Cosmos College of Management and Technology

Affiliated to Pokhara University

Saddobato, Lalitpur

Lab Report on: Circular Convolution calculation using the user defined & built in function

Lab report number: 05

Submitted By	**Submitted To**
Name: Aayush Karki	Department of ICT
Roll num: 200101	2080/10/04
Faculty: BE IT	....
Semester: V	
Sub: SSP	
Group: A	

Lab Number: 05

Lab Title: Circular Convolution calculation using the user defined & built in function

Lab Objective

1. Understand the concept of Circular Convolution
2. Implement the user defined methods and login to find the circular convolution
3. Use the built in function to find the Circular Convolution

Theory

Convolution is a mathematical operation that combines two sequences to produce a third sequence, typically representing how one sequence influences the other over time.
Circular Convolution is a specific type of convolution that assumes the sequences are periodic, meaning they repeat indefinitely. Circular convolution is a mathematical operation that combines two periodic sequences to produce a third periodic sequence. Unlike linear convolution, which assumes sequences of finite length and pads them with zeros, circular convolution assumes that the sequences are periodic, meaning they repeat indefinitely.

Circular convolution is often implemented using techniques such as the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) due to their efficiency in computation. These methods allow for faster computation of circular convolution compared to direct summation, especially when dealing with long sequences.

CODE

```
% Circular Convolution: Perform circular convolution of two sequences

% Input Sequences
x1 = input("Enter the first sequence: ");
x2 = input("Enter the second sequence: ");

% Calculate lengths of input sequences
l1 = length(x1);
l2 = length(x2);
```

```
% Determine the length of the result sequence
n1 = max(l1, l2);
l = l1 - l2;

% Zero-pad the shorter sequence to make both sequences of equal length
if l >= 0
        x2 = [x2, zeros(1, l)];
else
        x1 = [x1, zeros(1, -l)];
end

% Initialize the result sequence
y = zeros(1, n1);

% Perform Circular Convolution
for n = 1:n1
        % Initialize the convolution result at index n
        y(n) = 0;

        for i = 1:n1
                % Calculate the circular index j
                j = n - i + 1;

                % Adjust j to handle circular convolution
                if j <= 0
                        j = n1 + j;
                end

                % Accumulate the convolution result
                y(n) = y(n) + x1(i) * x2(j);
        end
end

% Display the Circular Convolution Result
disp("Circular Convolution Result:");
disp(y);

% Plot the Input Sequences and Circular Convolution Result
figure;
subplot(3, 1, 1);
stem(0:n1-1, x1);
```

```
title('Input Sequence 1');
xlabel('n');
ylabel('Amplitude');

subplot(3, 1, 2);
stem(0:n1-1, x2);
title('Input Sequence 2');
xlabel('n');
ylabel('Amplitude');

subplot(3, 1, 3);
stem(0:n1-1, y);
title('Circular Convolution Result');
xlabel('n');
ylabel('Amplitude');
```

Output

```
Enter the first sequence:
[ 1 2 3 4]

Enter the second sequence:
[4 5 6 7 8]

Circular Convolution Result:
65 65 60 50 60
```

%Performing the same circular convolution calculation using the built in function

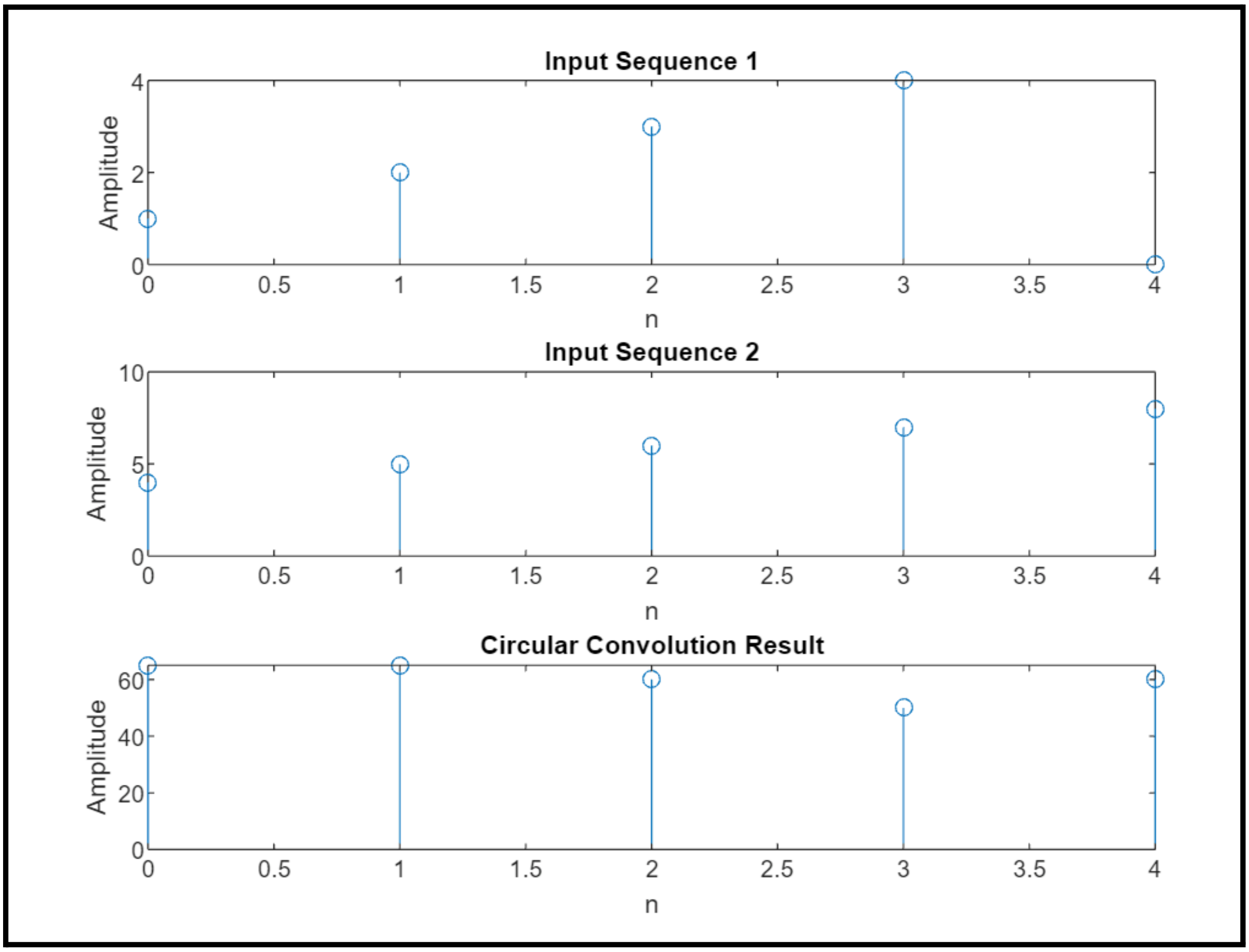
```
% Input arrays from the user
a = input("Enter the first array of integer values: ");
b = input("Enter the second array of integer values: ");

% Perform circular convolution using the inbuilt function 'cconv'
result = cconv(a, b, 5);

% Display the circular convolution result
disp("Circular Convolution Result:");
disp(result);
```

Output

```
Enter the first array of integer values:
  [ 1 2 3 4]
Enter the second array of integer values:
  [4 5 6 7 8]
Circular Convolution Result:
   65    65    60    50    60
```



Discussion

In this lab, we aimed to explore the concept of Circular Convolution and implement it using both user-defined methods and MATLAB's built-in function. The user-defined method involved manually performing the circular convolution by iterating through the sequences and adjusting indices to handle the circular nature of convolution. We observed that the sequences were zero-padded to ensure equal lengths and that the convolution result accurately represented the interaction between the input sequences.

Subsequently, we employed MATLAB's built-in function `cconv` to perform the same circular convolution calculation. This provided a more concise and efficient approach compared to the manual implementation. The built-in function allowed us to achieve the circular convolution result with fewer lines of code, demonstrating the convenience and computational efficiency of using built-in functions for common mathematical operations.

The output from both methods was consistent, confirming the correctness of the circular convolution calculation. The plots displayed the input sequences along with the resulting circular convolution, offering a visual representation of the convolution process.

CONCLUSION

In conclusion, this lab introduced the concept of Circular Convolution and demonstrated its implementation using both user-defined methods and MATLAB's built-in function. We successfully performed circular convolution on two sequences, highlighting the periodic nature of the operation.

The manual implementation involved careful indexing and adjustment to handle circular convolution, while the built-in function provided a more streamlined and efficient solution. This comparison emphasized the advantages of leveraging built-in functions for common mathematical operations, as they offer convenience and optimized performance.

Through this lab, we gained a deeper understanding of circular convolution, its mathematical underpinnings, and the practical aspects of implementing it in MATLAB. The knowledge gained from this lab can be applied in various signal processing and digital signal processing applications where circular convolution is a common operation.