LAB5 REPORT

SID:12012505

Name: 占陈郅

Part 1: Introduction.

- The last laboratory covers the Discrete Fourier Transform (DFT). This laboratory will continue the discussion of the DFT and will introduce the FFT.
- The main content contains in this lab:
  - Build DTFT Function
  - Learn *fftshift*
  - Zero Padding
  - Divide-and-Conquer DFT

Part 2: Result & Analysis.

5.2 Continuation of DFT Analysis

5.2.1 Shifting the Frequency Range

- Part 1 - Illustrate a representation for the DFT of a Hamming window x of length N = 20.

  Code:

```
x1 = hamming(20);

figure(1)
stem(0:19,abs(DFTsum(x1))),xlabel('n'),ylabel('abs(DFTsum(x1))'),title('magnitude plot for
DFT(x1)')
```

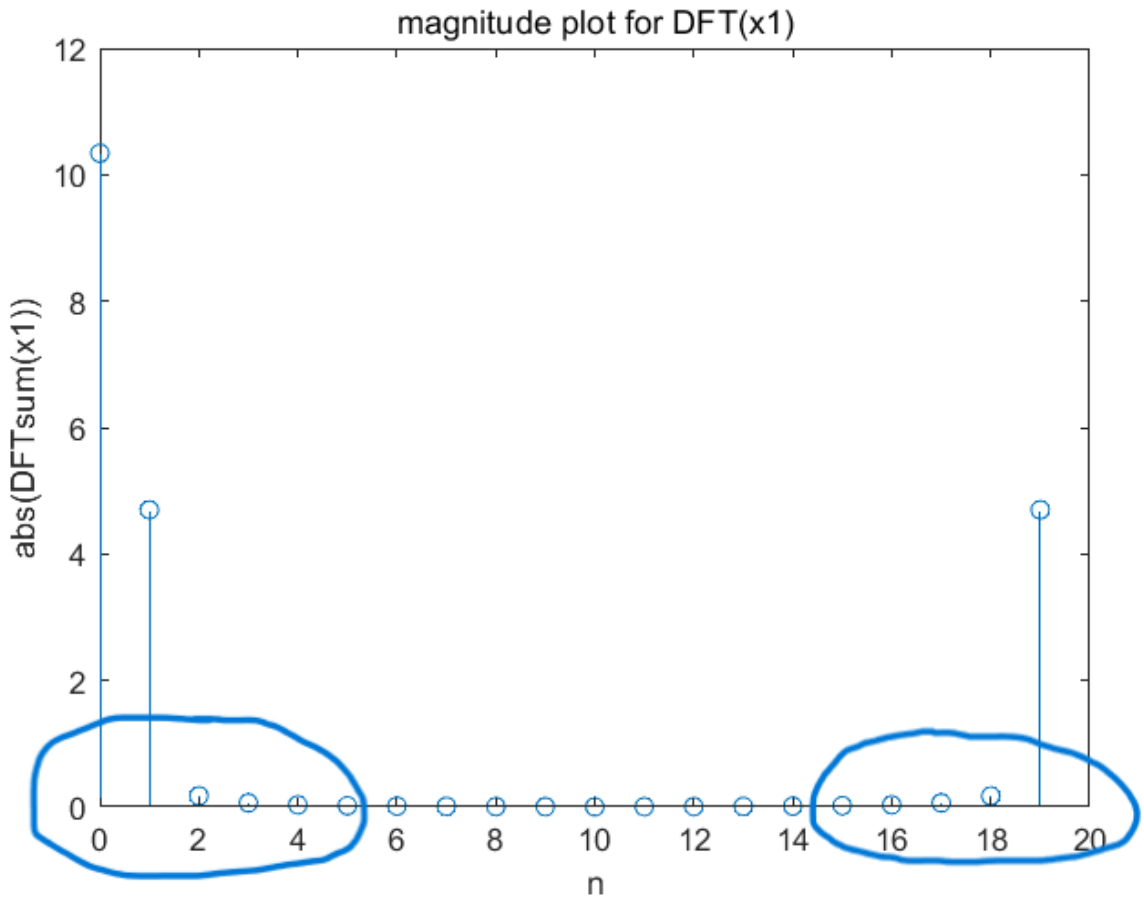

fig.1 Magnitude plot for DFT(x1) with circling the low frequency components

Analysis:

The circle above shows the low frequency components.

- Part 2 - Write a function to compute samples of the DTFT and their corresponding frequencies in the range -π to π.

  Code:

```
x1 = hamming(20);
[X,w] = DTFTsamples(x1);
figure(2)
stem(w,abs(X)),xlabel('w'),ylabel('DTFT(x1)'),title('magnitude plot for DTFT(x1)')
```

```
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;% shift the range from [0,2*pi] to [-pi,pi]
w = fftshift(w);
X = fftshift(DFTsum(x));
display(w);display(X);
end
```

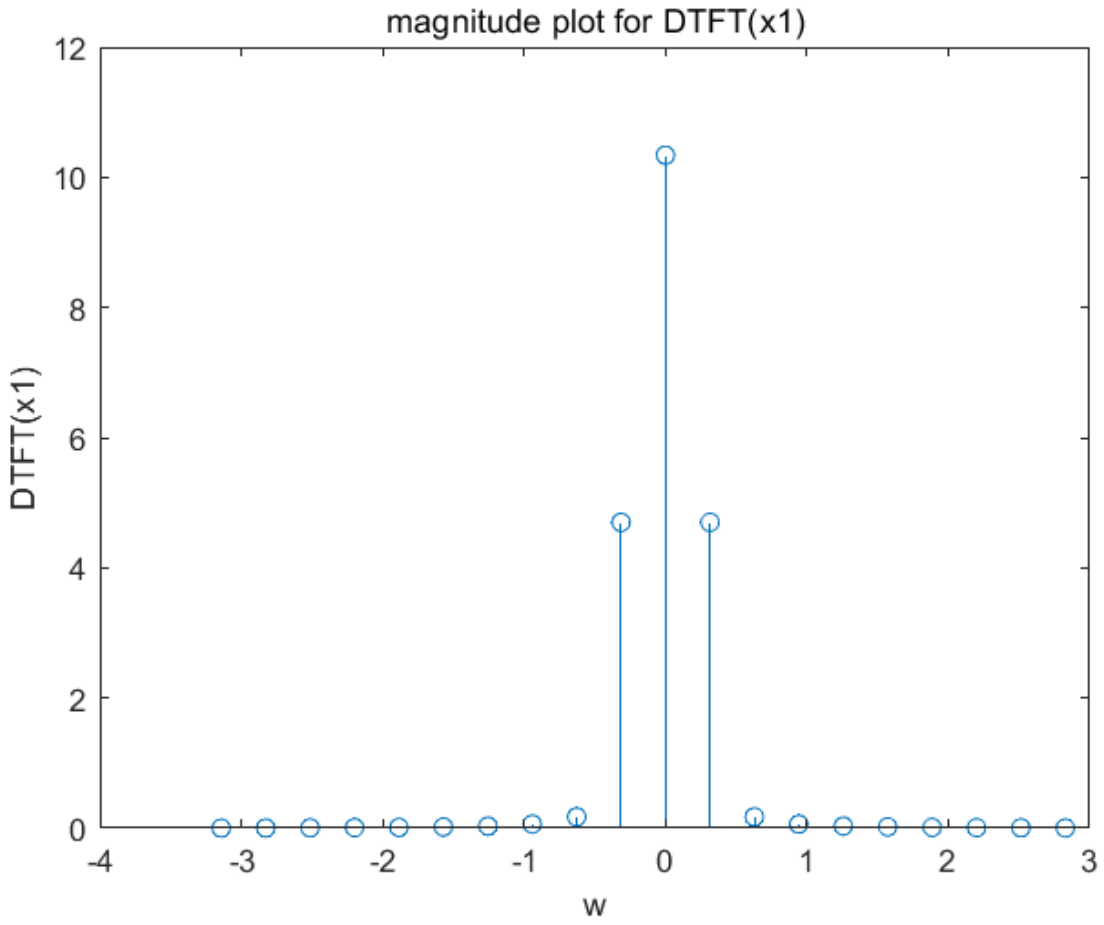

fig.2 Magnitude plot for DTFT(x1)

Analysis:

In the DTFTsample function, w = 2*pi*k/N is used to transform x in time domain to frequency domain, then shift the w to -π ~ π, and use *fftshift* to get the magnitude of DTFT of x.

5.2.2 Zero Padding

- In the following, you will compute the DTFT samples of $x(n)$ using both $N = 50$ and $N = 200$ point FT's. Notice that when $N = 200$, most of the samples of $x[n]$ will be zeros because $x[n] = 0$ for $n \geq 50$.This technique is known as "zero padding", and may be used to produce a finer sampling of the DTFT. For $N = 50$ and $N = 200$, do the following:

  1. Compute the vector x containing the values $x[0], \ldots, x[N-1]$.
  2. Compute the samples of $X[k]$ using your function DTFTsamples.
  3. Plot the magnitude of the DTFT samples versus frequency in rad/sample.

Code:

```
x1 = sin(0.1*pi*[0:49]);
n = zeros(1,0);%change value of N
x = horzcat(x1,n);%zero padding
[X,w] = DTFTsamples(x);
stem(w,abs(X)),xlabel('w'),ylabel('DTFT(x)'),title('magnitude plot for DTFT(x)')
```

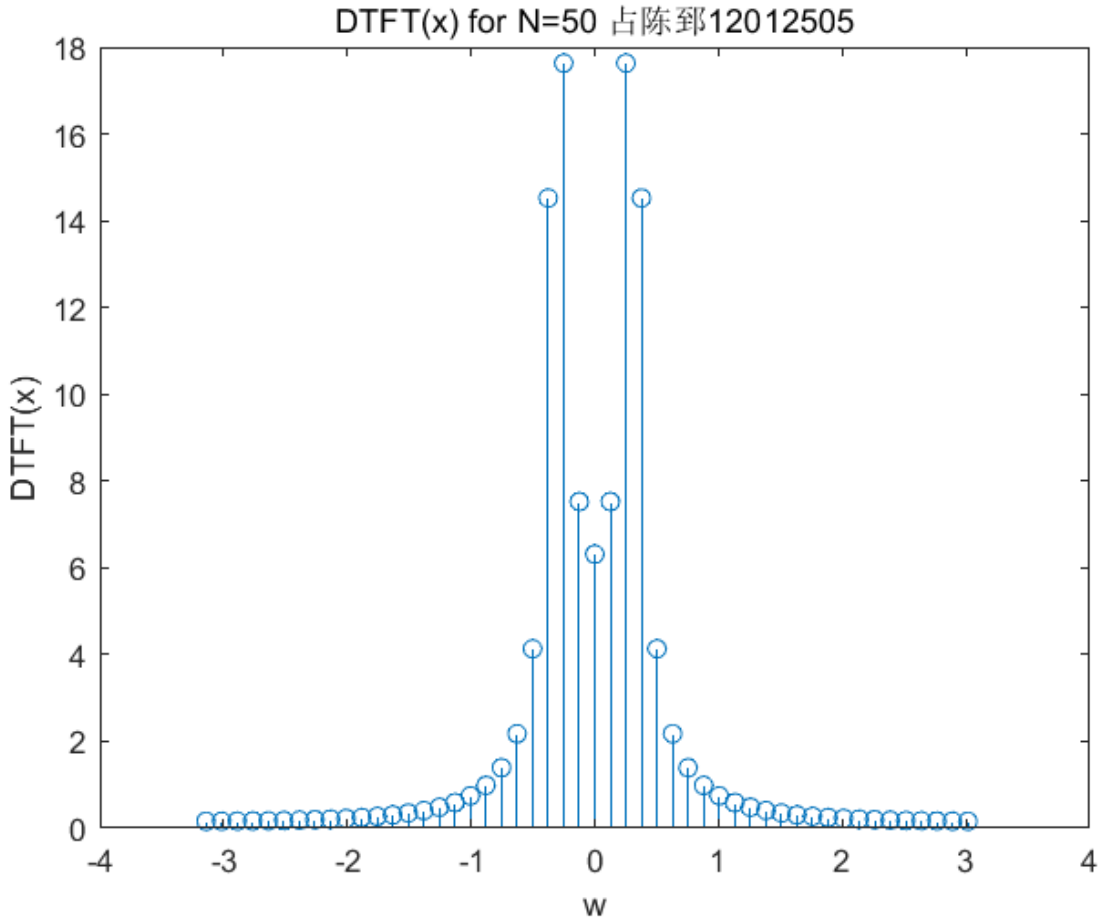

fig.3 DTFT(x1) without zero padding

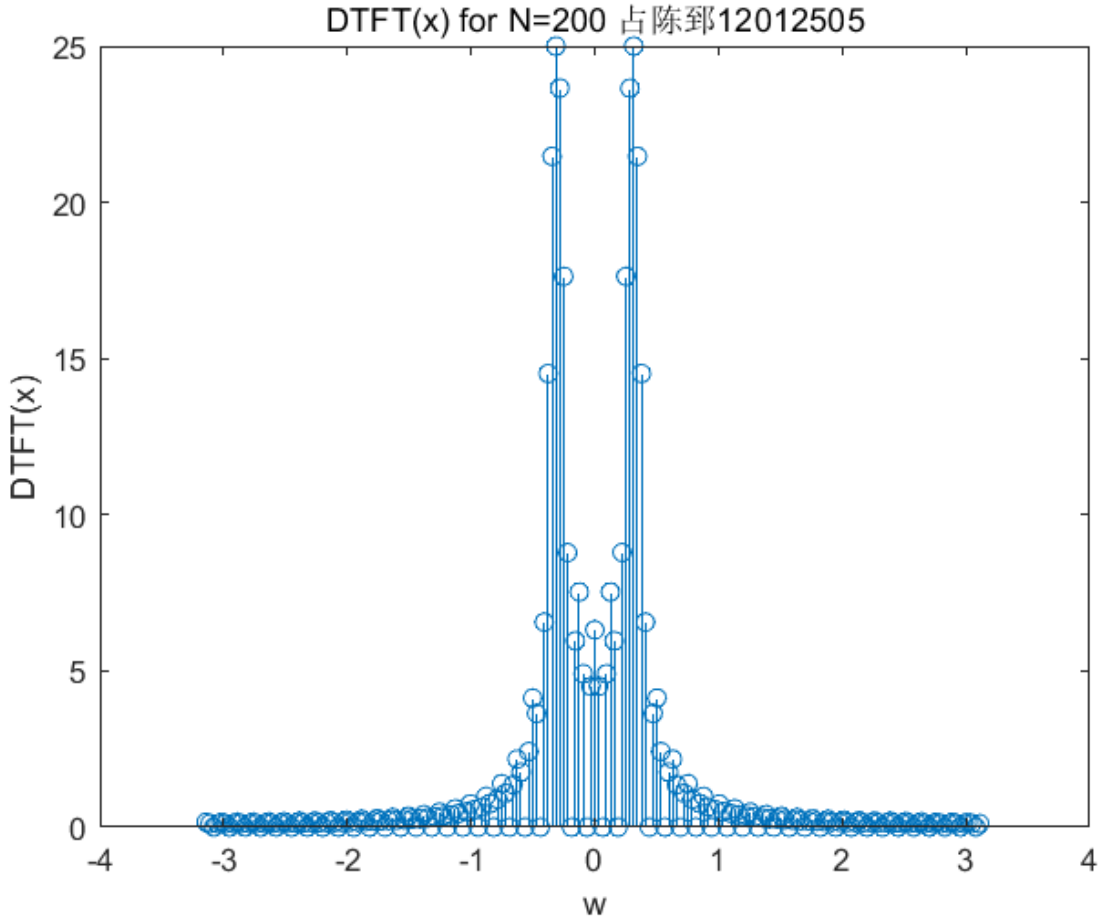

fig.4 DTFT(x) with zero padding

Analysis:

The second plot(N=200) looks more like true DTFT of x.

Why would that happen? To answer in short:

1. Zero padding do not change the frequency component of DTFT.
2. DFT sampling DTFT in one period.

Whatever N=50 or N=200, the results of DTFT are still periodic, but in the N=200 case, there are more 150 times sampling in one period of DTFT than N=50 case.

In conclusion, zero padding is the same as frequency-domain interpolation.

5.3 The Fast Fourier Transform Algorithm

5.3.1 Implementation of Divide-and-Conquer DFT

- Write a Matlab function dcDFT

  Code:

```
function X = dcDFT(x)
N = length(x);
x0 = x(1:2:N);  % Even part of x[n]
x1 = x(2:2:N);  % Odd part of x[n]
X0 = DFTsum(x0); X0 =[X0 X0];
X1 = DFTsum(x1); X1 = [X1 X1];
X = X0+ exp(-1j*2*pi/N*[0:1:N-1]).*X1;% equation5.13, using twiddle factior.
end
```

- Test

  Code:

```
clear;
n = 0:9;
x1 = (n==0);% case 1
x2 = ones(1,10);% case 2
x3 = exp(1j*2*pi*n/10);% case 3
N = 0:9;
sgtitle('dcDFT and DFTsum')
subplot(321),stem(n,abs(dcDFT(x1))),xlabel('N'),ylabel('dcDFT(x1)'),title('dcDFT(x1)');
subplot(322),stem(n,abs(DFTsum(x1))),xlabel('N'),ylabel('DFTsum(x1)'),title('DFTsum(x1)');
subplot(323),stem(n,abs(dcDFT(x2))),xlabel('N'),ylabel('dcDFT(x2)'),title('dcDFT(x2)');
subplot(324),stem(n,abs(DFTsum(x2))),xlabel('N'),ylabel('DFTsum(x2)'),title('DFTsum(x2)');
subplot(325),stem(n,abs(dcDFT(x3))),xlabel('N'),ylabel('dcDFT(x3)'),title('dcDFT(x3)');
subplot(326),stem(n,abs(DFTsum(x3))),xlabel('N'),ylabel('DFTsum(x3)'),title('DFTsum(x3)');
```

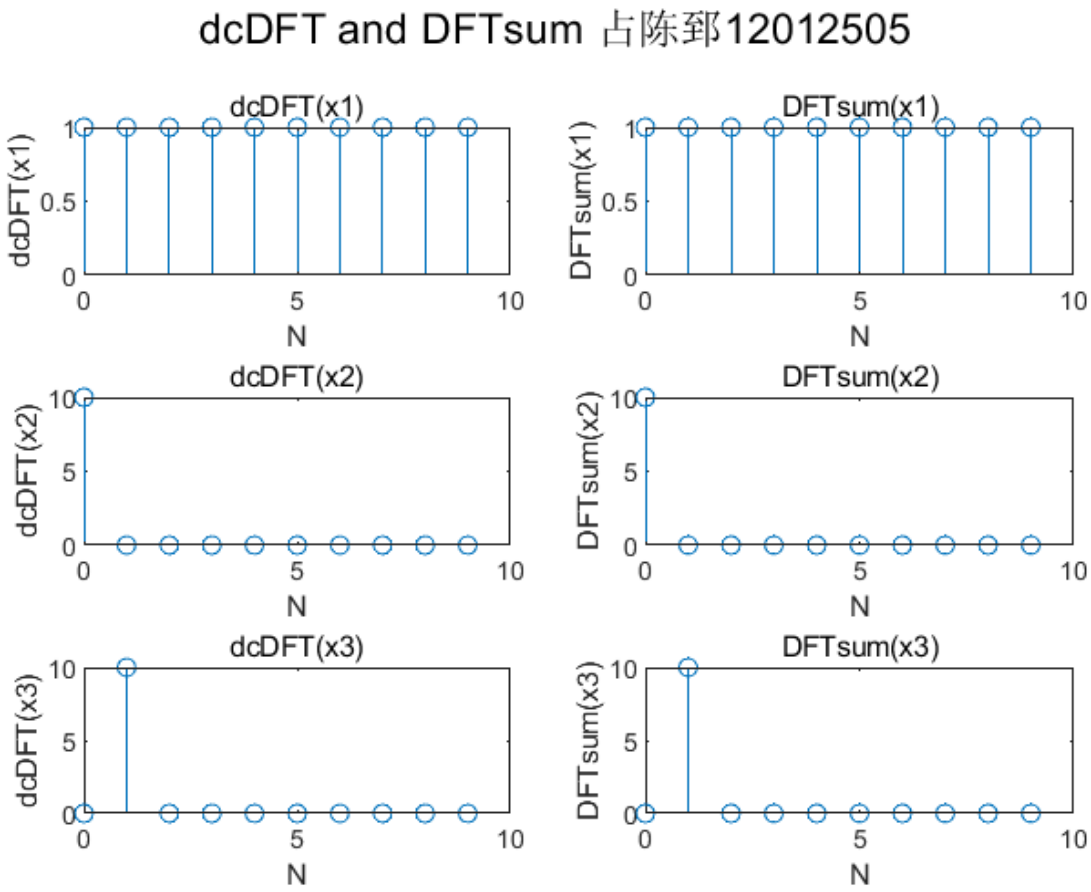

fig.5 dcDFT and DFTSum

Analysis:

The figure above shows dcDFT is as good as DFTsum.

The number of multiplies that are required in this approach to computing an N point DFT is

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

5.3.2 Recursive Divide and Conquer

- Write three Matlab functions to compute the 2-, 4-, and 8-point FFT's using the syntax

1. X=FFT2(x)

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

2. X=FFT4(x)

```
function X = FFT4(x)
 r = 0:1;
 X0 = FFT2([x(1) x(3)]);
 X1 = FFT2([x(2) x(4)]);
 temp = X1.*exp(-j*2*pi/4*r);
 X = [X0+temp X0-temp];
end
```

3. X=FFT8(x)

```
function X = FFT8(x)
 r = 0:3;
 X0 = FFT4([x(1) x(3) x(5) x(7)]);
 X1 = FFT4([x(2) x(4) x(6) x(8)]);
 temp = X1.*exp(-j*2*pi/8*r);
 X = [X0+temp X0-temp];
end
```

- Test your function FFT8 by using it to compute the DFT's of the following signals. Compare these results to the previous ones.

```
clear;
x = ones(1,8);
X = FFT8(x);
display(X)
```

```
X = 1×8
     8     0     0     0     0     0     0     0
```

```
Xn = fft(x)
```

```
Xn = 1×8
     8     0     0     0     0     0     0     0
```

fig.6 FFT8 compare to fft

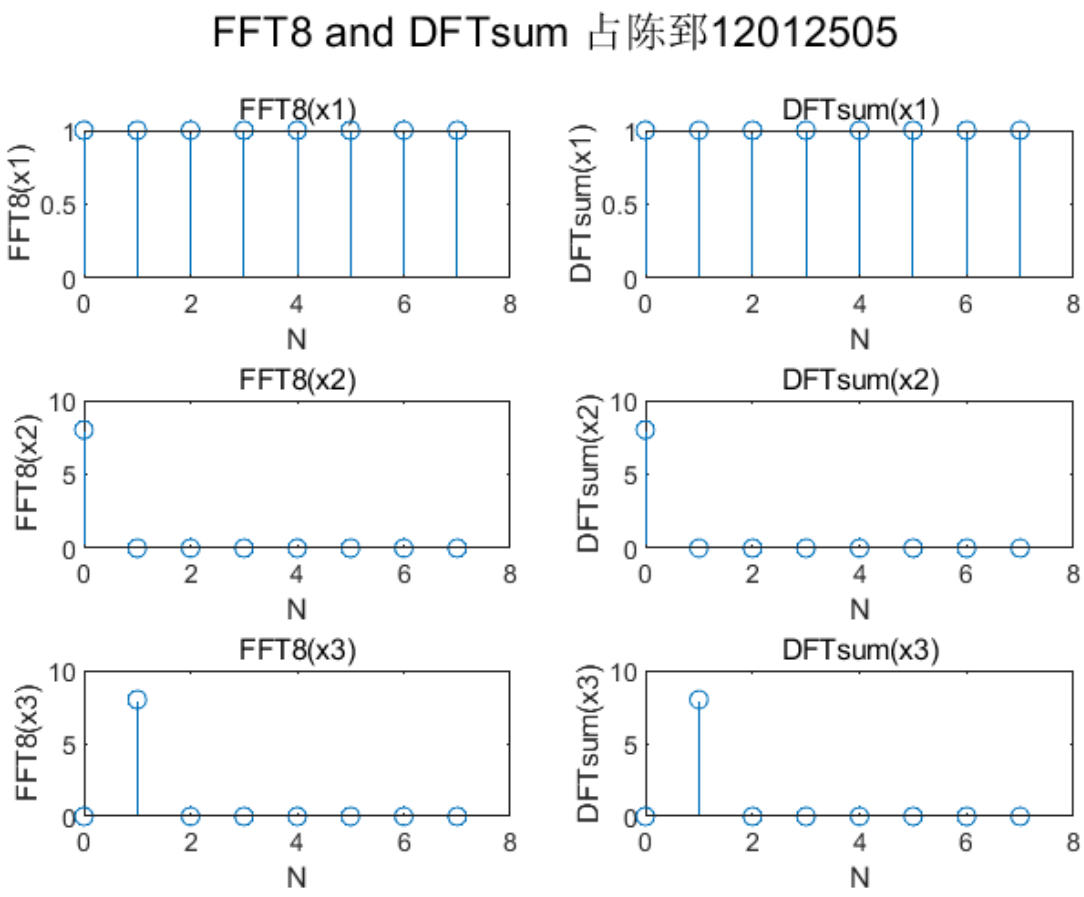

fig.7 FFT8 compare to DFTsum

Code:

```
clear;
x = ones(1,8);
X = FFT8(x);
display(X)
Xn = fft(x)

clear
n=0:1:7;
x1 = (n==0);
x2 = ones(1,8);
x3 = exp(j*2*pi*n/8);
N = 0:7;
sgtitle('FFT8 and DFTsum 占陈郅12012505')
subplot(321),stem(n,abs(FFT8(x1))),xlabel('N'),ylabel('FFT8(x1)'),title('FFT8(x1)');
subplot(322),stem(n,abs(DFTsum(x1))),xlabel('N'),ylabel('DFTsum(x1)'),title('DFTsum(x1)');
subplot(323),stem(n,abs(FFT8(x2))),xlabel('N'),ylabel('FFT8(x2)'),title('FFT8(x2)');
subplot(324),stem(n,abs(DFTsum(x2))),xlabel('N'),ylabel('DFTsum(x2)'),title('DFTsum(x2)');
subplot(325),stem(n,abs(FFT8(x3))),xlabel('N'),ylabel('FFT8(x3)'),title('FFT8(x3)');
subplot(326),stem(n,abs(DFTsum(x3))),xlabel('N'),ylabel('DFTsum(x3)'),title('DFTsum(x3)');
```

- Calculate the total number of multiplies **by twiddle factors**.

  For 8-point FFT.

$$\frac{8}{2}\times log_2 8 = 12$$

  For N = 2^p point FFT.

$$\frac{N}{2}\times log_2 N = p2^{p-1}$$

The number of multiplications could be even lower if we consider $W_N^0 = 1$ and $W_N^{N/2} = -1$ in Strategy2. In brief

$$\frac{N}{2}\times(log_2 N - 2) + 1$$

- Write a recursive function X=fft_stage(x) that performs one stage of the FFT algorithm for a power-of-2 length signal.

  Code:

```
function X = fft_stage(x)
N = length(x);  % Determine the length of the input signal.
if N == 2 % If N=2, then the function should just compute the 2-pt DFT as in (5.14), and then
return.
    X(1) = x(1)+x(2);
    X(2) = x(1)-x(2);
else %  If N>2, then the function should perform the FFT steps described previously (i.e.
decimate, compute (N/2)-point DFTs, re-combine), calling fft_stage to compute the (N/2)-point
DFTs
    r = 0:1:N/2-1;
    xeven = x(1:2:N);
    xodd = x(2:2:N);
    X0 = fft_stage(xeven);
    X1 = fft_stage(xodd);
    temp = X1.*exp(-j*2*pi/N*r);
    X = [X0+temp X0-temp];
end
end
```

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

  Code:

```
clear
n=0:1:7;
x1 = (n==0);
x2 = ones(1,8);
x3 = exp(j*2*pi*n/8);
N = 0:7;
sgtitle('FFT8 and fft\_stage 占陈郅12012505')
subplot(321),stem(n,abs(FFT8(x1))),xlabel('N'),ylabel('FFT8(x1)'),title('FFT8(x1)');
subplot(322),stem(n,abs(fft_stage(x1))),xlabel('N'),ylabel('fft\_stage(x1)'),title('fft\_stag
e(x1)');
subplot(323),stem(n,abs(FFT8(x2))),xlabel('N'),ylabel('FFT8(x2)'),title('FFT8(x2)');
subplot(324),stem(n,abs(fft_stage(x2))),xlabel('N'),ylabel('fft\_stage(x2)'),title('fft\_stag
e(x2)');
subplot(325),stem(n,abs(FFT8(x3))),xlabel('N'),ylabel('FFT8(x3)'),title('FFT8(x3)');
subplot(326),stem(n,abs(fft_stage(x3))),xlabel('N'),ylabel('fft\_stage(x3)'),title('fft\_stag
e(x3)');
```

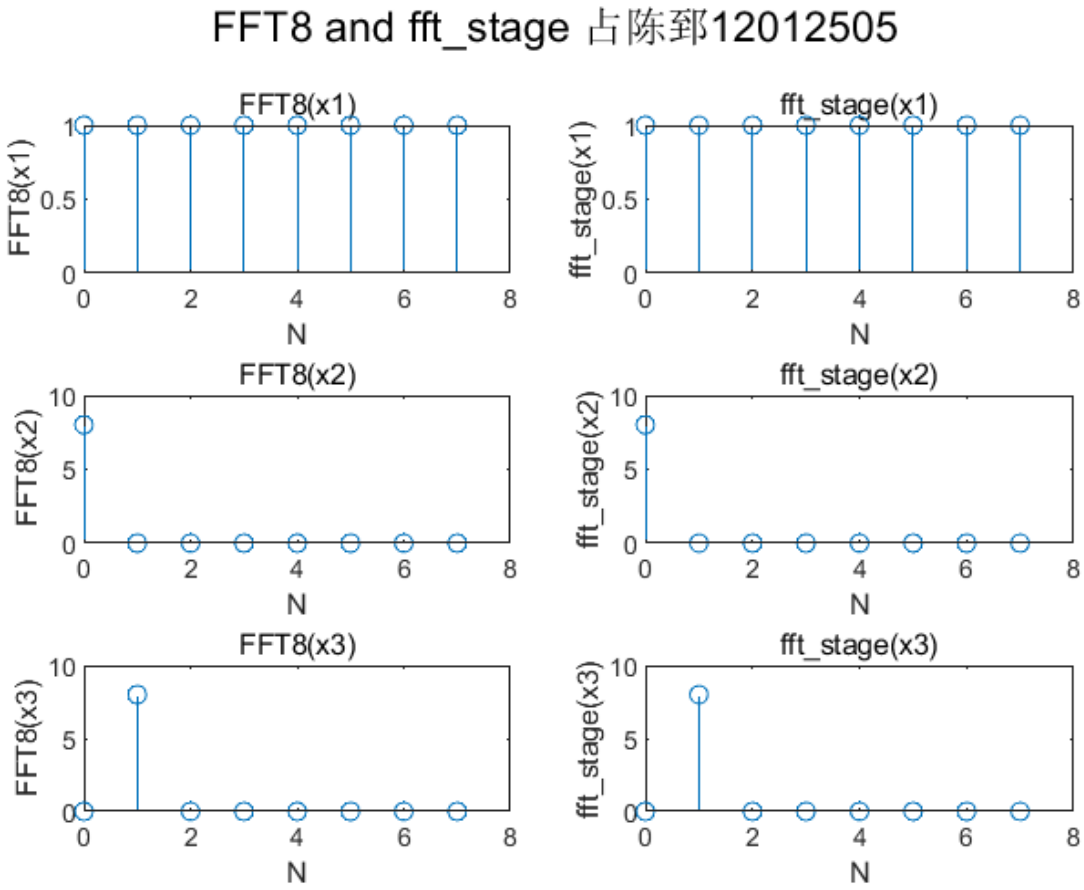

fig.8 FFT8 compare to fft_stage

fft_stage returns the same results as FFT8.

Part 3: Summary & Experience.

- This lab mainly talks about Fast Fourier Transform, with its nature, its qualities and variants.
- The Fast Fourier Transform FFT is a more efficient algorithm for computing the discrete Fourier transform with O (N log_2 N) computational complexity, which is more scalable than the original DFT with O (N^2) computational complexity.
- A common FFT implementation requires limiting the number of input data points to the form N=2^m (multiplication of 2). If the number of data you have is not several powers of 2, you can make up the next multiplication of 2 by adding zeros (zero padding) to the end. The added zeros do not affect the frequency domain spectrum, but only increase the apparent data density, i.e., the extra frequency domain points are actually linear interpolations of the valid data.