Lab5 Fast Fourier Transform

张旭东 12011923

1. Introduction

This lab can be divided into two parts: continuation of DFT analysis and the introduction of FFT algorithm. The first part is mainly to rotate the x label, k to the frequency ω in a more familiar range, $[-\pi, \pi]$ and analyze the level of similarity between DFT and $DTFT$ with the number of points in the DFT. The second part is mainly to introduce two important concepts-divide and conquer and recursion and analysis time complexity of $Fast\ Fourier\ Transform(FFT)$.

2. Continuation of DFT Analysis

From the previous laboratory, the expression of DFT and $inverse\ DFT$ are:

$$\begin{aligned} (DFT)X_N[k] &= \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \\ (inverseDFT)x[n] &= \frac{1}{N}\sum_{k=0}^{N-1} X_N[k]e^{j2\pi kn/N} \end{aligned} \tag{1}$$

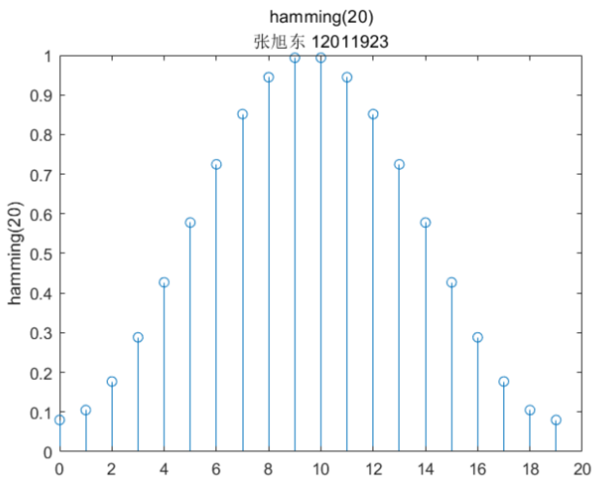
The MATLAB function of DFT is as following:

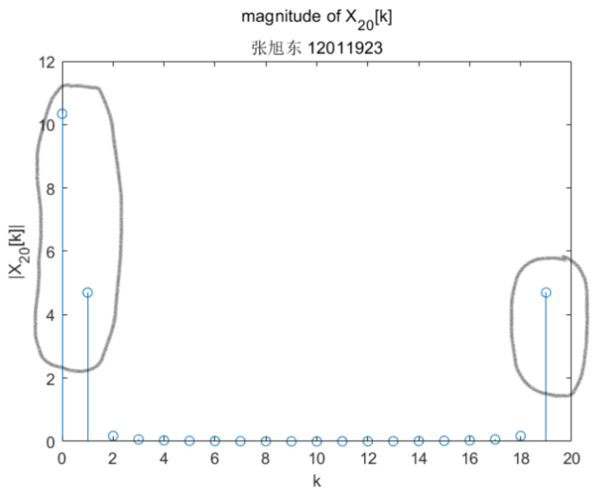
```
function X=DFTsum(x)
    N=length(x);
    X=zeros(1,N);
    for k =1:N
        for n=1:N
            X(k)=X(k)+x(n)*exp(-1i*2*pi*(k-1)*(n-1)/N);
        end
    end
end
```

There is one detail needed to be considered. The index of vector in MATLAB starts from 1 while the index of vector in formula 1 starts from 0. So, the exponential term in code should be $e^{-j2\pi(k-1)(n-1)/N}$.

2.1 Shifting the Frequency Range

In this section, a representation for the DFT of formula 1 that is a bit more intuitive will be illustrated. Starting from the 20 point DFT of Hamming window x.





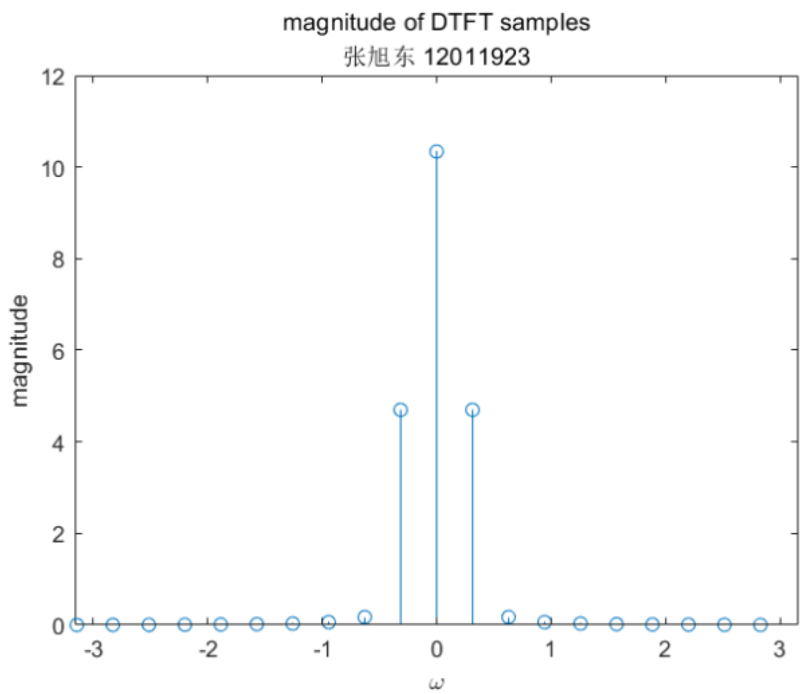
The circled region is the low frequency components. There are two obvious disadvantages in the plot of DFT. First, the DFT values are plotted against k rather than the frequency ω. Second, the arrangement of frequency samples in the DFT goes from 0 to 2π rather than from $-\pi$ to π, as conventional with the $DTFT$. In order to plot the DFT values similar to a conventional $DTFT$ plot, the label should change from k to ω. Each element of ω should be the frequency of the corresponding DFT sample $X(k)$, whose expression is:

$$\omega = \frac{2\pi k}{N} \qquad k \in [0, \ldots, N-1] \tag{2}$$

The x label change from k to frequency ω and some MATLAB command is used to let ω lie in the range from $-\pi$ to π. However, the resulting vectors X and ω are correct, but out of order. Fortunately, MATLAB provides a function, called $fftshift$, to swap the first and second halves of the vectors. A MATLAB function named $DTFTsamples$ is written to compute samples of the $DTFT$ and their corresponding frequencies in the range $-\pi$ and π. The code is shown below:

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

where X is the length N vector of $DTFT$ samples, x is an point vector and ω is the length of N vector of corresponding radial frequencies. The plot of magnitude of the $DTFT$ samples of the Hamming windows of length $N = 20$ used the $DTFTsamples$ is shown below:



The region of the low frequency component is shifted to the middle of the picture, which is in line with expectations.

2.2 Zero Padding

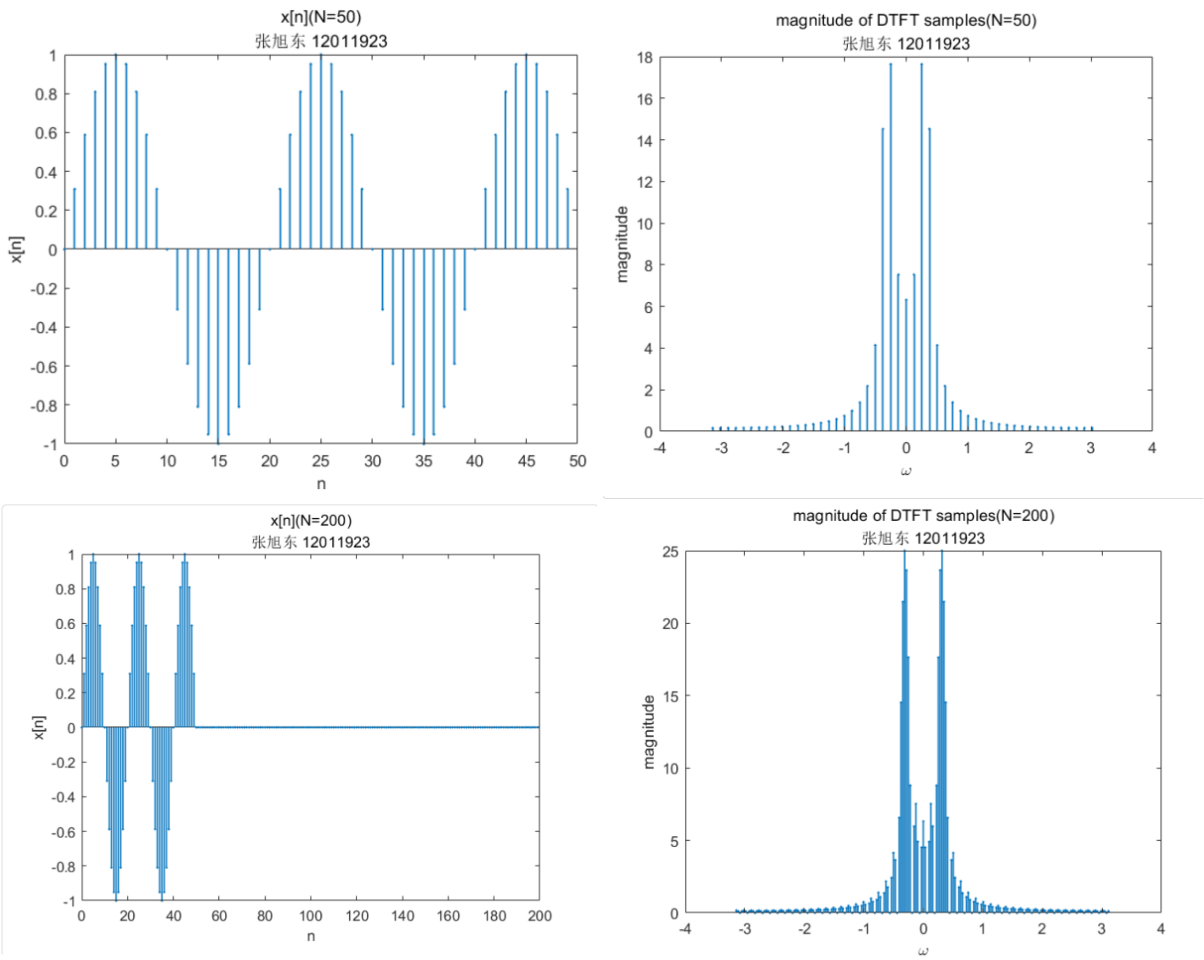
The spacing between samples of the $DTFT$ is determined by the number of points in the DFT .This is due to the conclusion that:

$$X(e^{j\omega}) = X[k] \qquad k \in [0, \ldots, N-1] \\ \omega = \frac{2\pi k}{N} \tag{3}$$

The space between samples is $\frac{2\pi}{N}$, which is determined by N. This can lead to surprising results when N is too small. A further explanation is given through the example below to illustrate this effect.

$$x[n] = \begin{cases} \sin(0.1\pi n) & 0 \leqslant n \leqslant 49 \\ 0 & otherwise \end{cases} \tag{4}$$

In the following, the $DTFT$ samples of $x(n)$ will be computed using both $N = 50$ and $N = 200$ points FT's. When $N \geqslant 50$, $x[N]$ will be zeros because $x[n] = 0$ for $n \geqslant 50$, which is called "zero padding" and has a good effect on producing a finer sampling of the $DTFT$.



From the above pictures, what can be found is that the plot with $N = 200$ looks more like the true $DTFT$. The difference between two plots is due to the length of the DFT. DFT is gotten by sampling the $DTFT$ in the range $[-\pi,\pi]$. With the number sampled increasing, the similarity between DFT and $DTFT$ becomes larger and larger. So DFT with larger length looks more like the true $DTFT$.

3. The Fast Fourier Transform Algorithm

According to formula 1, a straightforward implementation of the DFT can be computationally expensive because the number of multiplies grows as the square of the input length. To reduce this computation, an algorithm that could be used to compute the DFT much more efficiently ,which is known as the Fast Fourier Transform(FFT) was published in 1965. Two important concepts are used in it, which are divide-and-conquer and recursion. The former splits the problem into two smaller problems and the latter applies this divide-and -conquer method repeatedly until the problem is solved.

Another expression form is used to represent the N point DFT of the signal $x[n]$:

$$X[k] = DFT_N[x[n]] \tag{5}$$

If N is even, the sum in (5) can be broken into two parts, one containing all the terms for which n is even, and one containing all the terms for which n is odd:

$$X[k] = \sum_{n=0,\ even}^{N-1} x[n]e^{-j2\pi kn/N} + \sum_{n=0,\ odd}^{N-1} x[n]e^{-j2\pi kn/N} \tag{6}$$

In the first sum, n is replaced by m. In the second sum, n is replaced by $2m+1$. Then, formula 12 can be written as:

$$\begin{aligned} X[k] &= \sum_{m=0}^{\frac{N}{2}-1} x[2m]e^{-j2\pi k2m/N} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1]e^{-j2\pi k(2m+1)/N} \\ &= \sum_{m=0}^{\frac{N}{2}-1} x[2m]e^{-j2\pi km/(N/2)} + e^{-j2\pi k/N} \sum_{m=0}^{\frac{N}{2}-1} x[2m+1]e^{-j2\pi km/(N/2)} \end{aligned}$$

What id obvious is that the expression has the similar form of the definition for the DFT. The first term is an $\frac{N}{2}$ point DFT of the even-numbered data points in the original sequence and the second term is an $\frac{N}{2}$ point DFT of the odd-numbered data points in the original sequence. The N point of DFT of the complete sequence is the sum of $\frac{N}{2}$ point DFT of the even-numbered data points in the original sequence and $e^{-j2\pi k/N}$ times of $\frac{N}{2}$ point DFT of the odd-numbered data points in the original sequence.

To let the expression more concise, two new $\frac{N}{2}$ point data sequences $x_0[n]$ and $x_1[n]$ are defined. The former contains the even-numbered data points in the original sequence and the latter contains the odd-numbered data points in the original sequence. So the expression can be written as:

$$\begin{aligned} x_0[n] &= x[2n] \\ x_1[n] &= x[2n+1] \qquad n = 0, \ldots, \frac{N}{2}-1 \end{aligned} \tag{7}$$

$$X[k] = X_0[k] + e^{-j2\pi k/N} X_1[k] \qquad k = 0, \dots, N-1$$
$$X_0[k] = DFT_{N/2}[x_0[n]] \qquad X_1[k] = DFT_{N/2}[x_1[n]]$$

If N is even, the sum in (5) can be broken into two parts, one containing all the terms for which n is even, and one containing all the terms for which n is odd:

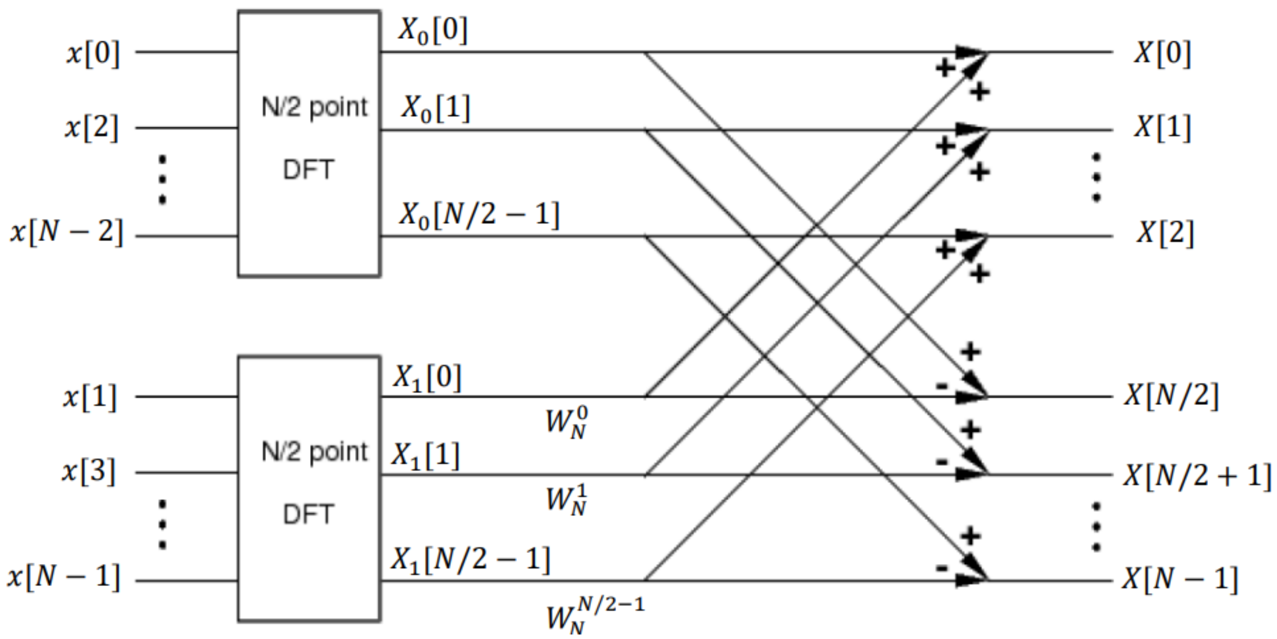
$$-e^{-j2\pi\frac{k}{N}} = e^{-j2\pi\frac{k+N/2}{N}} \tag{8}$$

The expression can be further written as:

$$\begin{aligned} W_N^k &= e^{-j2\pi k/N} \\ X[k] &= X_0[k] + W_N^k X_1[k] \\ X[k+N/2] &= X_0[k] - W_N^k X_1[k] \qquad k = 0, \dots, N/2-1 \end{aligned} \tag{9}$$

The complex constants defined by $W_N^k = e^{-j2\pi k/N}$ are commonly known as the twiddle.

The following pictures shows the operation for using the two $N/2$-point DFT to compute the N-point DFT.



3.1 Implementation of Divide-and-Conquer DFT

In this section, a function named $dcDFT$ will be written to implement the DFT transformation using formula 9. The function is shown as below:

```
function X=dcDFT(x)
            %index start form 1 in matlab while index start from 0
in formula
```

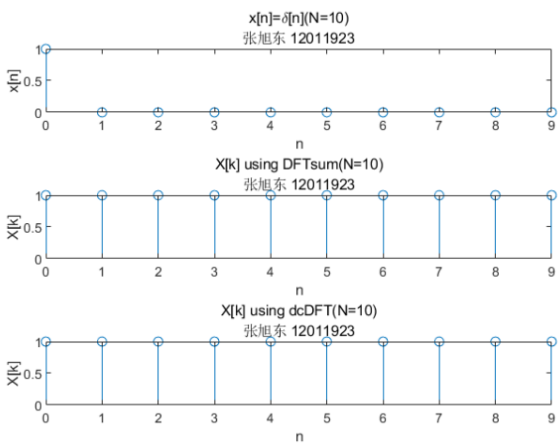
```
    x0=x(1:2:length(x));%even part;
    x1=x(2:2:length(x));%odd part;
    X0=DFTsum(x0);
    X1=DFTsum(x1);
    X=zeros(1,length(x));
    for k=1:length(x)/2
        X(k)=X0(k)+exp(-1j*2*pi/length(x)*(k-1))*X1(k);
    end
    for k=1:length(x)/2
        X(k+length(x)/2)=X0(k)-exp(-1j*2*pi/length(x)*(k-1))*X1(k);
    end
end
```

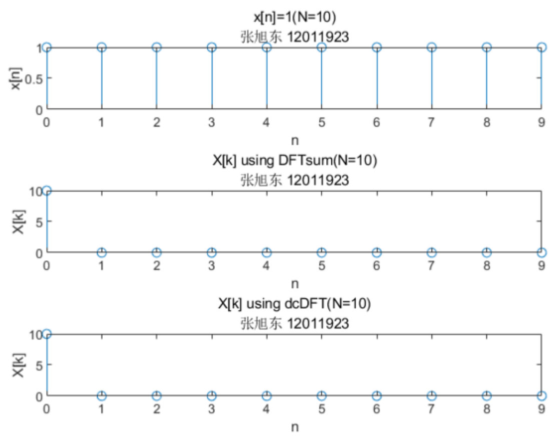
Where x is a vector of even length N, and X is its DFT. The function Separates the samples of x into even and odd points and use function $DFTsum$ to compute the two $N/2$ point $DFT's$. Finally, it multiply the twiddle factors to the second part and combine the two $DFT's$ to form X.

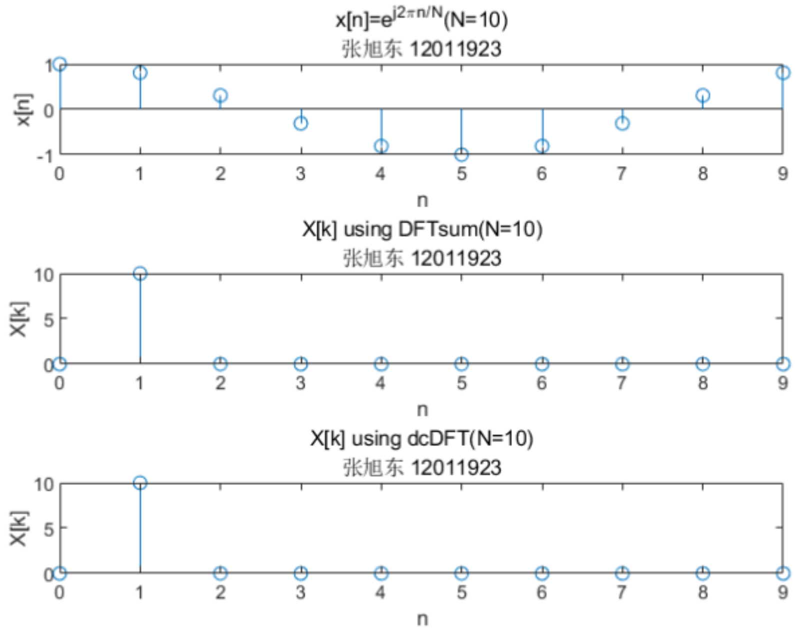
The correctness of function $dcDFT$ is tested using the following signals:

$$\begin{array}{lll} 1. & x[n]=\delta[n], & N=10 \\ 2. & x[n]=1 & N=10 \\ 3. & x[n]=e^{j2\pi n/N} & N=10 \end{array} \tag{10}$$

The result of test is shown below:







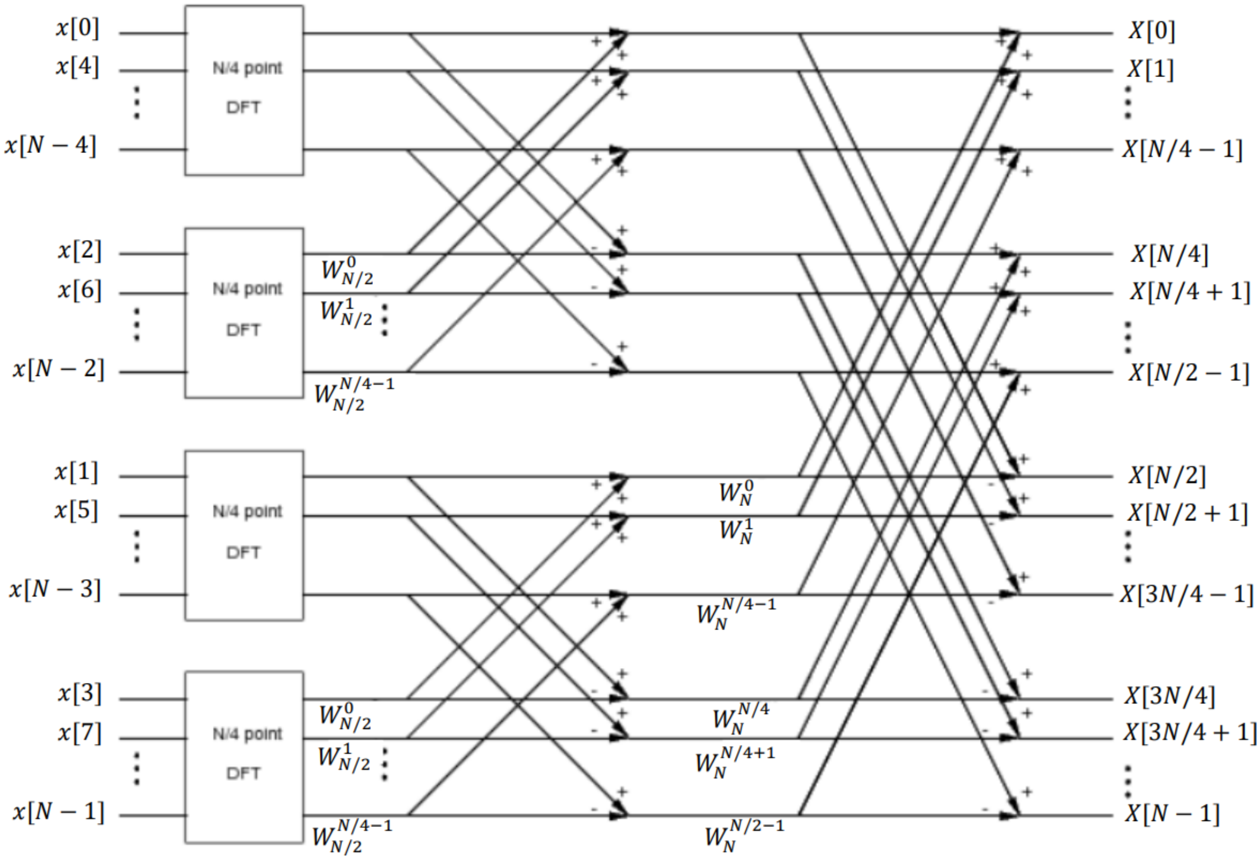
Comparing the plot using function $dcDFT$ to the plot using function $DFTsum$, it is can be found that the result generated by function $dcDFT$ is the same as that generated by function $DFTsum$, which means the function $dcDFT$ is correct.

According to formula 1, there are N^2 multiplies required to compute N point DFT. In the same way, there are $(\frac{N}{2})^2$ multiplies required to compute $\frac{N}{2}$ point DFT. So, there are $(\frac{N}{2})^2$ multiplies required to compute $X_0[k]$ and $X_1[k]$. What's more, there are $\frac{N}{2}$ multiplies required to compute $W_N^k X_1[k]$. At the last, the number of the multiplies required for this approach is:

$$(\frac{N}{2})^2 + (\frac{N}{2})^2 + \frac{N}{2} = \frac{N^2 + N}{2} \tag{11}$$

3.2 Recursive Divide-and-Conquer

In this section, the second basic concept underlying the FFT algorithm will be discussed. Suppose $N/2$ is also even, which means the same strategy in 3.1 can be applied to generate the following picture:

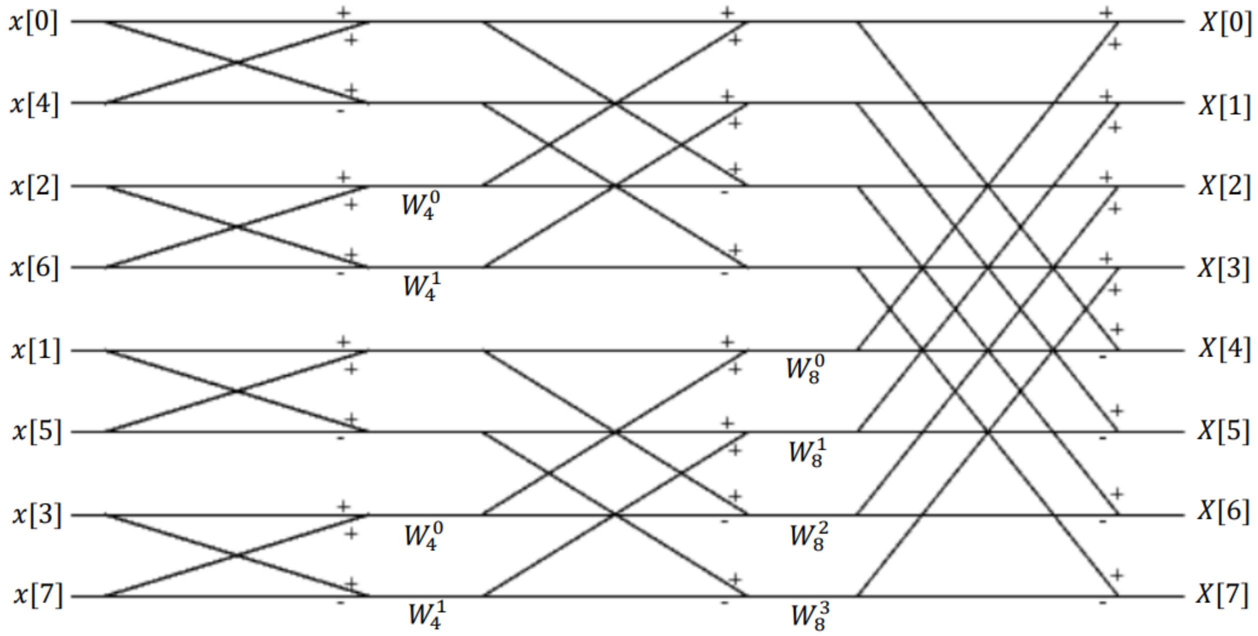


A problem coming from this strategy is that how many times can the strategy be repeated. Suppose N is a power of 2.($N = 2^p$,p is an integer). It is easy to know the value of times which is p until each subsequence contains only two points.

According to formula 1, the DFT for two points is:

$$\begin{aligned} X[0] &= x[0] + x[1] \\ X[1] &= x[0] - x[1] \end{aligned} \tag{12}$$

For $N = 2$, the twiddle factors W_N^k is simplified to 1. The following figure is for 8-point FFT:



Three functions named $FFT2$, $FFT(4)$, $FFT8$ are written to achieve the 2,4,8 point DFT. The function is shown as below:

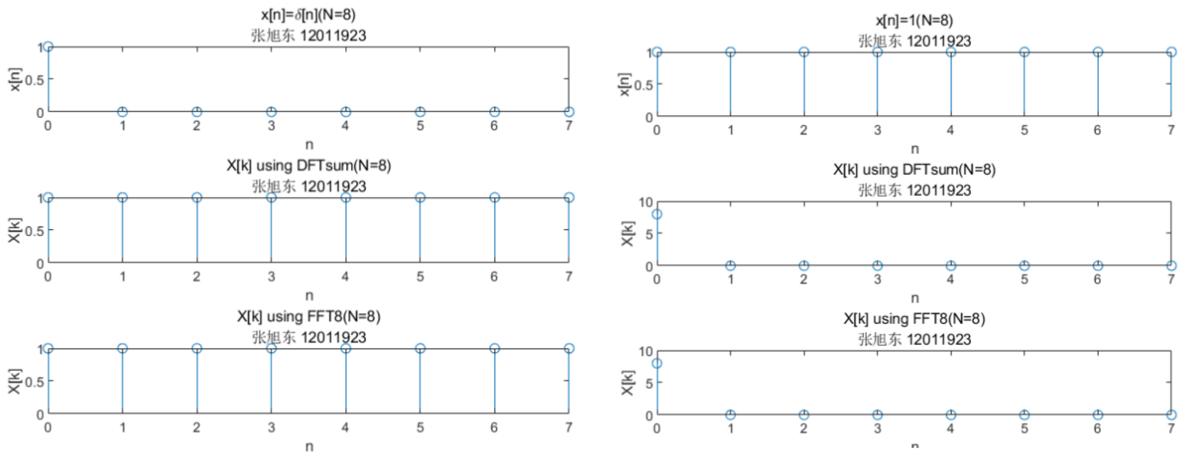
```
function X=FFT2(x)
            %x的长度为2
            X=zeros(1,2);
            %注意index
            X(1)=x(1)+x(2);
            X(2)=x(1)-x(2);
end

function X=FFT4(x)
            %x的长度为4
            X=zeros(1,length(x));
            x0=x(1:2:length(x));
            x1=x(2:2:length(x));
            X0=FFT2(x0);
            X1=FFT2(x1);
            for k=1:length(x)/2
                X(k)=X0(k)+exp(-1j*2*pi/length(x)*(k-1))*X1(k);
            end
            for k=1:length(x)/2
                X(k+length(x)/2)=X0(k)-exp(-1j*2*pi/length(x)*(k-
1))*X1(k);
            end
end

function X=FFT8(x)
```

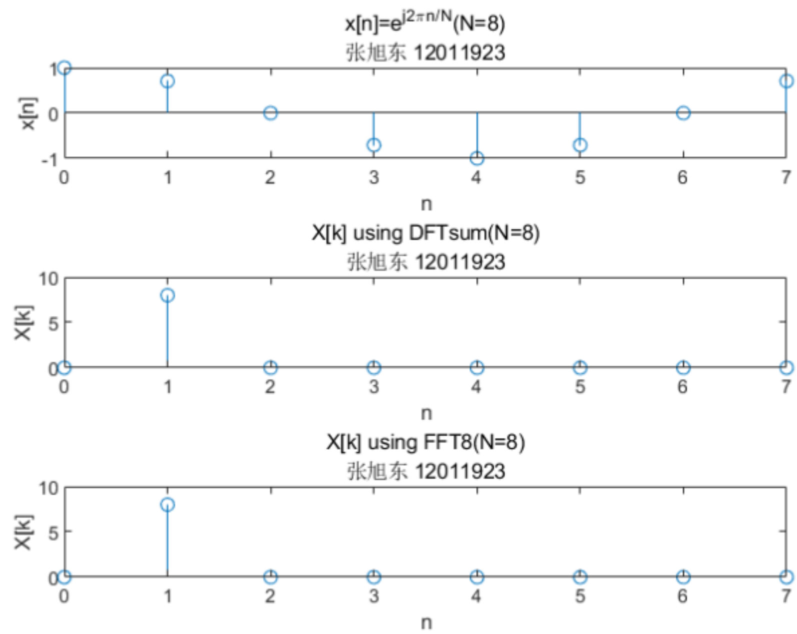
```
        %x的长度为8
        X=zeros(1,length(x));
        x0=x(1:2:length(x));
        x1=x(2:2:length(x));
        X0=FFT4(x0);
        X1=FFT4(x1);
        for k=1:length(x)/2
            X(k)=X0(k)+exp(-1j*2*pi/length(x)*(k-1))*X1(k);
        end
        for k=1:length(x)/2
            X(k+length(x)/2)=X0(k)-exp(-1j*2*pi/length(x)*(k-
1))*X1(k);
        end
end
```

The correctness of function $FFT8$ is tested using the following signals:

$$\begin{array}{lcc} 1. & x[n]=\delta[n], & N=8 \\ 2. & x[n]=1 & N=8 \\ 3. & x[n]=e^{j2\pi n/8} & N=8 \end{array} \tag{13}$$

The result of test is shown below:





Comparing the plot using function $FFT8$ to the plot using function $DFTsum$, it is can be found that the result generated by function $FFT8$ is the same as that generated by function $DFTsum$, which means the function $FFT8$ is correct.

The output of $FFT8$ for $x[n] = 1 \qquad N = 8$ is $X = [8, 0, 0, 0, 0, 0, 0, 0]$.

According to the flow diagram of 8-point FFT, the total number of multiplies by twiddle factors required is:

$$TotalNumber = 8 \tag{14}$$

Further, for N-point $DFT(N = 2^p)$, it can be divided into $logN(p = logN)$ stage according to the theory mentioned above, and the multiplies taken place in every stage is $\frac{N}{2}$ because even-numbered data in every stage is $\frac{N}{2}$ and every even-numbered data need to multiplies a twiddle factor. So, the total number of multiplication is:

$$Totalnumber = \frac{N}{2} logN \tag{15}$$

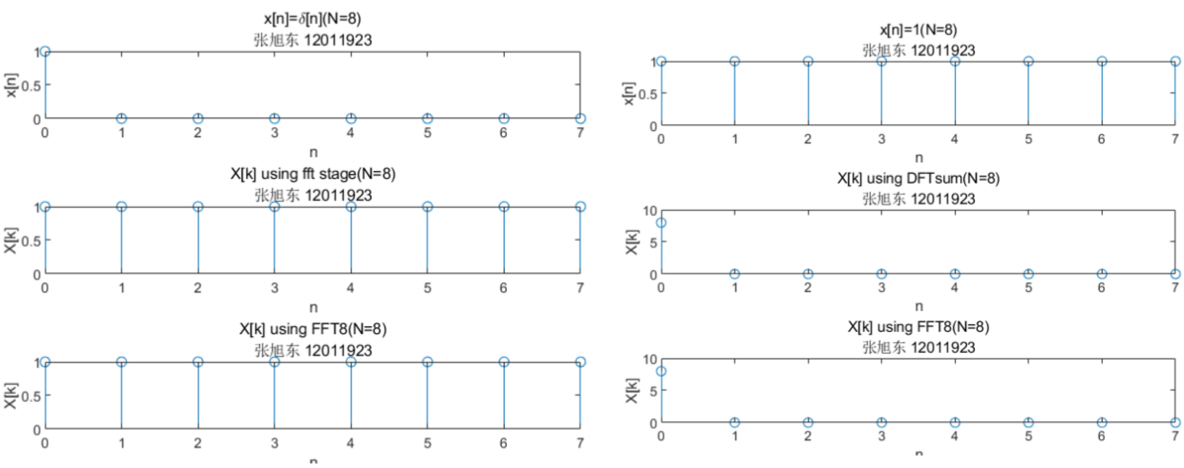
When $p = 10(N = 2^p)$, the number of multiplies required for FFT is $2^p p = 10 \cdot 2^{10}$ while that required for direct implementation is 10^{10}. The former is much less than the latter, which embodies the superiority of FFT in time complexity.

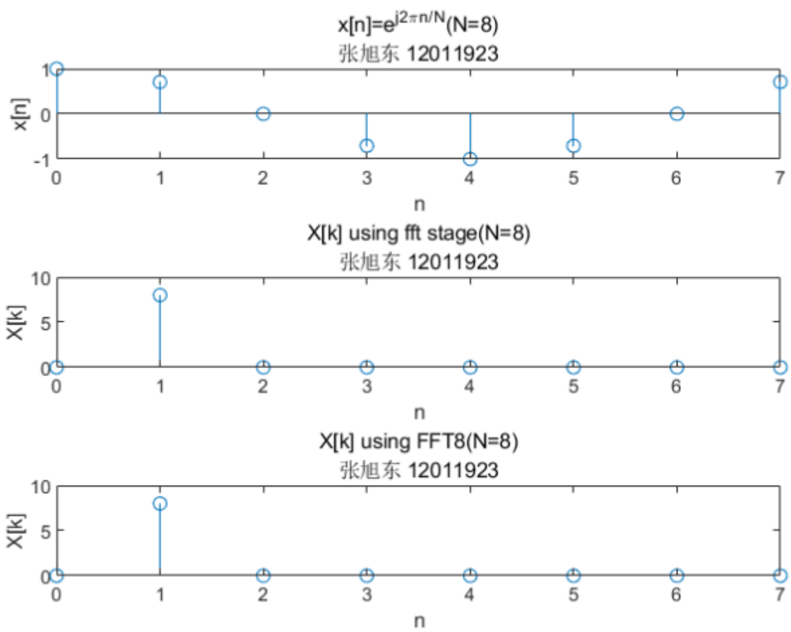
Obviously, it is easy to find that the function $FFT4$ and $FFT8$ have almost the same form. However, it is redundant to write a separate function for each stage of FFT. To avoid this, a recursive function named fft_stage is written to call itself within the body. The code is shown below:

```
function X=fft_stage(x)
    N=length(x);
```

```
    k=0:N/2-1;
    if N==2
        X=FFT2(x);
    else
        x0=x(1:2:N);
        x1=x(2:2:N);
        X0=fft_stage(x0);
        X1=fft_stage(x1);
        wkn=exp(-1j*2*pi*k/N);
        X=[(X0+X1.*wkn) (X0-X1.*wkn)];
    end

end
```

Three signals in formula (13) is used to test the correctness of function fft_stage and the result of test is shown below:





From the three above pictures, the result of $FFT8$ and $fftstage$ *is the same, which means verifies the function* $fftstage$.

4. Conclusion

From the section (2), with the number of points in the DFT, the level of similarity between DFT and $DTFT$ is getting higher and higher. The technique to increase number of points in the DFT is known as "zero padding", which increases number of points by adding a certain number of zeros to the tail of input signal and may produce a finer sampling of the $DTFT$.

From the section (3), with repeatedly using divide-and-conquer and recursion, the time complexity of FFT is much less than that of DFT with the number of points in the DFT increasing. Time complexity of DFT is N^2 while that of FFT is $NlogN$ (N is the number of points in the DFT). The proposal of FFT greatly reduce the cost of compute, which makes it possible to compute larger amount of data.