Lab5 Report (Fast Fourier Transform)

12111008 屠耘诚

***Abstract*—This report offers an in-depth exploration of the Fast Fourier Transform (FFT) algorithm and its application in digital signal processing. FFT, an essential method for efficiently computing the Discrete Fourier Transform (DFT), is pivotal in converting signals from time to frequency domain. The report highlights its suitability for signals of lengths that are powers of 2. Key concepts such as frequency range shifting and zero-padding of DFT are elaborated to deepen the understanding of FFT. The report also addresses the derivation and validation of FFT by comparing it with DTFT across various signals, using strategies like Divide-and-conquer and recursion, and underlining the algorithm's significance and practical applications in signal processing.**

Keywords—Fast Fourier Transform (FFT)， Discrete Fourier Transform (DFT)， Digital Signal Processing， Computational Efficiency， Divide-and-Conquer Strategy， Twiddle Factors， Zero Padding， Frequency Analysis

I. Introduction

The exploration of the Fast Fourier Transform (FFT) in digital signal processing, a topic both intricate and fascinating, is the focus of this report. FFT, more than just an algorithm, is a transformative approach to computing the Discrete Fourier Transform (DFT) efficiently, particularly for signal lengths that are powers of two. This report delves into the essence of FFT, starting with the fundamental techniques of shifting the frequency range and zero padding.

Shifting the frequency range helps us better understand the frequency components of a signal, improving clarity in the frequency domain. Zero padding, while not changing the frequency content, adds more points between existing DFT samples, leading to better approximation to the continuous DTFT

At the core of FFT's computational efficiency lies its divide-and-conquer strategy. By breaking down the DFT computation into smaller segments, FFT dramatically reduces the computational load, particularly for signals with lengths that are powers of two. The recursive nature of the FFT algorithm further enhances this efficiency. By repeatedly applying the divide-and-conquer approach to smaller and smaller DFT segments, FFT simplifies complex computations into manageable tasks.

This report will walk you through these critical aspects of FFT, offering a clear and concise explanation of how it revolutionizes digital signal processing. From the shifting of the frequency range to the intricacies of zero padding, divide-and-conquer strategy, and recursion, each step is carefully analyzed and presented. The journey through FFT in this report is not just educational but also aims to captivate and engage you in the fascinating world of digital signal processing.

II. Continuation of DFT Analysis

A. 2.1 Shifting the Frequency Range

1) Plot of the $|X20[k]|$. Circle the regions of the plot corresponding to low-frequency components.

First, create a Hamming window x of length N = 20, Then compute the 20-point DFT of x. Plot the magnitude of the DFT, $|X20[k]|$, versus the index k(0-19).

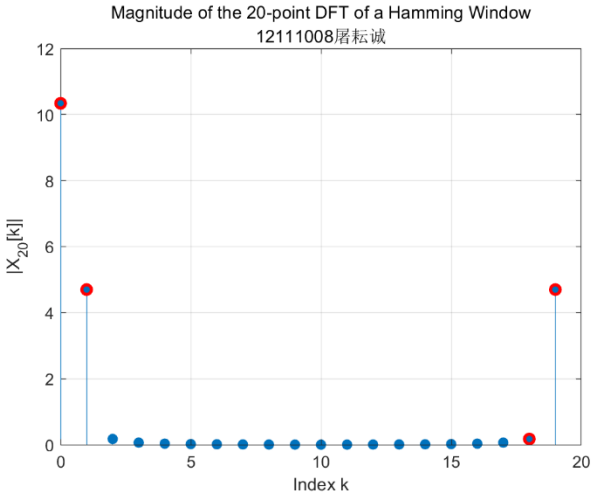

Fig. 1. Plot of magnitude of $|X20[k]|$ versus k

Analysis:

- The plot of | X20[k] | shows the distribution of frequency components in the Hamming window signal.
- The magnitude of the DFT reveals the strength of different frequencies in the signal.
- The arrangement of frequency samples goes against [0, 2π] rather than [$-\pi$, π]. Therefore, low-frequency components are circled as red at the beginning and end of the DFT spectrum.
- Low-frequency components often contain the most energy and represent the overall trend of the signal, while the high-frequency components typically correspond to rapid changes.

2) Plot of the Magnitude of the DTFT samples.

Since |X| is against k rather than ω. And the arrangement of frequency samples goes against [0, 2π] rather than [$-\pi$, π]. We use function DTFTsamples.

code for function DTFTsamples:

```
function [X, w] = DTFTsamples(x)
    N = length(x);
    k = 0:N-1;
    w = 2*pi*k/N;
    w(w >= pi) = w(w >= pi) - 2*pi;
```

```
    w = fftshift(w);
    X = fftshift(DFTsum(x));
end
```

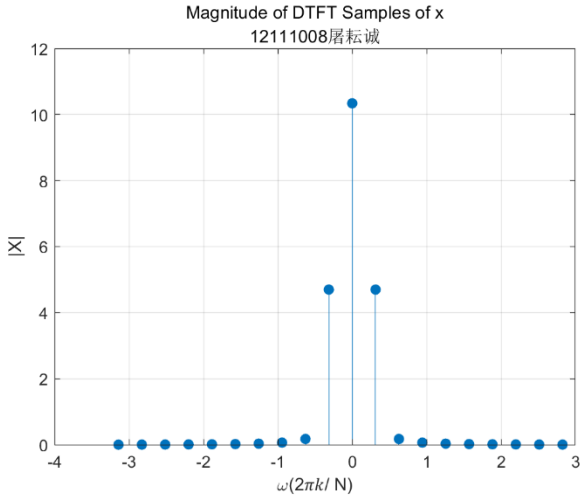

Fig. 2. Magnitude of these DTFT samples versus frequency in rad/sample

- The DTFTsamples function computes the DTFT samples of a given input signal x and returns these samples X along with their corresponding radial frequencies w.
- The function utilizes fftshift to reorder both the DTFT samples and the frequency vector.
- The frequency vector w is adjusted to range from $-\pi$ to π, aligning with the conventional DTFT representation.

B. 2.2 Zero Padding

The finite-duration input signal is defined as

$$x[n] = \begin{cases} \sin(0.1\pi n), & 0 \le n \le 49 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The DTFT samples of this signal were computed using two different DFT sizes: N=50 and N=200. In the case of N=200, zero padding was applied to the signal for n≥50, which means $length(x[n]) = N$.

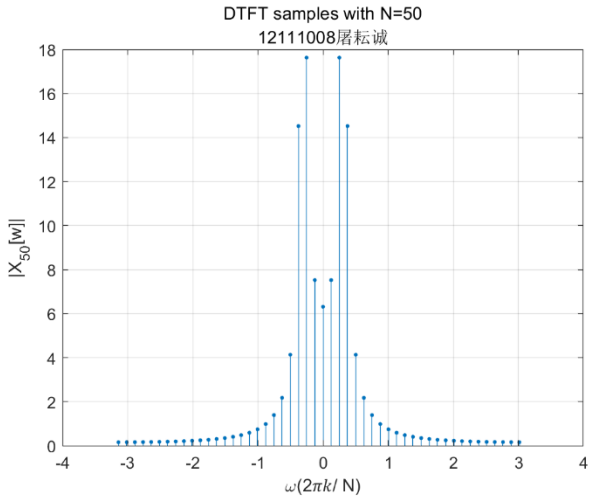

Fig. 3. Plot of DTFTsamples with N = 50 (no zero paddings)

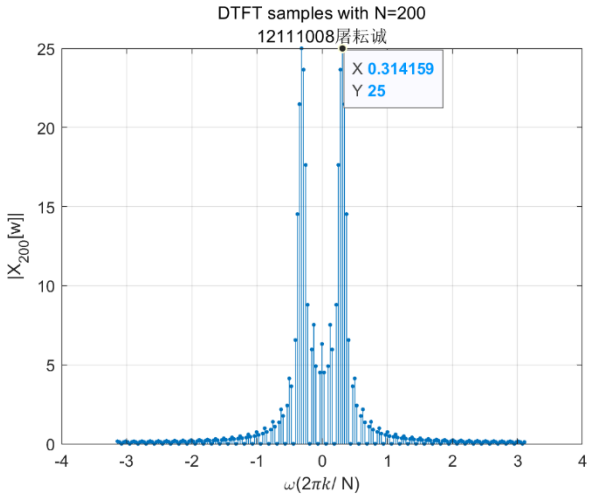

Fig. 4. Plot of DTFTsamples with N = 200 (zero paddings)

Result:

Figure 4 Plot of DTFT samples with N = 200 (zero paddings) looks more like the true DTFT.

When zero padding is applied (for $N = 200$), it doesn't change the frequency content of the signal but interpolates more points in between the existing DFT samples. This leads to a smoother and more continuous-looking spectrum.

Explain why the plots look so different:

1. DTFT and DFT Relationship:

The Discrete-Time Fourier Transform (DTFT) of a signal x[n] is a continuous function of frequency given by:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

The Discrete Fourier Transform (DFT), on the other hand, samples the DTFT at discrete frequencies.

$$X[k] = x[n]e^{-j\frac{2\pi}{N}kn}$$

Where: $X[k]$ is the k-th sample of the DFT.

For a signal of length N, the DFT samples are at frequencies $\omega_k = \frac{2\pi k}{N}$, where k=0, 1, ..., N–1.

2. Sampling Interval in DFT:

The frequency sampling interval in the DFT is determined by $\Delta\omega = \frac{2\pi}{N}$.

- For N=50, the sampling interval is $\frac{2\pi}{50}$.
- For N=200, the sampling interval is $\frac{2\pi}{200}$, which is smaller.

3. Impact of Zero Padding:

Zero padding increases the length of the signal to N by adding zeros, but it doesn't change the signal's frequency content.

What zero padding does is to interpolate additional points between the existing DFT samples, thereby reducing the sampling interval $\Delta\omega$ and providing more samples of the DTFT.

With a finer sampling interval, the DFT (with zero padding) better approximates the continuous DTFT.

We can also show that the DFT (with zero padding) better approximates the continuous DTFT by mathematical formula deduction.

$$X(e^{j\omega}) = j\pi\left(e^{-\frac{j(\omega+0.1\pi)(N-1)}{2}}\frac{\sin\left(\frac{(\omega+0.1\pi)N}{2}\right)}{\sin\left(\frac{(\omega+0.1\pi)}{2}\right)} - e^{-\frac{j(\omega-0.1\pi)(N-1)}{2}}\frac{\sin\left(\frac{(\omega-0.1\pi)N}{2}\right)}{\sin\left(\frac{(\omega-0.1\pi)}{2}\right)}\right)$$

Figure 4 provides a closer waveform to sinc waveforms. And the center of the waveform shown in Figure 4 (0.314159 and -0.314159) just satisfy the previous deduction that two sinc waves centered at -0.1π and 0.1π.

Therefore, the plot for N=200 looks different (and more accurate) compared to N=50 due to the finer frequency resolution, which results from a smaller sampling interval in the frequency domain.

III. The Fast Fourier Transform Algorithm

A straightforward implementation of the DFT is computationally expensive, with the number of multiplications growing as N^2 for an N-point DFT. The FFT addresses this by reducing computation through three concepts:

1. Divide-and-conquer : Split the DFT computation into smaller problems. For an even N, the DFT $X[k]$ is divided into two sums - one for even n and one for odd n:

$$X[k] = \sum_{n=0,n\ even}^{N-1} x[n]e^{-j\frac{2\pi}{N}kn} + \sum_{n=0,n\ odd}^{N-1} x[n]e^{-j\frac{2\pi}{N}kn}$$

The original sequence is divided into even and odd parts, termed as decimation in time. The N-point DFT of x[n] is then expressed as:

$$X[k] = X_0[k] + e^{-j\frac{2\pi}{N}k}X_1[k], k = 0, \dots, N-1$$

$X_0[k]$ and $X_1[k]$ are the $N/2$-point DFTs of even and odd parts, respectively.

2. Recursion: Repetitive application of the same process to solve a problem. After the initial divide-and-conquer step, the FFT algorithm recursively applies this method to smaller DFTs($X[k]$ can be divided into smaller part $X_0[k]$ and $X_1[k]$). This process continues

   (If N/2 is still the multiple of 2, the process can be executed again) until the problem is broken down into its simplest sub-problems, which are then directly solved.

3. Periodicity and Twiddle Factors: The FFT uses the periodic nature of the DFT. This means that we need to only compute $X_0[k]$ and $X_1[k]$ for $N/2$ values of k rather than the N values.

   A more efficient expression for the DFT is derived:

$$X[k] = X_0[k] + W_N^k X1[k]$$

$$X\left[k + \frac{N}{2}\right] = X_0[k] - W_N^k X1[k]$$

where $W_N^k = e^{-j\frac{2\pi}{N}k}$, which are the twiddle factors. This process is shown in figure 5.

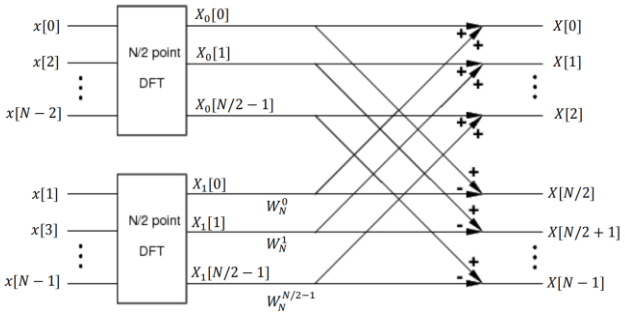

Fig. 5. The N-point DFT is computed using the two $N/2$-point DFT's $X_0^{(N/2)}[k]$ and $X_1^{(N/2)}[k]$

Therefore, The FFT algorithm significantly reduces the number of computations, leading to a complexity of $O(N\ log\ N)$.

A. 3.1 Implementation of Divide-and-Conquer DFT

Continuing from discussion above, We use

$$X[k] = X_0[k] + W_N^k X1[k]$$

$$X\left[k + \frac{N}{2}\right] = X_0[k] - W_N^k X1[k]$$

to implement function dcDFT. Here are the corresponding MATLAB code

```
function X = dcDFT(x)
N = length(x);  % Even number
x0 = x(1:2:N);  % Even part:x[0],x[2],...,x[N-2]
x1 = x(2:2:N);  % Odd part :x[1],x[3],...,x[N-1]
X0 = DFTsum(x0);
X1 = DFTsum(x1);
W=exp(-1j*2*pi*(0:N/2-1)/N);%twiddle factors;
X=zeros(1,N);
X(1:N/2)=X0+W.*X1;
X(N/2+1:N)=X0-W.*X1;
end
```

Testing the Function dcDFT, the result is shown in figure 6:

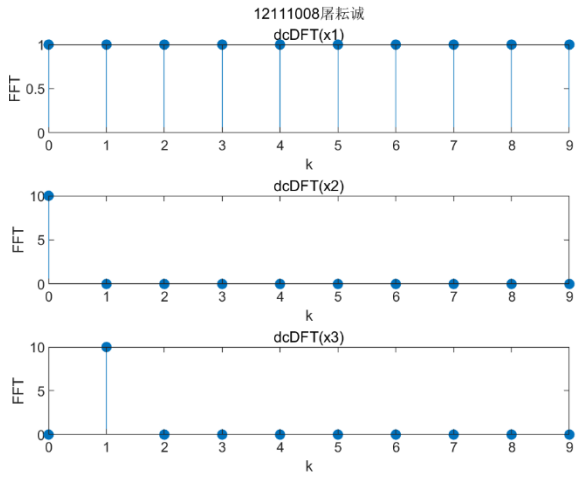

Fig. 6. Magnitude of dcDFT

- Signal 1 ($x[n] = \delta[n]$ for $N = 10$): This is the discrete delta function. The DFT should be a constant sequence.
- Signal 2 ($x[n] = 1$ for $N = 10$): This signal is a constant. Its DFT will have a peak at the zero frequency and zeros elsewhere.
- Signal 3 ($x[n] = e^{j2\pi n/N}$ for $N = 10$): This is a complex exponential signal. The DFT will show a peak at the frequency corresponding to $2\pi/N$, which matches k=1.

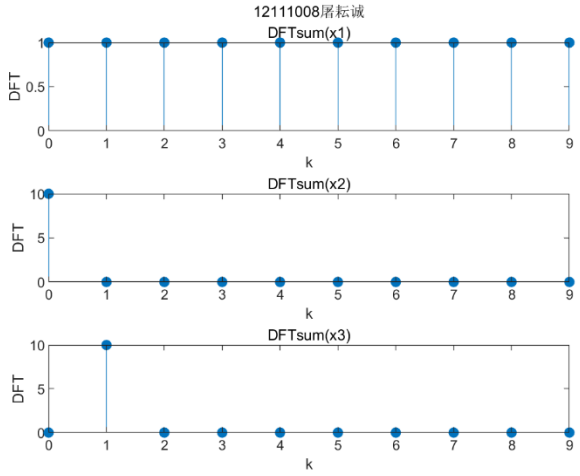

Fig. 7. Magnitude of DFTsum

Meanwhile, we can depict the corresponding DFT results for the three input signals using the DFTsum function, and the output is shown in Figure 7. The results demonstrate complete consistency between Figures 6 and 7, indicating that the DFT results obtained from both methods are identical. Therefore, the testing of the dcDFT function is successful.

Number of Multiplications:

1. DFTsum(x0) and DFTsum(x1) each require $N/2 \times N/2 = N^2/4$ multiplications.
2. The multiplications with the twiddle factors and the subsequent additions are linear operations, so they require N multiplications.

Combining these, the total number of multiplications for dcDFT is approximately

$$2 \times (N^2/4) + N = N^2/2 + N.$$

As for N=10, the result is $\frac{10^2}{2} + 10 = 60$

Special cases 1, -1, *j*, or $-j$:

If we consider a more detailed analysis of the multiplications in the dcDFT function, particularly taking into account the nature of the twiddle factors, we can reduce the count of multiplications. Specifically, we'll consider whether multiplying by 1, -1, *j*, or $-j$ (which are special cases of twiddle factors) should be counted as multiplications.

In many practical implementations, multiplications by 1, -1, *j*, or $-j$ are optimized and not counted as full complex number multiplications because they correspond to simple sign changes or rotations by 90 degrees in the complex plane.

However, the reduction in the number of multiplications is not significant for small values of N. These special cases occur at specific indexes (e.g., $W[0]=1$, $W[N/4]=-j$ for N divisible by 4), but they are a minority of the cases.

Therefore, if we do not consider multiply $W_N^{N/2}$=-1 as one multiplication, the result is $N^2/2 + N/2$. If we completely exclude 1, -1, j, and -j, then we will obtain $\frac{N^2}{2} + \frac{N-4}{2}$

For larger N, especially when N is a power of 2, these special cases might contribute to more substantial savings in the number of multiplications.

B. *3.2 Recursive Divide and Conquer*

1) FFT2, FFT4, FFT8

Here are the MATLAB functions FFT2, FFT4, and FFT8, which implement the Fast Fourier Transform (FFT) for 2, 4, and 8 points respectively. These functions use the divide and conquer strategy, where FFT8 calls FFT4, and FFT4 calls FFT2.

The corresponding flow diagram that results for an 8-point DFT when we decimate 3 times is shown in figure 8.

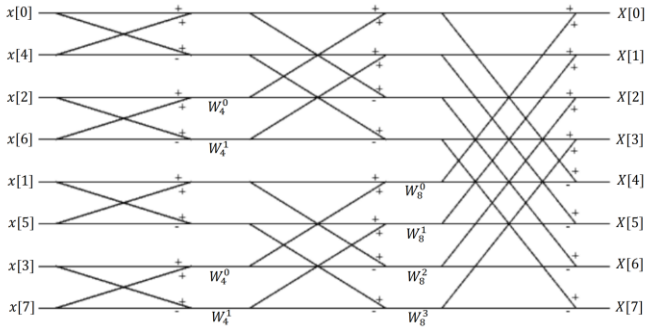

Fig. 8. Flow Diagram of 8-Point FFT

Here are the corresponding MATLAB code:

```
function X = FFT2(x)
X(1)=x(1)+x(2);
X(2)=x(1)-x(2);
end
```

```
function X=FFT4(x)
x0=[x(1) x(3)];%even
x1=[x(2) x(4)];%odd
X0=FFT2(x0);
X1=FFT2(x1);
W=exp(-1j*2*pi.*(0:1)/4);
X=zeros(1,4);
X(1:2)=X0+W.*X1;
X(3:4)=X0-W.*X1;
end
```

```
function X=FFT8(x)
x0=[x(1) x(3) x(5) x(7)];%even
x1=[x(2) x(4) x(6) x(8)];%odd
X0=FFT4(x0);
X1=FFT4(x1);
W=exp(-1j*2*pi*(0:3)/8);
X=zeros(1,8);
X(1:4)=X0+W.*X1;
X(5:8)=X0-W.*X1;
end
```

Now, let's test the FFT8 function with the specified signals and compare the results:

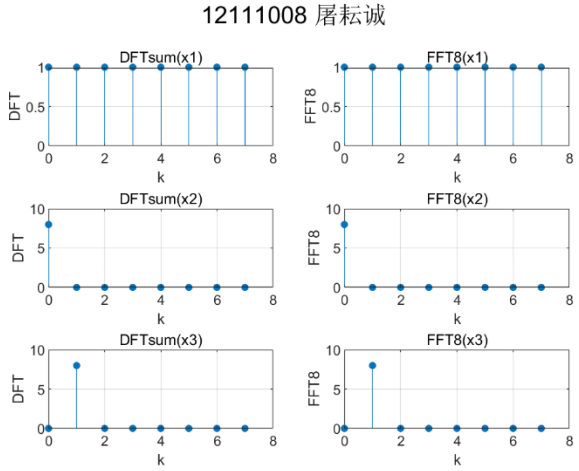

Fig. 9. Magnitude of FFT8 and DFTsum

- Signal 1 ($x[n]=\delta[n]$ for N=8): This is a discrete delta function. The DFT of a delta function is a constant sequence.
- Signal 2 ($x[n]$=1 for N=8): This signal is constant. Its DFT will have a peak at the zero frequency and zeros elsewhere.
- Signal 3 ($x[n]= e^{j2\pi n/N}$ for N=8):This is a complex exponential signal. The DFT will show a peak at the frequency corresponding to $2\pi/8$.

For convenience, we simultaneously processed three corresponding input signals using both the DFTsum function and the FFT8 function, and the output results are plotted together in Figure 8. The results indicate that the DFT results obtained from both the DFTsum and FFT8 methods are identical. Therefore, the testing of the FFT8 function is successful, and the processing results of FFT8 also align with our analysis.

Number of multiplies:

Here we need to determine the total number of multiplies by twiddle factors required for 8-point FFT. There are 3 stages in this case, the number of multiplies in the first stage is $(2^0-1)*(8/2^1)=0$, in the second stage is $(2^1-1)*(8/2^2)=2$ and in the third stage is $(2^2-1)*(8/2^3)=3$. The total number of multiplies by twiddle factors is 0+2+3=**5**. (Alternatively, we can also employ $\frac{8}{2}(\log_2 8-2)+1)=5$

As for N=2^p, without considering $W_N^0=1$ and $W_N^{\frac{N}{2}}=-1$, the total numbers are $\frac{N}{2}\log_2 N$. By excluding trivial complex multiplications with $W_N^0=1$ and $W_N^{\frac{N}{2}}=-1$, the total numbers are

$$\frac{N}{2}\log_2 N-\left(\underbrace{\frac{N}{2}+\frac{N}{4}+\cdots+1}_{m=\log_2 N\ stages}\right)=\frac{N}{2}\log_2 N-\sum_{m=1}^{\log_2 N}(\frac{1}{2})^{m-1}(\frac{N}{2})$$

which is $\mathbf{\frac{N}{2}(\log_2 N-2)+1}$

Since N=2^p, the total number of multiplies is

$$\mathbf{2^{p-1}(p-2)+1}$$

Taken p =10, the total number of multiplies is

$$2^9(10-2)+1=\mathbf{4097}.$$

As for direct implementation, the total number of multiplies is 1024^2=1,048,576, which is two order of magnitude higher than using FFT.

However, if we do not consider $W_N^{N/4}$=–j, since the total number of $W_N^{N/4}$is $N/2-1$, we will have multiplication times $\frac{N}{2}(\log_2 N-3)+2$, which is $2^{p-1}(p-3)+2$

Therefore, the FFT reduces the number of multiplications significantly compared to the direct implementation, illustrating its computational efficiency, especially for large N.

2) fft_stage

It's redundant to write a separate function for each specific length DFT when they each have the same form. Here, we write a recursive method to implement the general case of fft.

```
function X=fft_stage(x)
  N=length(x);
  X=zeros(1,N);
    if N==2
    X(1)=x(1)+x(2);
    X(2)=x(1)-x(2);
    else
        x0=x(1:2:N);
        x1=x(2:2:N);
        X0=fft_stage(x0);
        X1=fft_stage(x1);
        W=exp(-1j*2*pi.*(0:N/2-1)/N);
        X(1:N/2)=X0+W.*X1;
        X(N/2+1:N)=X0-W.*X1;
    end
end
```

Now we will test fft_stage on the three 8-pt signals given above, and verify that it returns the same results as FFT8.

We test for input signal x[n]=δ[n], x[n]=1, x[n]=e^(j2πn/N) for N=8, as they are shown in figure 10.

12111008 屠耘诚

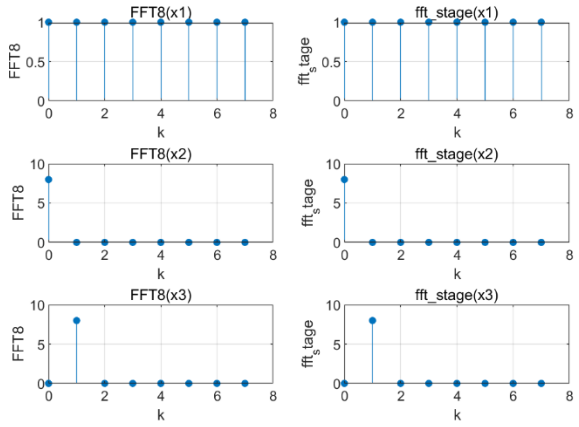

Fig. 10. Magnitude of FFT8 and fft_stage

The results indicate that the DFT results obtained from both the fft_stage and FFT8 methods are identical. Therefore, the testing of the fft_stage function is successful, and the processing results of fft_stage also align with our analysis.

IV. Conclusion

This report has thoroughly examined the Fast Fourier Transform (FFT) algorithm and its significant role in digital signal processing, particularly in efficiently computing the Discrete Fourier Transform (DFT). The report first delved into two crucial aspects of DFT analysis: shifting the frequency range and zero padding. It was observed that shifting the frequency range allows for a better understanding of the distribution of frequency components in a signal, while zero padding, though not altering the frequency content, aids in interpolating more points between existing DFT samples. With a finer sampling interval, the DFT (with zero padding) better approximates the continuous DTFT.

Furthermore, the report explored the FFT algorithm in detail, emphasizing its divide-and-conquer strategy, recursion, and the use of periodicity and twiddle factors. The divide-and-conquer approach, implemented through recursive algorithms, significantly enhances computational efficiency by breaking down the DFT computation into smaller, manageable problems. This methodology not only simplifies the calculations but also drastically reduces the computational complexity, especially for signals of lengths that are powers of 2.

The practical application of these concepts was demonstrated through MATLAB implementations and testing of various signals. These implementations showcased the effectiveness of the FFT in reducing the number of computations required, compared to a direct DFT implementation, thereby underscoring its importance in digital signal processing applications.