National Institute of Technology
Calicut

Department of Electronics and Communication Engineering

Digital Signal Processing LAB

Week 3 Report

Date: 07–08–2025
Instructor: Dr. Sakthivel V

Submitted by:
Mummana Jagadeesh B231113EC
Muhammed Nihal MP B230437EC

Aim

To perform and compare:

1. Linear convolution using manual MATLAB code and built-in `conv()` function.
2. Circular convolution using manual MATLAB code and built-in `cconv()` function.
3. Linear convolution using the circular convolution method.

Experiment 3A: Linear Convolution

Theory

Linear convolution is a fundamental operation in discrete-time signal processing that combines two sequences to produce a third sequence representing how the shape of one is modified by the other. For two finite-length sequences $x[n]$ of length N_1 and $h[n]$ of length N_2, the linear convolution is defined as:

$$y[n] = \sum_{k=0}^{N_1-1} x[k] \cdot h[n-k]$$

The resulting sequence $y[n]$ has length:

$$N = N_1 + N_2 - 1$$

Properties of Linear Convolution:

- **Commutative:** $x[n] * h[n] = h[n] * x[n]$
- **Associative:** $(x[n] * h_1[n]) * h_2[n] = x[n] * (h_1[n] * h_2[n])$
- **Distributive:** $x[n] * (h_1[n] + h_2[n]) = (x[n] * h_1[n]) + (x[n] * h_2[n])$
- Produces exact output without time aliasing (unlike circular convolution)

In MATLAB, linear convolution can be computed in two ways:

1. Using the built-in `conv(x,h)` function
2. Implementing manually using nested loops and index checks

MATLAB Implementation

To perform linear convolution in MATLAB:

- Define two sequences x and h
- Use `conv` for built-in computation
- Implement nested loops for manual computation
- Compare results visually using stem plots

MATLAB Code Listing

Code:

MATLAB Code for Linear Convolution

```
x = input('Enter sequence x: ');
x_start = input('Enter starting index of x: ');

h = input('Enter sequence h: ');
h_start = input('Enter starting index of h: ');

y_builtin = conv(x, h);

N1 = length(x);
N2 = length(h);
N = N1 + N2 - 1;

y_manual = zeros(1, N);
for n = 1:N
    for k = 1:N1
        if (n-k+1 >= 1) && (n-k+1 <= N2)
            y_manual(n) = y_manual(n) + x(k) * h(n-k+1);
        end
    end
end

y_start = x_start + h_start;
y_end = y_start + N - 1;
n_y = y_start:y_end;

disp('Indices:');
disp(n_y);

disp('Built-in Conv:');
disp(y_builtin);

disp('Manual Conv:');
disp(y_manual);

figure;

subplot(1, 2, 1);
stem(n_y, y_builtin, 'filled');
title('Built-in conv() Result');
xlabel('n');
ylabel('Amplitude');
grid on;
```

```
subplot(1, 2, 2);
stem(n_y, y_manual, 'filled');
title('Manual Convolution Result');
xlabel('n');
ylabel('Amplitude');
grid on;
```

Explanation of the Code

The code demonstrates **1D convolution** using both MATLAB's built-in function and a manual implementation:

- It prompts the user to enter an input sequence $x[n]$ and an impulse response $h[n]$, along with their starting indices.
- The built-in convolution is computed using MATLAB's `conv()` function, producing y_{builtin}.
- Manual convolution is performed using nested loops: for each output sample n, it sums products of $x[k]$ and $h[n-k+1]$, carefully handling valid index ranges.
- The output sequence indices are calculated as $n_y = x_{\text{start}}+h_{\text{start}} : x_{\text{start}}+h_{\text{start}}+N-1$.
- Both built-in and manual convolution results are displayed and plotted side by side for comparison.

This approach helps to visualize and verify the convolution process while reinforcing the concept of direct summation-based convolution.

Output

Both manual and built-in methods produce identical results, confirming the correctness of the manual implementation. The output sequence length is $N_1 + N_2 - 1 = 5 + 3 - 1 = 7$.

Outputs:

```
>> lconv
Enter sequence x: [-1 0 -3 2 4 1 2 -2 -1]
Enter starting index of x: -2
Enter sequence h: [-3 1 2]
Enter starting index of h: 1
Indices:
    -1     0     1     2     3     4     5     6     7     8     9

Built-in Conv:
     3    -1     7    -9   -16     5     3    10     5    -5    -2

Manual Conv:
     3    -1     7    -9   -16     5     3    10     5    -5    -2

>>
```

Figure 1: Command Window

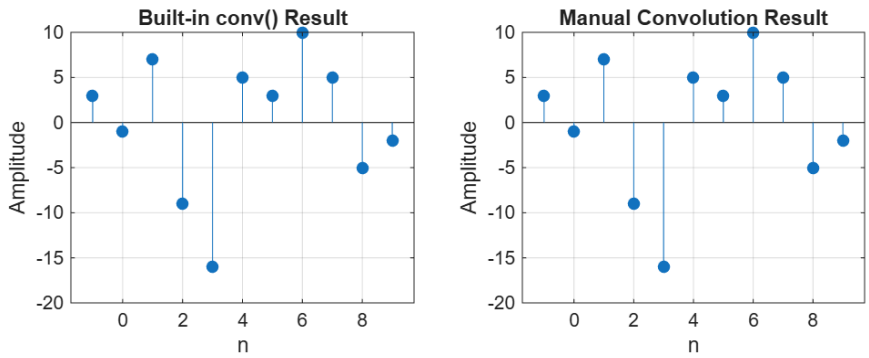

Figure 2: Linear Convolution using Manual vs Builtin

Experiment 3B: Circular Convolution

Theory

Circular convolution is an operation on two finite-length sequences where the result is computed as if the sequences were periodic, with period N. For sequences $x[n]$ and $h[n]$ of equal length N, the circular convolution is defined as:

$$y[n] = \sum_{m=0}^{N-1} x[m] \cdot h[(n-m) \bmod N]$$

for $n = 0, 1, \ldots, N-1$.

Unlike linear convolution, which produces a result of length $N_1 + N_2 - 1$, circular convolution produces a result of the same length as the input sequences. It is closely related to the Discrete Fourier Transform (DFT), as multiplication in the DFT domain corresponds to circular convolution in the time domain.

Properties of Circular Convolution:

- **Commutative:** $x[n]h[n] = h[n]x[n]$
- **Associative:** $(xh_1)h_2 = x(h_1h_2)$
- **Distributive:** $x(h_1 + h_2) = (xh_1) + (xh_2)$
- Can be computed using the inverse DFT of the product of the DFTs of the sequences.

MATLAB Implementation

To perform circular convolution in MATLAB:

- Define two sequences of equal length (zero-pad if necessary)
- Use `cconv(x,h,N)` for the built-in computation
- Implement the manual version using modular indexing for wrapping
- Compare results using stem plots

MATLAB Code Listing

Code:

MATLAB Code for Circular Convolution

```
clx = input('Enter sequence x: ');
x_start = input('Enter starting index of x: ');
h = input('Enter sequence h: ');
h_start = input('Enter starting index of h: ');

N1 = length(x);
N2 = length(h);
N = max(N1, N2);

if N1 < N
    x = [x zeros(1, N - N1)];
end
if N2 < N
    h = [h zeros(1, N - N2)];
end

y_builtin = cconv(x, h, N);

y_manual = zeros(1, N);
for n = 1:N
    for k = 1:N
        t = n - k;
        while t < 0
            t = t + N;
        end
        while t >= N
            t = t - N;
        end
        y_manual(n) = y_manual(n) + x(k) * h(t + 1);
    end
end

n_y = 0:N-1;

disp('Indices:');
disp(n_y);
disp('Built-in Circular Conv:');
disp(y_builtin);
disp('Manual Circular Conv:');
disp(y_manual);

figure;
```

```
subplot(1, 2, 1);
stem(n_y, y_builtin, 'filled');
title('Built-in cconv() Result');
xlabel('n');
ylabel('Amplitude');
grid on;

subplot(1, 2, 2);
stem(n_y, y_manual, 'filled');
title('Manual Circular Convolution Result');
xlabel('n');
ylabel('Amplitude');
grid on;
```

Explanation of the Code

The code demonstrates **circular convolution** using both MATLAB's built-in function and a manual method:

- The user provides an input sequence $x[n]$, an impulse response $h[n]$, and the starting index of x.
- Both sequences are **zero-padded** to have the same length $N = \max(\text{length}(x), \text{length}(h))$ to perform circular convolution correctly.
- Built-in circular convolution is computed using MATLAB's `cconv(x, h, N)` function.
- Manual circular convolution is implemented using nested loops. For each output sample n, the code sums products $x[k] \cdot h[t+1]$, where $t = n - k$ is wrapped around using modulo N logic to handle the circular nature.
- The indices of the output are set as $0 : N - 1$.
- Both built-in and manual results are displayed and plotted side by side to verify correctness.

This approach helps illustrate the difference between linear and circular convolution and shows how the wrapping of indices is handled in circular convolution.

Outputs:

```
>> circonv
Enter sequence x: [-1 0 -3 2 4 1 2 -2 -1]
Enter starting index of x: -2
Enter sequence h: [-3 1 2]
Enter starting index of h: 1
Indices:
     0     1     2     3     4     5     6     7     8

Built-in Circular Conv:
   -2.0000   -3.0000    7.0000   -9.0000  -16.0000    5.0000    3.0000   10.0000    5.0000

Manual Circular Conv:
    -2    -3     7    -9   -16     5     3    10     5
```

Figure 3: Command Window

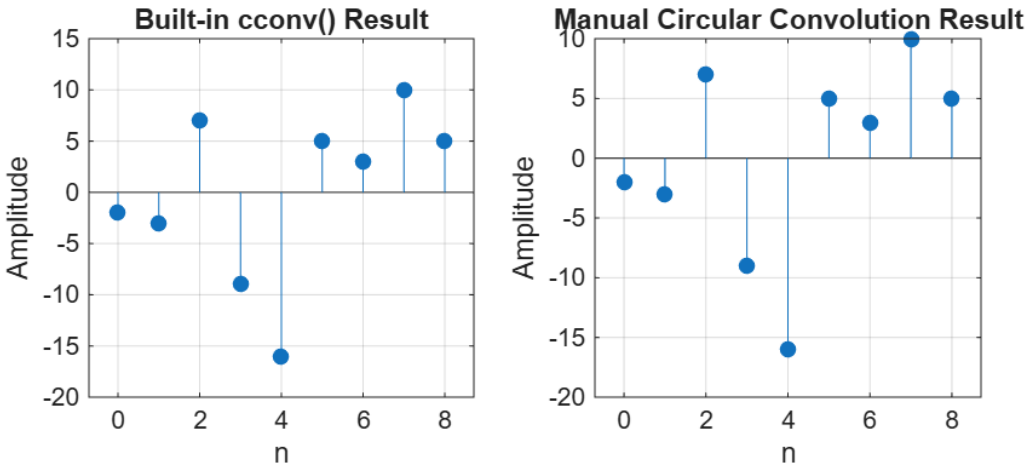

Figure 4: Circular Convolution using Manual vs Builtin

Experiment 3C: Linear Convolution using Circular Convolution

Theory and Mathematical Proof

Linear convolution of two finite-length sequences $x[n]$ and $h[n]$ of lengths L_x and L_h is given by:

$$y_{\text{lin}}[n] = \sum_{k=0}^{L_x-1} x[k] \cdot h[n-k], \quad n = 0, 1, \ldots, L_x + L_h - 2$$

The output length is:

$$N_{\text{lin}} = L_x + L_h - 1$$

Circular convolution of two length-N sequences is defined as:

$$y_{\text{circ}}[n] = \sum_{k=0}^{N-1} x[k] \cdot h[(n-k) \bmod N], \quad n = 0, 1, \ldots, N-1$$

Key Idea: If $N \geq N_{\text{lin}}$, and we zero-pad $x[n]$ and $h[n]$ to length N, then:

$$y_{\text{circ}}[n] = y_{\text{lin}}[n], \quad n = 0, 1, \ldots, N_{\text{lin}} - 1$$

because the modulo indexing in circular convolution will never cause aliasing — the non-overlapping zero-padded portions ensure that wrap-around terms vanish.

Proof: Let $x'[n]$ and $h'[n]$ be zero-padded versions of $x[n]$ and $h[n]$ to length $N \geq L_x + L_h - 1$. In circular convolution:

$$y_{\text{circ}}[n] = \sum_{k=0}^{N-1} x'[k] \cdot h'[(n-k) \bmod N]$$

For $0 \leq n \leq L_x + L_h - 2$, all terms with $(n-k) \bmod N \geq L_h$ or $k \geq L_x$ are zero due to padding. This reduces the sum to:

$$y_{\text{circ}}[n] = \sum_{k=0}^{L_x-1} x[k] \cdot h[n-k]$$

which is exactly the linear convolution definition. Hence, **linear convolution can be obtained from circular convolution via sufficient zero-padding**.

MATLAB Implementation

Steps:

- Compute $N_{\text{lin}} = L_x + L_h - 1$.
- Zero-pad both sequences to length N_{lin}.
- Perform manual circular convolution on the padded sequences.
- Compare with direct manual linear convolution.

MATLAB Code Listing

Code:

MATLAB Code for Linear Convolution using Circular Convolution

```
x = input('Enter sequence x: ');
x_start = input('Enter starting index of x: ');

h = input('Enter sequence h: ');
h_start = input('Enter starting index of h: ');

N1 = length(x);
N2 = length(h);
N_lin = N1 + N2 - 1;

x_pad = [x, zeros(1, N_lin - N1)];
h_pad = [h, zeros(1, N_lin - N2)];

y_circ_for_linear = zeros(1, N_lin);

for n = 1:N_lin
    for m = 1:N_lin
        k = n - m;
        while k < 0
            k = k + N_lin;
        end
        while k >= N_lin
            k = k - N_lin;
        end
        y_circ_for_linear(n) = y_circ_for_linear(n) + x_pad(m) * h_pad(k + 1);
    end
end

y_linear = zeros(1, N_lin);

for n = 1:N_lin
    for k = 1:N1
        if (n - k + 1 >= 1) && (n - k + 1 <= N2)
            y_linear(n) = y_linear(n) + x(k) * h(n - k + 1);
        end
    end
end

n_axis = (x_start + h_start) : (x_start + h_start + N_lin - 1);

disp('Indices:');
disp(n_axis);

disp('Manual Linear Convolution:');
disp(y_linear);
```

```
disp('Linear Convolution via Circular Convolution + Padding:');
disp(y_circ_for_linear);

figure;

subplot(1, 2, 1);
stem(n_axis, y_linear, 'filled', 'LineWidth', 1.5);
title('Manual Linear Convolution');
xlabel('n'); ylabel('Amplitude'); grid on;

subplot(1, 2, 2);
stem(n_axis, y_circ_for_linear, 'filled', 'LineWidth', 1.5);
title('Linear Conv via Circular Conv + Padding');
xlabel('n'); ylabel('Amplitude'); grid on;
```

Explanation of the Code

The code demonstrates how **linear convolution** can be computed manually and via **circular convolution with zero-padding**:

- The user inputs sequences $x[n]$ and $h[n]$ with their starting index for x.
- The sequences are zero-padded to length $N_{\text{lin}} = N_1 + N_2 - 1$, which is the length of their linear convolution.
- **Circular convolution with padding** is performed by treating the padded sequences as circular and summing products with index wrapping (modulo N_{lin}). This produces $y_{\text{circ\_for\_linear}}$, which matches linear convolution due to sufficient padding.
- Manual linear convolution is also implemented directly with nested loops, summing products for valid indices.
- Output indices are computed as $n_{\text{axis}} = x_{\text{start}} + h_{\text{start}} : x_{\text{start}} + h_{\text{start}} + N_{\text{lin}} - 1$.
- Both methods are displayed and plotted side by side to show that linear convolution can be equivalently obtained via circular convolution if the sequences are properly zero-padded.

This illustrates the relationship between linear and circular convolution and how padding allows circular convolution to replicate linear convolution results.

Outputs

```
>> lcconv
Enter sequence x: [-1 0 -3 2 4 1 2 -2 -1]
Enter starting index of x: -2
Enter sequence h: [-3 1 2]
Enter starting index of h: 1
Indices:
    -1     0     1     2     3     4     5     6     7     8     9

Manual Linear Convolution:
     3    -1     7    -9   -16     5     3    10     5    -5    -2

Linear Convolution via Circular Convolution + Padding:
     3    -1     7    -9   -16     5     3    10     5    -5    -2
```

Figure 5: Command Window

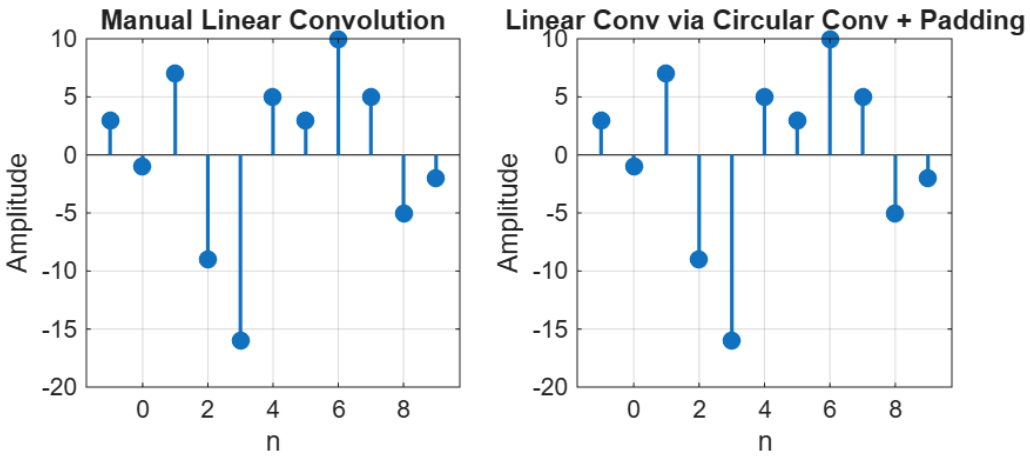

Figure 6: Linear Convolution using Circular Convolution