Heaven's Light is Our Guide

Department of Electronics & Telecommunication Engineering
Rajshahi University of Engineering & Technology

Laboratory Report on

ETE 3216 (Sessional Based on ETE 3215)

Experiment 2

Experimental Study of FFT (Fast Fourier Transform) & IFFT (Inverse Fast Fourier Transform) using MATLAB

Submitted by:
Bimika Binte Hasan
Roll No. 2104045
Session: 2021-22

Submitted to:
Dr. Md. Kamal Hosain
Professor
Dept. of ETE, RUET

Date of Experiment : 13/10/2025
Date of Submission : 27/10/2025

Report Writing	**(Teacher's Section)**	**Lab Viva**
☐ Excellent		☐ Excellent
☐ Good		☐ Good
☐ Average	_____	☐ Average
☐ Poor	Signature	☐ Poor

Objectives

- To understand the concepts of Fast Fourier Transform (FFT) and Inverse FFT (IFFT), and their duality between time and frequency domains.
- To compute and verify FFT and IFFT of discrete-time signals through manual calculation and MATLAB implementation.

2.1 Theory

The Discrete Fourier Transform (DFT) is an important tool in digital signal processing, used to convert a discrete-time signal from the time domain into the frequency domain. However, the direct computation of DFT requires $O(N^2)$ operations, which becomes computationally expensive for large data sizes.

The Fast Fourier Transform (FFT) is an efficient algorithm that reduces the computational complexity of DFT from $O(N^2)$ to $O(N\log N)$. FFT is widely used in signal processing, communications, image processing, and numerical analysis.

Discrete Fourier Transform (DFT)

For a discrete-time sequence $x(n)$, where $n = 0, 1, \ldots, N-1$, the DFT is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1$$

where $X(k)$ represents the frequency components.

Concept of FFT

FFT is an optimized implementation of DFT that uses a divide-and-conquer approach. It splits the signal into:

- Even-indexed samples
- Odd-indexed samples

Let:

$$x_{\text{even}}(n) = x(2n), \quad x_{\text{odd}}(n) = x(2n+1)$$

Then:

$$X(k) = X_{\text{even}}(k) + W_N^k X_{\text{odd}}(k)$$

$$X(k+N/2) = X_{\text{even}}(k) - W_N^k X_{\text{odd}}(k)$$

where:

$$W_N^k = e^{-j\frac{2\pi}{N}k}$$

is the twiddle factor.

This reduces the DFT size by a factor of 2 at each stage, forming a tree structure, ultimately achieving:

$$O(N \log N)$$

Even-Odd Decomposition

The polynomial representation of a sequence enables recursive splitting:

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$

Evaluating this polynomial at roots of unity helps compute frequency domain values faster.

Inverse FFT (IFFT)

The Inverse DFT (IDFT) converts frequency-domain data back to the time domain:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn}$$

IFFT uses the same FFT algorithm with modifications:

- Replace $e^{-j\frac{2\pi}{N}kn}$ with $e^{j\frac{2\pi}{N}kn}$
- Divide each final output element by N

Applications of FFT & IFFT

- Signal spectral analysis
- Fast convolution and correlation
- Image filtering and compression
- OFDM-based wireless communication
- Audio processing (equalization, echo cancellation)

Key Advantages

- High computational efficiency
- Real-time signal processing capability
- Essential for modern digital systems

2.2 Code

(i) FFT for x(n)={2,4,6,8},

```
x = [2, 4, 6, 8];
N = length(x);
X = zeros(1, N);
x_even = zeros(1, N);
x_odd = zeros(1, N);
for n = 1:N
    x_even(n) = (x(n) + x(mod(n, N) + 1)) / 2;  % Even part
    x_odd(n) = (x(n) - x(mod(n, N) + 1)) / 2;   % Odd part
end
for k = 1:N
    sum = 0;
    for n = 1:N
        sum = sum + x_even(n) * exp(-1j * 2 * pi * (k-1) * (n-1) / N);
    end
    X(k) = sum;
end
X_odd = zeros(1, N);
for k = 1:N
    sum = 0;
    for n = 1:N
        sum = sum + x_odd(n) * exp(-1j * 2 * pi * (k-1) * (n-1) / N);
    end
    X_odd(k) = sum; % Odd part contribution
end
X_combined = X + X_odd;
disp('FFT of the sequence using DFT formula for even part:');
disp(X);
disp('FFT of the sequence using DFT formula for odd part:');
disp(X_odd);
disp('Combined FFT of the sequence:');
disp(X_combined);
figure;
```

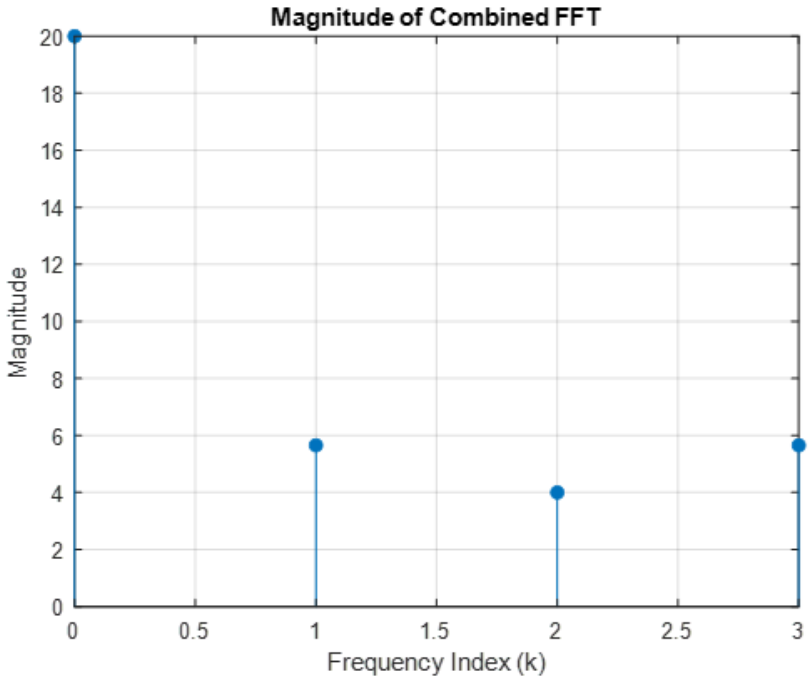
```
stem(0:N-1, abs(X_combined), 'filled');
title('Magnitude of Combined FFT');
xlabel('Frequency Index (k)');
ylabel('Magnitude');
grid on;
```

Output

```
FFT of the sequence using DFT formula for even part:
  20.0000+0.0000i  -4.0000-0.0000i   0.0000-0.0000i
   -4.0000-0.0000i

FFT of the sequence using DFT formula for odd part:
   0.0000+0.0000i  -0.0000+4.0000i  -4.0000-0.0000i
   0.0000-4.0000i

Combined FFT of the sequence:
  20.0000+0.0000i  -4.0000+4.0000i  -4.0000-0.0000i
   -4.0000-4.0000i
```



(ii) IFFT for X(K)={20, -4 + 4i, -4, -4 - 4i},

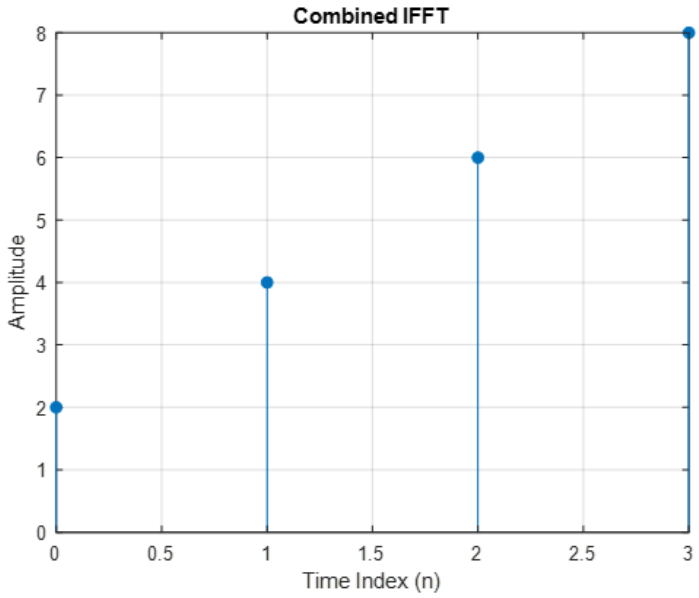
```
x = [20, -4 + 4i, -4, -4 - 4i];
N = length(x);
x_even = zeros(1, N);
x_odd = zeros(1, N);
for n = 1:N
    x_even(n) = (x(n) + x(mod(n, N) + 1)) / 2;  % Even part
    x_odd(n) = (x(n) - x(mod(n, N) + 1)) / 2;   % Odd part
end
x_even_ifft = zeros(1, N);
x_odd_ifft = zeros(1, N);
for n = 1:N
    for k = 1:N
        sum = sum + x_even(k) * exp(1j * 2 * pi * (k-1) * (n-1) / N);
    end
    x_even_ifft(n) = sum / N;
end
for n = 1:N
    for k = 1:N
        sum = sum + x_odd(k) * exp(1j * 2 * pi * (k-1) * (n-1) / N);
    end
    x_odd_ifft(n) = sum / N;
end
x_combined_ifft = x_even_ifft + x_odd_ifft;
disp('IFFT of the sequence (even part):');
disp(x_even_ifft);
disp('IFFT of the sequence (odd part):');
disp(x_odd_ifft);
disp('Combined IFFT of the sequence (even + odd parts):');
disp(x_combined_ifft);
figure;
stem(0:N-1, real(x_combined_ifft), 'filled');
title('Combined IFFT');
xlabel('Time Index (n)');
ylabel('Amplitude');
grid on;
```

Output

```
IFFT of the sequence (even part):
   2.0000+0.0000i   2.0000-2.0000i   0.0000+0.0000i
   4.0000+4.0000i

IFFT of the sequence (odd part):
   0.0000+0.0000i   2.0000+2.0000i   6.0000-0.0000i
   4.0000-4.0000i

Combined IFFT of the sequence (even + odd parts):
   2.0000+0.0000i   4.0000+0.0000i   6.0000-0.0000i
   8.0000-0.0000i
```



Discussion & **Conclusion**

This experiment successfully demonstrated the transformation of a signal between time and frequency domains using FFT and IFFT in MATLAB. The frequency spectrum obtained through FFT provided clear insight into the signal's spectral properties. The IFFT accurately reconstructed the original signal, confirming the reversible nature of Fourier-based processing. The use of MATLAB's built-in functions ensured fast and efficient computation compared to traditional DFT. Overall, this experiment enhanced understanding of spectral analysis and the practical importance of FFT algorithms in digital signal processing.