National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering and Computer Science

Faculty Member: Dr. Ahmad Salman Dated: 17/03/2023

Semester: 6th Section: BEE 12C

EE-330 Digital Signal Processing

Lab 8: DFT Properties and Block Convolution Methods

Group Members

		PLO4 - CLO4		PLO5 - CLO5	PLO8 - CLO6	PLO9 - CLO7
Name	Reg. No	Viva / Quiz / Lab Performance	Analysis of data in Lab Report	Modern Tool Usage	Ethics and Safety	Individual and Teamwork
		5 Marks	5 Marks	5 Marks	5 Marks	5 Marks
Danial Ahmad	331388					
Muhammad Umer	345834					
Tariq Umar	334943					

1 Table of Contents

2 DFT Properties and Block Convolution Methods

2.1 Objectives

The objective of this lab is to implement the block convolution methods:

- Overlap and Add
- Overlap Save

2.2 Introduction

Convolution is a fundamental operation in signal processing, used in various applications such as audio and image processing, communication systems, and control systems. Block convolution is a technique used to convolve long sequences of data by dividing them into smaller, manageable blocks. Two commonly used block convolution techniques are overlap and add (OLA) and overlap save (OLS).

2.3 Software

MATLAB is a high-level programming language and numerical computing environment. Developed by MathWorks, it provides an interactive environment for numerical computation, visualization, and programming. MATLAB is widely used in various fields, including engineering, science, and finance, due to its capabilities for matrix and vector operations, implementation of algorithms, and creation of graphical representations of data. The objective of this lab is to provide a hands-on experience with the A-to-D sampling and the D-to-A reconstruction processes that are essential for digital image processing. We will also demonstrate a commonly used method of image zooming (reconstruction) that produces "poor" results, which will help illustrate the importance of understanding the underlying principles of digital image processing.

2.4 Lab Report Instructions

All questions should be answered precisely to get maximum credit. Lab report must ensure following items:

- Lab objectives
- MATLAB codes
- Results (graphs/tables) duly commented and discussed
- Conclusion

3 Lab Procedure

3.1 Lab Task 1

3.1.1 Implementation of Circular Flipping

Write a simple function in MATLAB, with the name *cflip_bee()* that implements circular flipping of an input array *x*.

```
function y = cflip_bee(x, N)
    % This function should take an input array x and circularly flip it
    % where x = input array
    % N = the number of points for circular flipping (DFT points)
    % y = output that should be Modulo N circularly flipped version of x

    % Append zeros to the end of x to make it the same length as N
    x = [x zeros(1, N - length(x))];
    y = zeros(1, N);

    % Flip x and store the modulo N circularly flipped version in y
    for i = 0:(N-1)
        idx = mod(-i, N) + 1;
        y(i+1) = x(idx);
    end

end
                                            Output
>> x = [1 2 3 4 5];

>> cflip_bee(x, 7)
     1     0     0     5     4     3     2
```

3.1.2 Implementation of Circular Shifting

Write a function in MATLAB, with the name *cshift_bee()* that implements circular shifting of an input array *x*.

```
function y = cshift_bee(x, r, N)
    % where x = input array
    % r = amount of shift (in samples), left shift r > 0 and right shift r < 0
    % N = the number of points for circular flipping (DFT points)
    % y = output that should be Modulo N circularly shifted version of x by an amount r

    % Append zeros to the end of x to make it the same length as N
    x = [x zeros(1, N - length(x))];

    % if r < 0, shift right
    if r < 0
        y = [x(N + r + 1:N) x(1:N + r)];
    % if r > 0, shift left
    elseif r > 0
        y = [x(r + 1:N) x(1:r)];

    % if r = 0, no shift
    else
        y = x;
    end

end
                                        Output
>> x = [1 2 3 4 5];

>> cshift_bee(x, 3, 7)
     4     5     0     0     1     2     3
```

3.1.3 Implementation of Circular Convolution

Write a function in MATLAB, with the name *cconv_bee()* that implements circular convolution of an input array *x* with an array *h*.

```
function y = cconv_bee(x, h, N)
    % where x, h = input arrays
    % N = the number of DFT points
    % y = N point output of circular convolution

    % zero pad x and h to length N
    x = [x zeros(1, N - length(x))];
    h = [h zeros(1, N - length(h))];

    y = zeros(1, N);
    for m = 0:(N-1)
        h_fs = cshift_bee(cflip_bee(h, N), m, N);
        for n = 0:(N-1)
            y(m + 1) = y(m + 1) + x(n + 1) * h_fs(n + 1);
        end
    end
end
                                        Output
>> x = [2, 1, 2, 1];
>> h = [1, 2, 3, 4];
>> N = 4;
>> cconv_bee(x, h, N)

result =
    14     16     14     16
```

3.2 Lab Task 2

In this part, we want to implement the block convolution methods that will require use of the *cconv_bee()* function that you created earlier. For implementing the methods, refer to the class lecture slides or your notes. We will now write two functions that take at their input the two sequences *x* and *h*, whose block convolution is to be performed. The functions should take as their input arguments *x*, *h* and *L*, i.e., the block size that needs to be processed.

- Write a code for implementing Overlap and Add Method.

```
function y = oadd_bee(x, h, L)
    % where x, h = input arrays
    % L = block length
    % N = the number of DFT points
    % y = output of the convolution using the overlap and add method

    P = length(h);
    N = L + P - 1;
    x1 = [x, zeros(1, L)];
    y = zeros(1, length(x1) + P - 1);

    for r = 1:ceil(length(x) / L)
        xr = x1((r - 1) * L + 1:r * L);
        yr = cconv_bee(xr, h, N);

        if (r == ceil(length(x) / L) && rem(length(x), L))
            yr = yr(1:N - (L - rem(length(x), L)));
        end

        y = y + [zeros(1, (r - 1) * L), yr, zeros(1, length(y) - ...
            L * (r - 1) - length(yr))];
    end

end
```

- Write a code for implementing Overlap Save Method.

```
function y = osave_bee(x, h, L)
    % where x, h = input arrays
    % L = block length
    % N = the number of DFT points
    % y = output of the convolution using the overlap and add method

    P = length(h);
    N = L + P - 1;
    x1 = [x, zeros(1, L)];
    y = zeros(1, length(x1) + P - 1);
    xr = zeros(1, L);
    for r = 1:ceil((length(x) + P - 1) / (L - P + 1))
        xr = [xr(L - P + 2:L), x1((r - 1) * ...
            (L - P + 1) + 1:(r - 1) * (L - P + 1) + (L - P + 1))];
        yr = cconv_bee(xr, h, N);
        y((r - 1) * (L - P + 1) + 1:(r - 1) * (L - P + 1) + (L - P + 1)) = yr(P:L);
    end

    y = y(1:length(x) + P - 1);

end
```

- Compare the results of both against each other and against the direct convolution of whole length *x*.

```
>> x = [1 2 3 4 5];
>> h = [0.5 0.7 1.2];
>> L = length(h)+1;
>> osave_bee(x, h, L)
    2.6000    1.7000    6.4000    6.5000    6.0000    8.3000         0

>> oadd_bee(x, h, L)
  Columns 1 through 10
    0.5000    4.8000    6.4000    6.5000    6.6000    1.7000         0         0
      0         0
  Column 11
         0

>> conv(x, h)
    0.5000    1.7000    4.1000    6.5000    8.9000    8.3000    6.0000
```

When comparing overlap add and overlap save, both methods have similar computational efficiency and accuracy in producing the output signal. The main difference between the two is that overlap save requires more memory to store the overlapping portion.

When comparing both methods to direct convolution, both overlap add and overlap save are significantly faster and require fewer computational resources, especially when dealing with long signals. However, direct convolution is still the most accurate method and should be used when accuracy is critical, even though it requires more computational resources.

3.3 Lab Task 3

Implement the two block convolution methods but this time using the MATLAB functions *fft()* and *ifft()*.

```
function y = oadd_fun(x, h, L)
    % where x, h = input arrays
    % L = block length
    % y = output of the convolution using the overlap and add method

    P = length(h);
    N = L + P - 1;
    x1 = [x, zeros(1, L)];
    h1 = [h, zeros(1, L - P)];

    H = fft(h1, N);
    y = zeros(1, length(x1) + P - 1);

    for r = 1:ceil(length(x) / L)
        xr = x1((r - 1) * L + 1:r * L);
        X = fft(xr, N);
        Y = X .* H;
        yr = ifft(Y, N);

        if (r == ceil(length(x) / L) && rem(length(x), L))
            yr = yr(1:N - (L - rem(length(x), L)));
        end

        y = y + [zeros(1, (r - 1) * L), yr, zeros(1, length(y) - L * (r - 1) -
length(yr))];
    end

end

function y = osave_fun(x, h, L)
    % where x, h = input arrays
    % L = block length
    % y = output of the convolution using the overlap and add method

    P = length(h);
    N = L + P - 1;
    x1 = [x, zeros(1, L)];
    y = zeros(1, length(x1) + P - 1);
    xr = zeros(1, L);

    H = fft(h, N);

    for r = 1:ceil((length(x) + P - 1) / (L - P + 1))
        xr = [xr(L - P + 2:L), x1((r - 1) * ...
            (L - P + 1) + 1:(r - 1) * (L - P + 1) + (L - P + 1))];
        Xr = fft(xr, N);

        Yr = Xr .* H;
        yr = ifft(Yr);

        y((r - 1) * (L - P + 1) + 1:(r - 1) * (L - P + 1) + (L - P + 1)) = yr(P:L);
    end

    y = y(1:length(x) + P - 1);

end

                                        Output
>> x = [1 2 3 4 5];
>> h = [0.5 0.7 1.2];
>> L = length(h)+1;


>> osave_fun(x, h, L)
    0.5000    1.7000    4.1000    6.5000    8.9000    8.3000    6.0000

>> oadd_fun(x, h, L)
  Columns 1 through 10
    0.5000    1.7000    4.1000    6.5000    8.9000    8.3000    6.0000         0
  0         0

  Column 11
         0
```

4 Conclusion

In conclusion, this lab experiment has provided a comprehensive understanding of the overlap and add (OLA) and overlap save (OLS) block convolution methods and their practical applications. Through the implementation of these methods, we have compared their computational efficiency and accuracy, and explored the effects of varying the block sizes on the computation time and accuracy of the convolved signal. We have learned that OLA and OLS are widely used in signal processing applications where long sequences of data need to be convolved efficiently. The OLA method is particularly useful for real-time processing where the input signal is received continuously, while the OLS method is more suitable for offline processing where the entire input signal is available beforehand.