Linear and Circular Convolution

Objectives

1. To understand the concepts of linear and circular convolution in signal processing and their applications.
2. To compare manual, FFT-based, and MATLAB built-in function methods for performing convolution operations.
3. To analyze the impact of zero-padding on linear and circular convolution results.
4. To explore the use of MATLAB library functions (`conv, cconv, conv2`) for 1D and 2D convolution.

Apparatus

Software: MATLAB R2022b

Hardware: Personal Computer

Theory

Digital convolution plays an important role in digital filtering, utilized to determine the interaction between two signals. This operation is critical in analyzing systems such as Linear Time-Invariant (LTI) systems, where it helps determine its unit-impulse response h(n), which relates the system input and output. Convolution can be classified into linear convolution and circular convolution, each serving distinct purposes and applications.

Linear convolution is used to compute the output of an LTI system when an input signal is passed through it. It provides the true response of the system to the input. For two discrete-time sequences x[n] (input signal) and h[n] (impulse response), the linear convolution is mathematically expressed as follows.

$$\boldsymbol{y_{lc}[n]} = \mathbf{x[n]} * \mathbf{h[n]} = \sum_{\boldsymbol{k=-\infty}}^{\infty} \boldsymbol{x[k]} \cdot \boldsymbol{h[n-k]} = \sum_{\boldsymbol{k=-\infty}}^{\infty} \boldsymbol{h[k]} \cdot \boldsymbol{x[n-k]}$$

Here, the result $y_{lc}[n]$ has a length of N+M -1, where N and M are the lengths of x[n] and h[n], respectively. Note that for a causal system, which implies an impulse response of h[n] = 0 for n < 0 and the lower limit of the convolution sum begins at 0 instead of ∞, that is

$$\boldsymbol{y_{lc}[n]} = \sum_{\boldsymbol{k=0}}^{\infty} \boldsymbol{x[k]} \cdot \boldsymbol{h[n-k]} = \sum_{\boldsymbol{k=0}}^{\infty} \boldsymbol{h[k]} \cdot \boldsymbol{x[n-k]}$$

- Circular Convolution

  Circular convolution is used when signals are treated as periodic. Unlike linear convolution, it assumes the sequences wrap around, producing a result that exhibits periodicity. The circular convolution of two discrete-time sequences x[n] and h[n], both of length N, is given by the following expression.

$$\boldsymbol{y_{cc}[n] = \mathbf{x[n]} \circledast \mathbf{h[n]} = \sum_{k=0}^{N-1} x[k] \cdot h[(n-k) \bmod N]}$$

  Here, N is the length of the periodic sequence, and the modulus operator (mod N) enforces periodic indexing. The result $y_{cc}[n]$ will also have a length of N, as circular convolution assumes periodicity and does not extend the length of the sequences. Circular convolution is particularly useful in applications involving the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) due to its computational efficiency and compatibility with frequency-domain operations.

- Convolutions and Zero Padding

  Padding in DSP refers to the process of adding extra data, typically zeros, to a signal to achieve a specific purpose or meet certain computational requirements. This technique is widely used in various signal processing applications, including convolution, Fourier transforms, and filtering. In the case of linear convolution, zero-padding is commonly applied to extend the signal lengths to N+M−1, where N and M are the lengths of the two input sequences. This extension ensures that the output accurately represents the true linear convolution result without aliasing or unwanted periodic artifacts. Padding is particularly critical in FFT-based computations, as it prevents wraparound effects and maintains the integrity of the convolution operation.

  Moreover, in circular convolution for the sequences of unequal lengths, zero-padding is used to make their lengths equal to max (N, M). This alignment ensures accurate computation while preserving periodicity. Zero-padding not only resolves length mismatches but also improves computational efficiency in FFT operations by allowing the use of power-of-two lengths.

- Convolution Techniques

  Convolution can be computed using various methods, including FFT-based approaches, direct computation techniques, and built-in functions, for both linear and circular convolution.

  The FFT method leverages the convolution theorem, which states that convolution in the time domain corresponds to pointwise multiplication in the frequency domain. This allows for efficient computation of convolutions, particularly when working with large datasets.

FFT-based techniques are especially valuable for circular convolution but can also be used for linear convolution with proper handling of sequence lengths.

Manual convolution techniques involve directly calculating the sum of products between the two sequences at each shift index. This method is most useful for small datasets or for understanding the underlying mechanics of convolution. While manual convolution can be computationally expensive for large sequences, it is essential for learning and implementing convolution in the time domain.

MATLAB offers built-in functions that make the implementation of convolution straightforward. These functions are optimized for performance and are widely used in practical applications. Examples include "`conv`" for linear convolution and "`cconv`" for circular convolution in MATLAB. These built-in tools provide an efficient and error-free way to compute convolution, making them particularly useful in engineering and research tasks.

MATLAB code Overview

1. Input Sequence Definition
   Two discrete sequences, x[n] and h[n] are defined in this step. These sequences represent fundamental input signals that will be used for both linear and circular convolution operations.

   - MATLAB code for defining the Input Sequences is,

```
% Input sequences
x = [1 2 3 4];  % Input sequence x[n]
h = [1 2 1 2];  % Input sequence h[n]
```

2. Linear Convolution (Manual)
   Linear convolution is computed manually by directly summing the products of the input sequences at each shift index. The output "ylc_`manual`" represents the result of this manual convolution process.

   - MATLAB code for Linear Convolution (Manual) is,

```
m = length(x);
n = length(h);
L = m + n - 1;
N = max(m, n);
if m < n
    x = [x zeros(1, N - m)];
    h = [h zeros(1, N - n)];
end
```

```
ylc_manual = zeros(1, length(x) + length(h) - 1);
for n = 1:length(ylc_manual)
    for k = 1:length(x)
     if (n - k + 1) > 0 && (n - k + 1) <= length(h)
 ylc_manual(n) = ylc_manual(n) + x(k) * h(n - k + 1);
        end
    end
end
```

3. Circular Convolution (Manual)
   Circular convolution is computed manually by using the "`mod`" function to handle the wrapping around of the sequences.

   • MATLAB code for Circular Convolution (Manual) is,

```
ycc_manual = zeros(1, N);
for k = 0:N-1
    sum = 0;
    for j = 0:N-1
        sum = sum + x(mod(k - j, N) + 1) * h(j + 1);
    end
    ycc_manual(k + 1) = sum;
end
```

4. Linear Convolution using FFT
   Linear convolution can be computed using the FFT and its inverse (IFFT). The output, "`ylc_fft`", represents the result of linear convolution computed using FFT.

   • MATLAB code for FFT-Based Linear Convolution is,

```
% Linear convolution using FFT
x_padded = [x zeros(1, L - N)];  % Zero-pad x to L samples
for linear convolution via FFT
h_padded = [h zeros(1, L - N)];  % Zero-pad h to L samples
for linear convolution via FFT
ylc_fft = ifft(fft(x_padded) .* fft(h_padded)); % Linear
convolution using FFT
```

5. FFT-Based Circular Convolution
   Circular convolution can be efficiently computed using FFT by transforming two discrete-time sequences, x[n] and h[n], into the frequency domain, multiplying their

FFTs element-wise, and applying the Inverse FFT (IFFT) to obtain the result in the time domain.

- MATLAB code for FFT-Based Circular Convolution:

```
% Circular convolution using FFT
ycc_fft = ifft(fft(x).* fft(h));
```

6. Linear Convolution using Built-in Function
   MATLAB provides a built-in function, "`conv`", to compute the linear convolution of two sequences.

- MATLAB code for Linear Convolution using Built-in Function is,

```
% Linear convolution using built-in function
ylc_builtin = conv(x, h);
```

7. Circular Convolution using Built-in Function
   The built-in function "`cconv`" is used to compute the circular convolution of two sequences.
- MATLAB code for Linear Convolution using Built-in Function is,

```
% Circular convolution using built-in function
ycc_builtin = cconv(x, h, N);
```

Procedure

1. Open MATLAB and create a new script by navigating to **Home → New → Script**

2. Define two input sequences x[n] = [1, 2, 3, 4] and h[n] = [1, 2, 1, 2], to perform both linear and circular convolution operations.

3. Determine the lengths of the sequences x[n] and h[n] to calculate the length of the convolution outputs and implement a loop-based approach to compute the linear convolution manually by summing the products of the sequences at each shift index.

4. Zero-pad the sequences to the same length, then use a nested loop to compute the circular convolution manually. Applying the modulo operation to handle the wrapping of sequences when their indices exceed the signal length.

5. Use the FFT to compute the linear convolution and circular convolution.

6. Use MATLAB's built-in "`conv()`" function to compute the linear convolution of x[n] and h[n] using built-in function.

7. Use MATLAB's "`cconv()`" built-in function to compute the circular convolution. Ensure the sequences are zero-padded to the same length before applying the function.

8. Use the "`stem()`" function to visualize the results of manual, FFT-based, and built-in convolutions. Create three separate figures, each figure containing three subplots: the first subplot shows the input sequence, followed by the corresponding linear convolution result in the second subplot, and the circular convolution result in the third subplot.

9. Save the script with an appropriate name, (e.g., convolution_comparison.m)

10. Execute the script to observe the outputs. Verify and compare the results of linear and circular convolutions using different methods.

11. Save the figure by selecting **File → Save As** in the Figure window.

Results

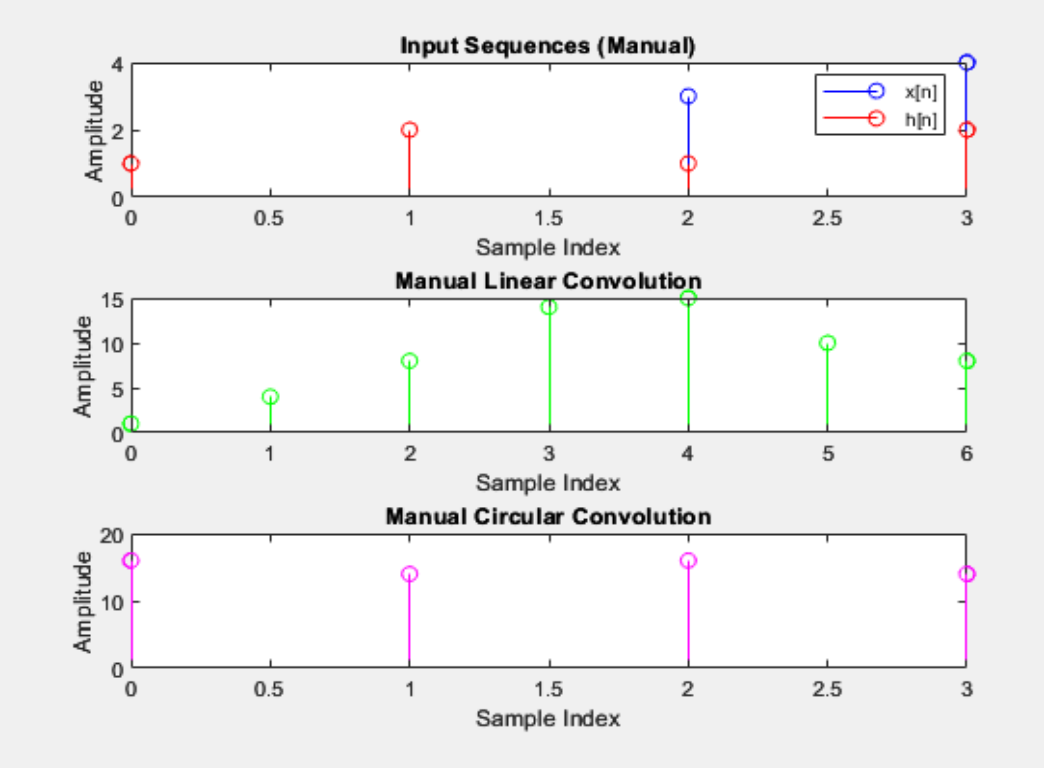

Figure 1. Manual Convolution Results (Linear and Circular)

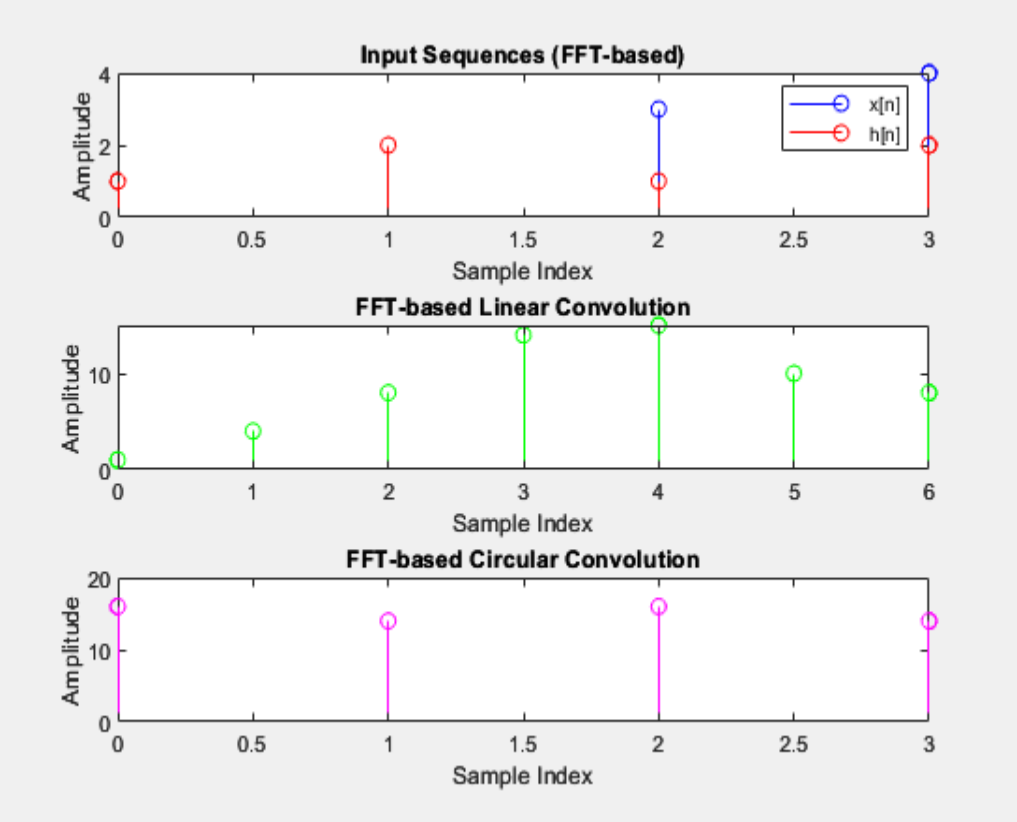

Figure 2. FFT-Based Convolution Results (Linear and Circular).

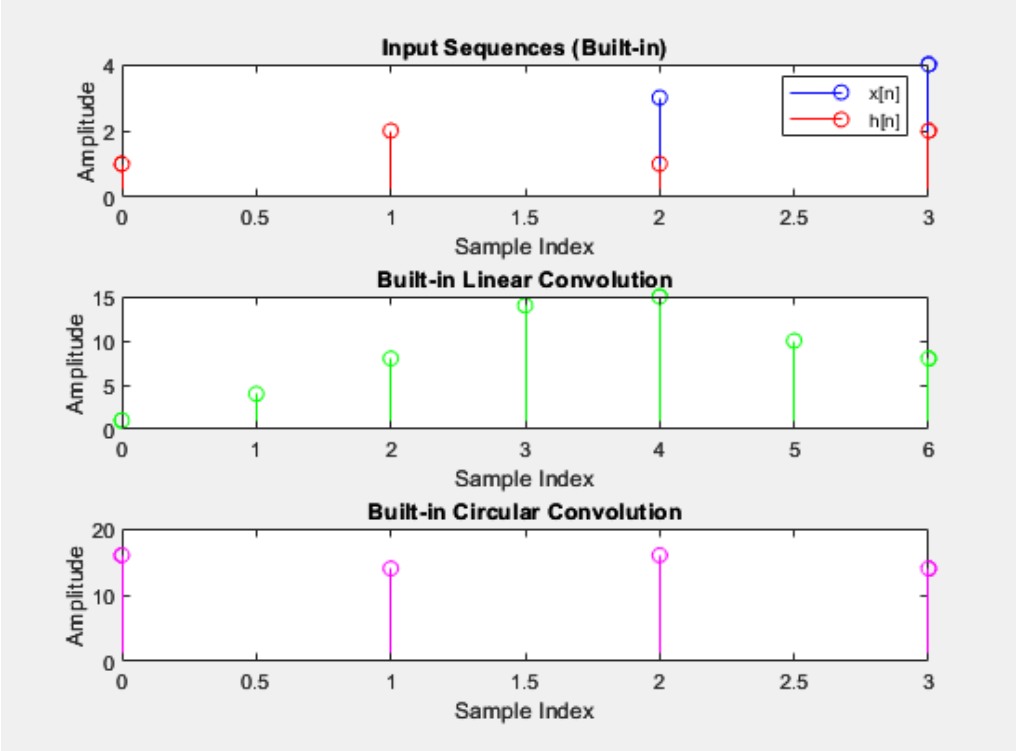

Figure 3. Built-in Function Convolution Results (Linear and Circular).

Exercise:

1. (a) Write a MATLAB program to compute the circular convolution of the sequence's x[n] = [1 0 -1] and h[n] = [2 1] using the following methods. Display the results in the command window.
   (i) Circular Convolution using FFT
   (ii) Built-in Convolution without Zero Padding.
   (b) Plot the results of both methods in a single figure. Use appropriate legends, titles, and axis labels.
   (c) Modify the program to zero-pad the shorter sequence to the length of the longer one before performing the circular convolution. Recompute and display the results.
   (d) Discuss how zero-padding affects the convolution output.

2. (a) Write a MATLAB program to compute the linear convolution of two discrete time sequences x[n] = [1 2 3] and h[n] = [1 1 1 1] using the following methods. Display the results in the command window.
   (i) Manual Convolution without Zero Padding.
   (ii) Manual Convolution with Zero Padding.
   (iii) Built-in Convolution with Zero Padding.
   (b) Plot the results of all three methods (manual convolution without zero-padding, manual convolution with zero-padding, and built-in convolution with zero-padding) in a single figure. Use appropriate legends, titles, and labels for the plots.
   (c) Compare the results of the three methods. Discuss the differences and explain the importance of zero-padding in linear convolution.

3. Convolution is a key operation in signal processing, essential for modifying and analyzing signals. By combining an input signal with a filter, convolution allows for tasks such as noise suppression and feature extraction, enabling the enhancement of signal quality and clarity. This process is particularly effective for removing random noise while retaining important signal characteristics.

   Write a MATLAB program to perform the following tasks.
   (i) Generate a sine wave signal x[n] = sin(2πft) with a frequency of 5 Hz, sampled at a rate of 1000 Hz, and add Gaussian noise with a standard deviation of 0.5 to the signal.
   (ii) Define a simple moving average filter $h_{avg}[n]$ as follows,

$$h_{avg}[n] = [0.25 \quad 0.25 \quad 0.25 \quad 0.25].$$

   (iii) Perform linear convolution of x[n] with $h_{avg}[n]$ using both a manual implementation of convolution and the built-in MATLAB function.

(iv) Plot the original noisy signal and the filtered signal on the same figure with appropriate labels and titles to visualize the effect of filtering.
(v) Explain how the moving average filter reduces noise in the signal.

4. Two-dimensional (2D) convolution is a fundamental technique in image processing, widely used for operations such as blurring, sharpening, and edge detection by applying filters to images. It involves sliding a kernel (filter) over an image matrix and computing weighted sums at each position.

Write a MATLAB program to perform the following tasks.

(i) Import a grayscale image (e.g., "cameraman.tif") into MATLAB.
(ii) Define a 3×3 averaging filter and a 3×3 sharpening filter as follows.

Hint: The averaging (h_{avg}) and sharpening (h_{sharp}) filters can be defined as follows, respectively.

$$h_{avg}[n] = \frac{1}{9}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \qquad h_{sharp}[n] = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$$

(iii) Use the "`conv2`" function to perform 2D convolution on the image with both filters (h_avg and h_sharp). Apply the convolution with the 'same' option to maintain the image dimensions.
(iv) Use the `imshow` function to display the original grayscale image, the filtered image using the averaging filter, and the filtered image using the sharpening filter in a single figure. Ensure each image has an appropriate title.

Hint: Use `uint8()` and the `max` and `min` functions to clip and convert image values.