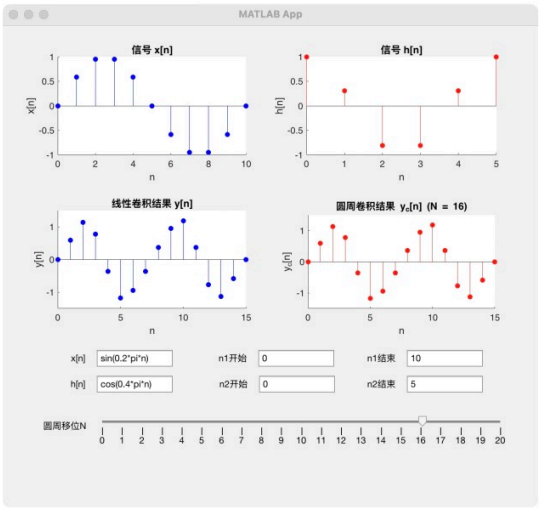
5.5 4.6 实验结果



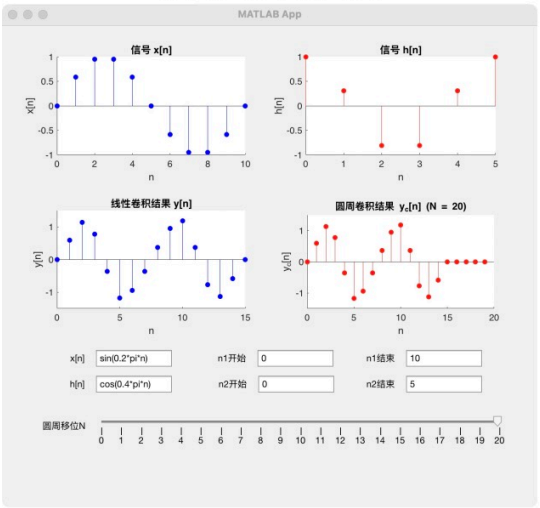
Step 1:输入对应值



Step 2:移动滑块
显示不同长度N下圆周卷积
和线性卷积的对比



Step 3:移动滑块
显示不同长度N下圆周卷积
和线性卷积的对比



Step 4:移动滑块
显示不同长度N下圆周卷积
和线性卷积的对比

图 12: 两种卷积对比结果

5.6 4.7 实验结果

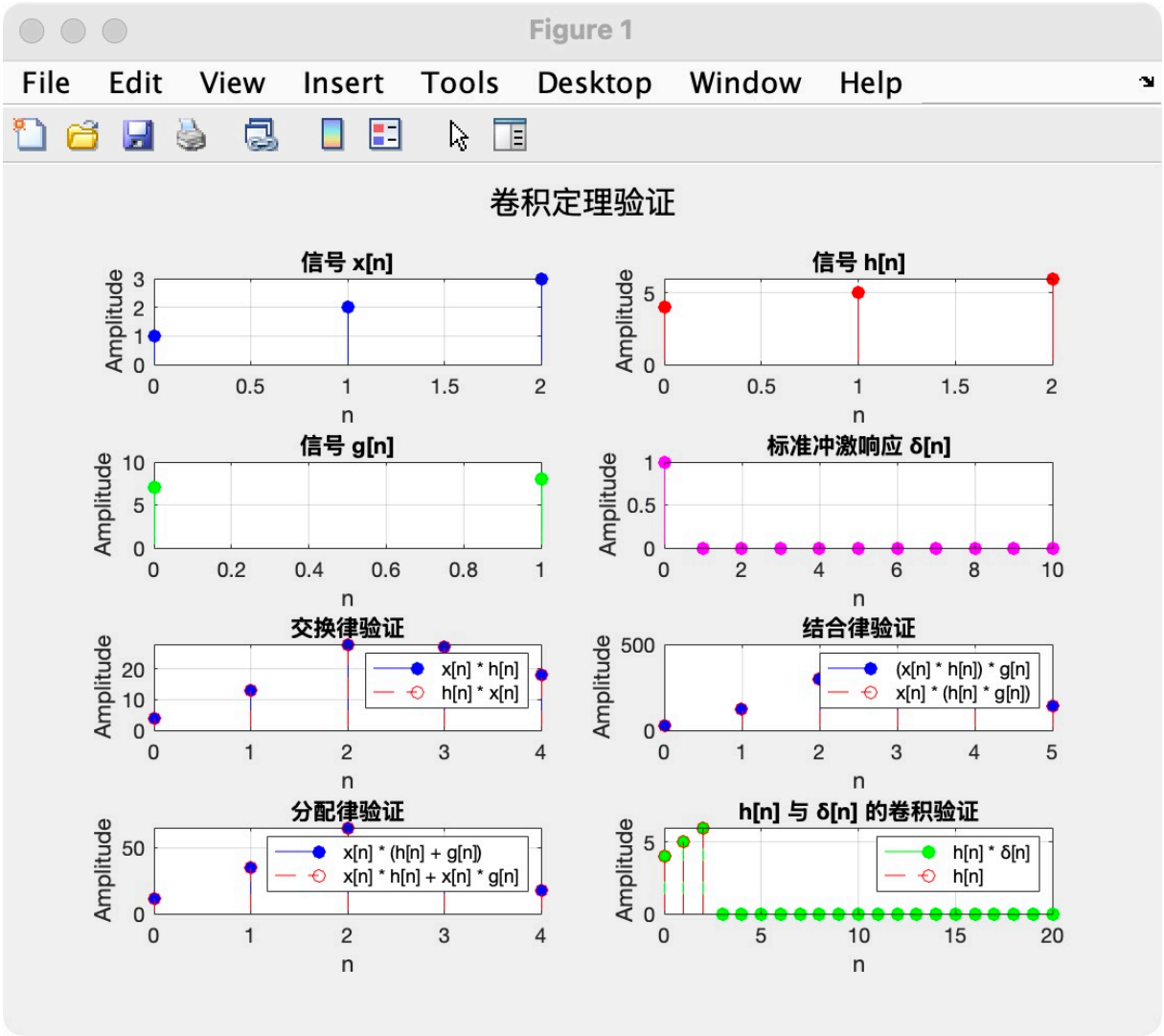


图 13: 其他定理验证结果

5.7 实验分析

本实验通过 MATLAB[7] 和 App Designer[8] 实现了卷积运算的计算、可视化和交互演示。首先，利用 MATLAB 中的 `conv` 函数，完成了线性卷积的计算。根据公式 (1)，线性卷积[9] 的结果长度 $L_y = L_x + L_h - 1$ 与实验中信号的长度吻合。输入信号 $x[n]$ 的长度为 11，信号 $h[n]$ 的长度为 6，卷积结果 $y[n]$ 的长度为 16。这表明，线性卷积通过将信号 $x[n]$ 和 $h[n]$ 翻转并移位，实现了信号在时域上的累积和扩展。通过实验绘图，观察到了两个信号的叠加效果及其相乘累加的卷积结果，这验证了线性卷积在信号处理中的作用。

随后，通过 App Designer 设计了一个交互式图形界面，允许用户输入自定义的信号表达式并查看卷积结果。这种设计提供了更灵活的信号输入和结果展示，用户可以自由调整输入信号的形式，实时查看卷积的计算结果。实验结果显示，输入不同的正弦和余弦函数，卷积后的结果展示了信号

的叠加和相位的变化，卷积运算在教学演示中表现出了良好的可操作性和可视化效果。

在卷积计算过程中，通过信号的叠加可以观察到卷积结果的长度和形状。这一叠加过程展示了信号经过线性系统后，其频率成分的变化和时间延长的效果。信号 $h[n]$ 被视为系统的脉冲响应，通过卷积计算得到了输入信号 $x[n]$ 经过系统后的输出，体现了卷积运算在滤波器设计中的应用。

卷积计算的可视化过程展示了卷积的具体步骤。首先，信号 $h[n]$ 被时间翻转为 $h(-n)$，然后按照移位参数进行平移，通过 GUI 界面用户可以控制滑块来观察不同移位结果与 $x[n]$ 的卷积。信号相乘和相加的过程逐步展示了卷积的每一个计算步骤，帮助用户直观理解卷积是如何通过移位、相乘和累加操作来得到最终的卷积结果。

在圆周卷积[3] 的计算中，我使用了 MATLAB 的 `cconv` 函数。根据公式 (2)，圆周卷积对有限长信号的处理方式与线性卷积不同，信号在计算时会进行周期延拓。在实验中，用户可以通过控制圆周卷积的长度 N 来观察信号的周期性影响。当 N 小于信号总长度时，卷积结果出现了周期重叠的现象，而当 N 满足 $N \geq L_x + L_h - 1$ 时，圆周卷积的结果与线性卷积相同，这验证了圆周卷积在周期信号处理中的优势。

通过补零操作，实验对比了线性卷积与圆周卷积的结果。圆周卷积适合处理周期信号，而线性卷积则适用于无限长的非周期信号。当信号长度满足 $N \geq L_x + L_h - 1$ 时，线性卷积和圆周卷积的结果一致，但在 N 较小时，圆周卷积由于周期性产生了边界效应，结果出现了重叠。

最后，实验验证了卷积的交换律、结合律和分配律。根据公式 (**??**)，信号 $x[n]$ 和 $h[n]$ 的卷积结果满足 $x[n] * h[n] = h[n] * x[n]$。实验中，通过对不同信号进行卷积运算，观察到了信号卷积结果的一致性，证明了卷积运算的交换律和结合律。分配律的验证表明，卷积运算具有线性叠加特性，即 $x[n] * (h[n] + g[n]) = x[n] * h[n] + x[n] * g[n]$。这些定理的验证证明了 MATLAB 实现的卷积函数在理论上的准确性。

6 实验总结和思考

本次实验成功实现了线性卷积和圆周卷积的可视化和交互式操作，特别是通过 App Designer 提升了实验的直观性和可操作性。使用 App Designer 设计的 GUI 界面，不仅让用户能够通过滑块和按钮等控件自由控制卷积计算的每一步，还能通过实时更新信号图形，观察到卷积运算中每一个步骤对结果的影响。这种交互式设计大大增强了实验的演示效果，使得复杂的卷积运算变得更易于理解和操作。

App Designer 的使用极大地提高了卷积运算的可视化效果。在卷积过程的可视化中，用户可以通过界面直观观察到卷积的翻转、移位、相乘和累加操作，从而对卷积运算有了更为深入的理解。通过滑块控制卷积过程的移位步长，用户可以实时观察不同移位量下的卷积结果变化。这种实时交互设计让卷积理论的演示更加生动，也帮助我在学习信号处理时能够更加清晰地理解卷积的本质。

此外，FFT 的应用让我更好地理解了频域卷积的优势。在实验中，FFT 的引入极大提升了卷积运算的效率，特别是在处理长信号时，FFT 大幅缩短了运算时间，验证了卷积在大规模信号处理中具有的计算优势。相比于时域卷积，频域卷积在实际应用中具有更高的效率。

通过本次实验，我不仅掌握了卷积理论及其在 MATLAB 中的实现，还学会了如何设计图形用户界面来增强实验的交互性和可视化效果。未来我将继续优化 App Designer 界面设计，探索更多信号处理算法的应用场景，并结合实际工程项目，进一步提升我在信号处理和 MATLAB 编程方面的能力。本次实验的收获对我未来的信号处理学习和实践具有重要的推动作用。

参考文献

[1] 俞一彪 and 孙兵. 数字信号处理. 南京东南大学出版社, 2021.

[2] 伍世云 and 王益艳. “Matlab 在 DSP 课程教学中的几个典型应用”. In: 计算机与现代化 08 (2011), pp. 185–189. ISSN: 1006-2475.

[3] Forester W. Isen. *DSP for MATLAB™ and LabVIEW™ II:Discrete Frequency Transforms*. Morgan Claypool Publishers, 2009.

[4] Aly El-Osery. “MATLAB Tutorial”. In: *October.. http://www. ee. nmt. edu* (2004).

[5] 李楠, 冯明军, and 自兴发. “基于 MATLAB 的线性卷积实现”. In: 楚雄师范学院学报 25.12 (2010), pp. 56–59+65. ISSN: 1671-7406.

[6] 陈琳. “基于 MATLAB 的线性卷积及其快速实现方法”. In: 电脑与信息技术 21.04 (2013), pp. 29–31. ISSN: 1005-1228. DOI: 10.19414/j.cnki.1005-1228.2013.04.009.

[7] 李军成, 杨炼, and 刘成志. *MATLAB* 语言基础. 南京大学出版社, 2022.

[8] 李星新 and 黄熹. *MATLAB 2020 GUI* 程序设计从入门到精通. 机械工业出版社, 2021.

[9] 张登奇 and 陈佳. “离散卷积的算法分析及 MATLAB 实现”. In: 湖南理工学院学报 *(*自然科学版*)* 26.02 (2013), pp. 48–52+89. ISSN: 1672-5298.