Batch: B1 Roll No.: 16010121045

Experiment No. 4

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Compute DFT & IDFT of discrete time signals using Matlab.

Objective: To learn & understand the Fourier transform operations on discrete time signals.

Expected Outcome of Experiment:

CO	Outcome
CO3	Analyze signals in frequency domain through various image transforms

Books/ Journals/ Websites referred:

1. http://www.mathworks.com/support/
2. www.math.mtu.edu/~msgocken/intro/intro.html
3. www.mccormick.northwestern.edu/docs/efirst/matlab.pdf
4. A.Nagoor Kani "Digital Signal Processing", 2nd Edition, TMH Education.

Pre Lab/ Prior Concepts:

Implementation details along with screenshots:

Given a sequence of N samples $f(n)$, indexed by $n = 0..N-1$, the Discrete Fourier Transform (DFT) is defined as $F(k)$, where $k=0..N-1$:

$$F(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f(n) e^{-j2\pi kn/N}$$

$F(k)$ are often called the 'Fourier Coefficients' or 'Harmonics'.

The sequence $f(n)$ can be calculated from $F(k)$ using the Inverse Discrete Fourier Transform (IDFT):

$$f(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} F(k) e^{+j2\pi nk/N}$$

In general, both $f(n)$ and $F(k)$ are complex.

Annex A shows that the IDFT defined above really is an *inverse* DFT.

Conventionally, the sequences $f(n)$ and $F(k)$ is referred to as 'time domain' data and 'frequency domain' data respectively. Of course there is no reason why the samples in $f(n)$ need be samples of a time dependent signal. For example, they could be spatial image samples (though in such cases a 2 dimensional set would be more common).

Although we have stated that both n and k range over 0..N-1, the definitions above have a periodicity of N:

$$F(k + N) = F(k) \qquad f(n + N) = f(n)$$

So both $f(n)$ and $F(k)$ are defined for all (integral) n and k respectively, but we only need to calculate values in the range 0..N-1. Any other points can be obtained using the above periodicity property.

For the sake of simplicity, when considering various Fast Fourier Transform (FFT) algorithms, we shall ignore the scaling factors and simply define the FFT and Inverse FFT (IFFT) like this:

$$FFT_N(k,f) = \sum_{n=0}^{N-1} f(n)\, e^{-j2\pi kn/N} = \sqrt{N} F(k)$$

$$IFFT_N(n,F) = \sum_{k=0}^{N-1} F(k)\, e^{+j2\pi nk/N} = \sqrt{N} f(n)$$

In fact, we shall only consider the FFT algorithms in detail. The inverse FFT (IFFT) is easily obtained from the FFT.

Implementation steps with screenshots for DFT.

```
% Define the input signal
x = input('Enter the input signal (in square brackets, e.g., [1 2 3 4 5 6 7 8]): ');

% Zero-pad to the nearest power of 2
N = length(x);
N_fft = 2^nextpow2(N);
x_padded = [x, zeros(1, N_fft - N)];

% Perform bit reversal and display the sequence of bits formed
bit_rev_indices = bitrevorder(0:N_fft-1);
x_bit_rev = x_padded(bit_rev_indices + 1);

% Cast x_bit_rev to complex type
x_bit_rev = complex(x_bit_rev);

disp('Bit-reversed input signal:');
disp(x_bit_rev);

% Calculate twiddle factors
n = 0:N_fft-1;
k = 0:N_fft-1;
WN = exp(-1i*2*pi/N_fft);
W = WN.^(n'*k);

% Calculate DFT using matrix multiplication
X = x_bit_rev * W;

% Print the results
disp(' ');
disp('Twiddle factors:');
disp(W);
disp(' ');
disp('DFT:');
disp(X);
```

```
Enter the input signal (in square brackets, e.g., [1 2 3 4 5 6 7 8]):
[2,1,2,1,1,2,1,2]
Bit-reversed input signal:
     2     1     2     1     1     2     1     2


Twiddle factors:
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
   1.0000 + 0.0000i   0.7071 - 0.7071i   0.0000 - 1.0000i  -0.7071 - 0.7071i  -1.0000 - 0.0000i  -0.7071 + 0.7071i  -0.0000 + 1.0000i   0.7071 + 0.7071i
   1.0000 + 0.0000i   0.0000 - 1.0000i  -1.0000 - 0.0000i  -0.0000 + 1.0000i   1.0000 + 0.0000i   0.0000 - 1.0000i  -1.0000 - 0.0000i  -0.0000 + 1.0000i
   1.0000 + 0.0000i  -0.7071 - 0.7071i  -0.0000 + 1.0000i   0.7071 - 0.7071i  -1.0000 - 0.0000i   0.7071 + 0.7071i   0.0000 - 1.0000i  -0.7071 + 0.7071i
   1.0000 + 0.0000i  -1.0000 - 0.0000i   1.0000 + 0.0000i  -1.0000 - 0.0000i   1.0000 + 0.0000i  -1.0000 - 0.0000i   1.0000 + 0.0000i  -1.0000 - 0.0000i
   1.0000 + 0.0000i  -0.7071 + 0.7071i   0.0000 - 1.0000i   0.7071 + 0.7071i  -1.0000 - 0.0000i   0.7071 - 0.7071i  -0.0000 + 1.0000i  -0.7071 - 0.7071i
   1.0000 + 0.0000i  -0.0000 + 1.0000i  -1.0000 - 0.0000i   0.0000 - 1.0000i   1.0000 + 0.0000i  -0.0000 + 1.0000i  -1.0000 - 0.0000i   0.0000 - 1.0000i
   1.0000 + 0.0000i   0.7071 + 0.7071i  -0.0000 + 1.0000i  -0.7071 + 0.7071i  -1.0000 - 0.0000i  -0.7071 - 0.7071i   0.0000 - 1.0000i   0.7071 - 0.7071i


DFT:
  12.0000 + 0.0000i   1.0000 + 0.4142i  -0.0000 - 0.0000i   1.0000 + 2.4142i   0.0000 - 0.0000i   1.0000 - 2.4142i   0.0000 - 0.0000i   1.0000 - 0.4142i
```

```
2-point DFT:
Pair 1-2: 12+0i  -1.33227e-15-1.33227e-15i
Pair 2-3: 1+0.41421i              1+2.4142i
Pair 3-4: -1.3323e-15-1.3323e-15i          0-6.2172e-15i
Pair 4-5: 1+2.4142i             1-2.4142i
Pair 5-6: 0-6.2172e-15i   3.9968e-15-3.9968e-15i
Pair 6-7: 1-2.4142i             1-0.41421i
Pair 7-0: 3.9968e-15-3.9968e-15i                   12+0i

4-point DFT:
Pairs 1-2, 3-4: 12+0i  -1.33227e-15-1.33227e-15i              1+2.41421i            0-6.21725e-15i
Pairs 2-3, 4-5: 1+0.41421i              1+2.4142i          0-6.2172e-15i              1-2.4142i
Pairs 3-4, 5-6: -1.3323e-15-1.3323e-15i          0-6.2172e-15i              1-2.4142i   3.9968e-15-3.9968e-15i
Pairs 4-5, 6-7: 1+2.4142i             1-2.4142i   3.9968e-15-3.9968e-15i             1-0.41421i
Pairs 5-6, 7-0: 0-6.21725e-15i     3.9968e-15-3.9968e-15i              1-0.414214i                  12+0i
Pairs 6-7, 0-1: 1-2.41421i              1-0.414214i                12+0i              1+0.414214i
Pairs 7-0, 1-2: 3.9968e-15-3.9968e-15i                   12+0i              1+0.414214i  -1.33227e-15-1.33227e-15i

8-point DFT:
  12.0000 + 0.0000i   1.0000 + 0.4142i  -0.0000 - 0.0000i   1.0000 + 2.4142i   0.0000 - 0.0000i   1.0000 - 2.4142i   0.0000 - 0.0000i   1.0000 - 0.4142i
```

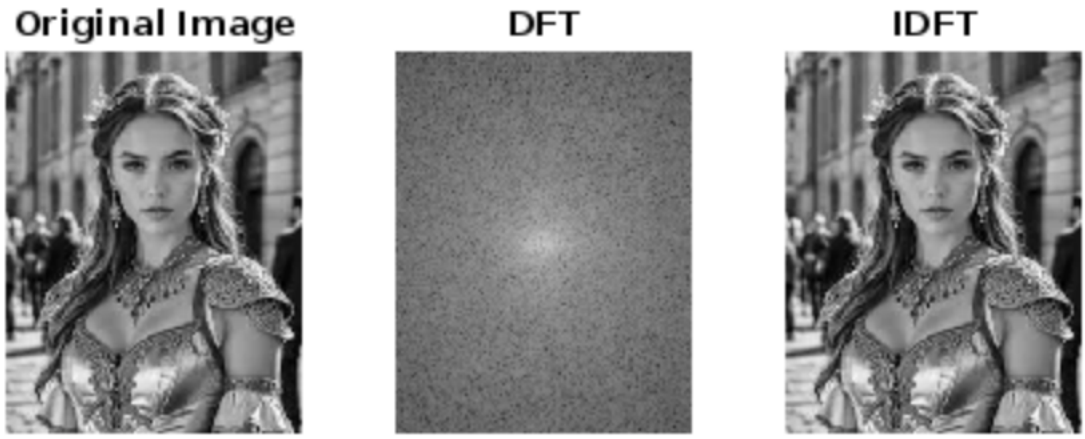
Implementation steps with screenshots for IDFT.

```
function idft_test()
    function x_reconstructed = idft_8pt(X)
        N = length(X);
        n = 0:N-1;
        k = 0:N-1;
        WN_IDFT = exp(1i*2*pi/N);
        W_IDFT = WN_IDFT.^(n'*k);
        x_reconstructed = (X * W_IDFT) / N;
    end

    disp('Enter the type of input:');
    disp('1. Real numbers');
    disp('2. Complex numbers');
    choice = input('Choice: ');

    switch choice
        case 1
            x_real = input('Enter the real part of the input signal (in square brackets,
e.g., [1 2 3 4]): ');
            N_real = length(x_real);
            bit_rev_indices_real = bitrevorder(0:N_real-1);
            x_bit_rev_real = x_real(bit_rev_indices_real + 1);
            x_reconstructed_real = idft_8pt(x_bit_rev_real);
            disp('Reconstructed signal (real part):');
            disp(x_reconstructed_real);

        case 2
            disp('Enter the complex part of the input signal in the format [real1 imag1;
real2 imag2; ...]:');
            x_complex = input('Input: ');
            N_complex = size(x_complex, 1);
            bit_rev_indices_complex = bitrevorder(0:N_complex-1);
            x_bit_rev_complex = (x_complex(bit_rev_indices_complex + 1, 1) + 1i *
x_complex(bit_rev_indices_complex + 1, 2)).';
            x_reconstructed_complex = idft_8pt(x_bit_rev_complex);
            disp('Reconstructed signal (complex part):');
            disp(x_reconstructed_complex);

        otherwise
            disp('Invalid choice');
    end
end
```

```
>> Exp9IDFT
Enter the type of input:
1. Real numbers
2. Complex numbers
Choice:
1
Enter the real part of the input signal (in square brackets, e.g., [1 2 3 4]):
[10,11,21,12]
Reconstructed signal (real part):
  13.5000 + 0.0000i  -0.2500 + 2.2500i  -3.0000 + 0.0000i  -0.2500 - 2.2500i

>> Exp9IDFT
Enter the type of input:
1. Real numbers
2. Complex numbers
Choice:
2
Enter the complex part of the input signal in the format [real1 imag1; real2 imag2; ...]:
Input:
[12 1,23 3,134 54]
Reconstructed signal (complex part):
  12.0000 + 1.0000i
```

On Images:

```
% Read the image
image = imread('images.jpg');

% Convert the image to grayscale if it's RGB
if size(image, 3) == 3
    image = rgb2gray(image);
end

% Display the original image
figure;
subplot(1, 3, 1);
imshow(image);
title('Original Image');

% Compute the DFT of the image using the fft2 function
dft_image = fft2(double(image));

% Shift the zero-frequency component to the center
dft_image_shifted = fftshift(dft_image);

% Display the magnitude spectrum of the DFT
magnitude_spectrum = log(1 + abs(dft_image_shifted));
subplot(1, 3, 2);
imshow(magnitude_spectrum, []);
title('DFT');

% Compute the IDFT of the DFT image using the ifft2 function
reconstructed_image = ifft2(ifftshift(dft_image_shifted));

% Display the reconstructed image
subplot(1, 3, 3);
imshow(uint8(abs(reconstructed_image)));
title('IDFT');
```



Conclusion:-

Hence, we have implemented the concept of Discrete Fourier Transform and Inverse Discrete Fourier Transform in this experiment using MATLAB

Date: _____ **Signature of faculty in-charge**

Post Lab Descriptive Questions

1. **Compare and discuss the computational efficiency of DFT and FFT:** The Discrete Fourier Transform (DFT) computes the Fourier transform of a sequence by directly evaluating a sum of complex exponentials. Its computational complexity is O(N^2), where N is the number of samples. However, the Fast Fourier Transform (FFT) is an algorithm that computes the DFT with a complexity of O(N log N). The FFT exploits the periodicity and symmetry properties of the DFT to reduce the number of computations significantly, making it much more computationally efficient than the DFT for large N.

2. **Give the properties of DFT and IDFT:**

    - DFT (Discrete Fourier Transform):
        - Linearity: DFT is a linear operation.
        - Periodicity: The DFT of a periodic sequence is also periodic.
        - Time Shifting: Shifting a signal in the time domain results in phase modulation in the frequency domain.
        - Frequency Shifting: Shifting a signal in the frequency domain results in modulation in the time domain.
    - IDFT (Inverse Discrete Fourier Transform):

- Linearity: IDFT is also a linear operation.
- Conjugate Symmetry: The input and output sequences of the IDFT are conjugate symmetric.
- Time Reversal: Reversing the time sequence of a signal in the time domain results in complex conjugation in the frequency domain.
- Circular Convolution: Circular convolution in the time domain corresponds to multiplication in the frequency domain.

3. **Discuss the impact on computation time & efficiency when the number of samples N increases:** As the number of samples N increases, the computational time for both DFT and FFT also increases. However, the increase in time for DFT is quadratic (O(N^2)), while for FFT, it grows logarithmically (O(N log N)). Therefore, as N increases, the computational efficiency of FFT becomes significantly better compared to DFT. This is because FFT exploits the periodicity and symmetry properties of the DFT to reduce the number of computations required, leading to faster computation times for larger datasets.

4. **How to compute maximum length N for a circular convolution using DFT and IDFT?** To compute the maximum length N for circular convolution using DFT and IDFT, the length of the input sequences being convolved must be considered. The maximum length N for circular convolution is equal to the sum of the lengths of the input sequences minus one. Mathematically, N = M + L - 1, where M is the length of the first sequence and L is the length of the second sequence. This ensures that the circular convolution does not suffer from aliasing or truncation effects. By computing the DFT of both sequences, performing point-wise multiplication in the frequency domain, and then taking the inverse DFT of the result, circular convolution can be efficiently computed for signals of any length up to N.