**实验报告**

信息工程

曾翌晖

3210106327

2023年11月4日

专业：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

课程名称： 数字信号处理 成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实验名称： FFT算法编程 实验类型： 设计

**一、实验目的和要求**

FFT是快速计算DFT的一类算法的总称。通过序列分解，用短序列的DFT代替长序列的DFT，使得计算量大幅下降。基4-FFT是混合基FFT的一个特例。

通过编写基4-FFT及基2-FFT算法程序，加深对FFT思路、算法结构的理解。

**二、实验内容和步骤**

编写16点基4-FFT算法的Matlab/Python程序。

产生16点输入序列x，**用自己的学号**作为前10点的抽样值，后面补6个零值抽样。算出16点频谱序列X，用stem(X)（Matlab函数，Python可用类似功能的函数代替）显示频谱图形。

编写基2DIT-FFT或者基2DIF-FFT程序，验证基4-FFT的频谱序列计算结果。

撰写实验报告。

**三、主要仪器设备**

用Matlab/Python。

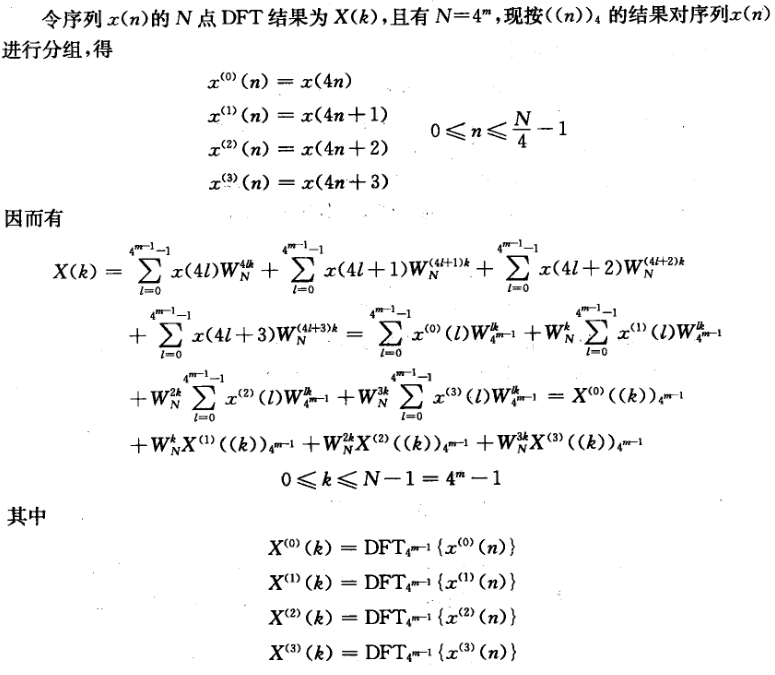
**四、实验数据记录和处理**

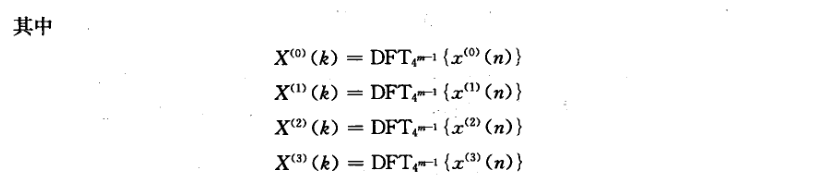
4.1基4-FFT算法思路、流图结构简述如下

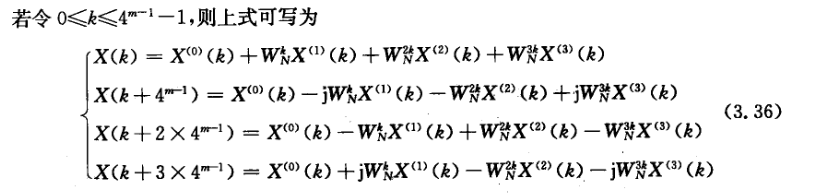
算法思路：它是基于蝶形运算的思想，通过将DFT分解为多个较小规模的DFT来降低计算复杂度。基4FFT算法是把长度N=4t的序列一分为四，将N点DFT表示为4个N/4点DFT的线性组合。然后再把N/4点DFT一分为四表示为四个N/16点的DFT。如此重复下去直至分解成4点DFT的运算。通过这样的分组和位序调整，进而通过多级蝶形复合运算，就可以通过低点数的DFT得到高点数的DFT，从而简化计算量。

具体步骤如下：

第2页

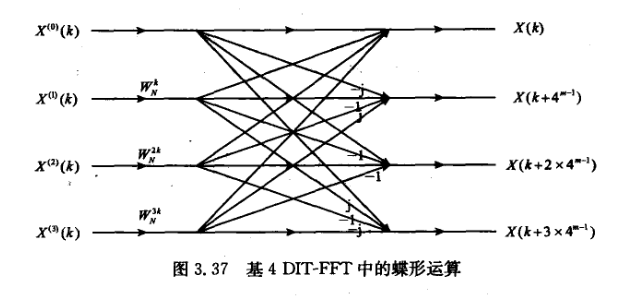






流图结构：

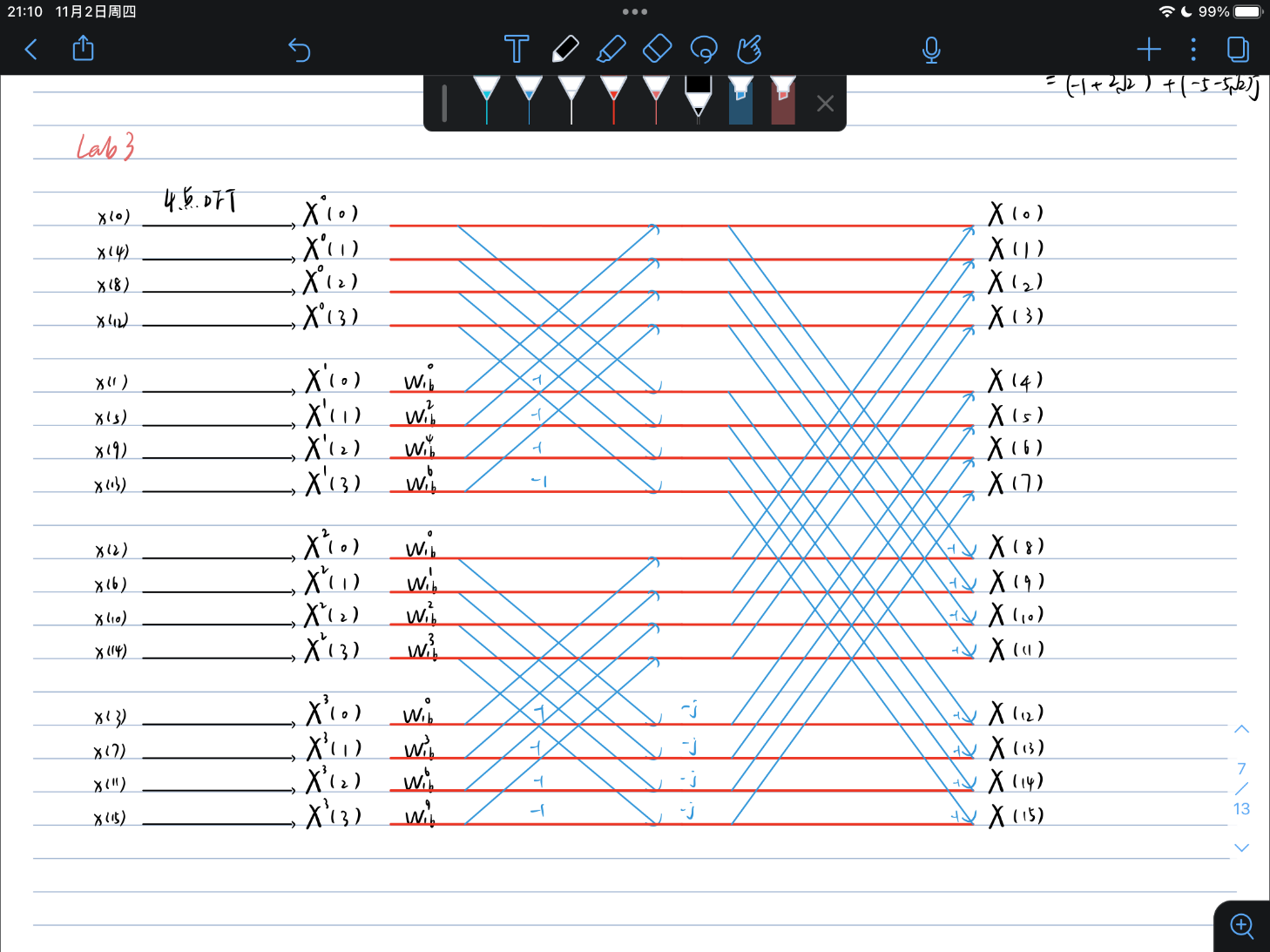
第3页



基 4 运算需要 3 次复数乘法和 12 次复数加法。

4.2 16点基4-FFT算法的流图绘出如下

流图结构：



4.3 16点基4-FFT算法的Matlab/Python程序如下（列出）

% 16点基4-FFT算法

% 定义输入、初始化输出

N = 16;

x = [3 2 1 0 1 0 6 3 2 7 0 0 0 0 0 0];

X = zeros(1,16);

% 对输入的16个数字进行分组（MATLAB下标从1开始）

x\_1 = [x(1) x(5) x(9) x(13)];

x\_2 = [x(2) x(6) x(10) x(14)];

x\_3 = [x(3) x(7) x(11) x(15)];

x\_4 = [x(4) x(8) x(12) x(16)];

x\_matrix = [x\_1.' x\_2.' x\_3.' x\_4.'];

% 第一级蝶形复合，根据定义做4点DFT运算

k = 0:3;

W4 = exp(-1i\*2\*pi\*(k'\*k)/4);% 定义DFT变换的矩阵W

W = exp(-1i\*2\*pi/16);

% 二级蝶形复合

% 定义X0 X1 X2 X3为4元素向量，构成16个一级蝶形复合的结果

X\_matrix = W4 \* x\_matrix;

X0 = X\_matrix(:,1);

X1 = X\_matrix(:,2);

X2 = X\_matrix(:,3);

X3 = X\_matrix(:,4);

for i = 1:4

matrix1 = W4\*[X0(i) W^(i-1)\*X1(i) W^(2\*i-2)\*X2(i) W^(3\*i-3)\*X3(i)].';

X(i) = matrix1(1);

X(i+4) = matrix1(2);

X(i+8) = matrix1(3);

X(i+12) = matrix1(4);

end

% 时域图

n = 0:N-1;

figure;

stem(n, x);title('输入序列');

第4页

% 频域图

figure;

subplot(2,2,1);stem(n,abs(X));title('幅度谱');

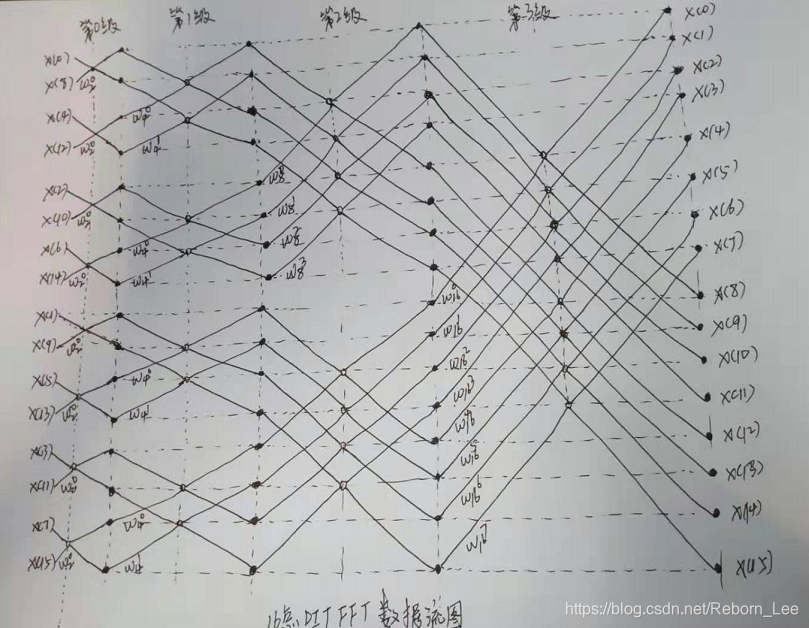
subplot(2,2,2);stem(n,real(X));title('频谱实部');

subplot(2,2,3);stem(n,imag(X));title('频谱虚部');

subplot(2,2,4);stem(n,angle(X));title('相角');

4.4 16点基2-FFT算法Matlab/Python程序如下（列出）

蝶形运算图：



% 16点基2-FFT算法

% 定义输入、初始化输出及蝶形复合结果

N = 16;

x = [3 2 1 0 1 0 6 3 2 7 0 0 0 0 0 0];

% 在一级蝶形复合之前，先按照顺序将输入序列分组

group1 = [x(1) x(9)];group2 = [x(5) x(13)];group3 = [x(3) x(11)];group4 = [x(7) x(15)];

group5 = [x(2) x(10)];group6 = [x(6) x(14)];group7 = [x(4) x(12)];group8 = [x(8) x(16)];

matrix = reshape([group1; group2; group3; group4; group5; group6; group7; group8], 8, 2);

% 一级蝶形复合

W0 = [1 1;1 -1]; % 定义旋转因子

X1 = W0 \* matrix.'; % 定义存储一级蝶形复合结果的2×8矩阵

% 二级蝶形复合

W = exp(-1i\*2\*pi/16); % 定义W

X2 = zeros(4,4);

% 通过循环运算后为X2赋值

for i = 1:2

for j = 1:4

matrix1 = W0\*[X1(i,2\*j-1) W^(4\*i-4)\*X1(i,2\*j)].';

X2(i,j) = matrix1(1);

X2(i+2,j) = matrix1(2);

end

end

% 三级蝶形复合

X3 = zeros(8,2);

% 通过循环运算后为X3赋值

for i = 1:4

for j = 1:2

matrix2 = W0\*[X2(i,2\*j-1) W^(2\*i-2)\*X2(i,2\*j)].';

X3(i,j) = matrix2(1);

X3(i+4,j) = matrix2(2);

end

end

% 四级蝶形复合

X = zeros(16,1);

% 通过循环运算后为X赋值

第5页

for i = 1:8

matrix3 = W0\*[X3(i,1) W^(i-1)\*X3(i,2)].';

X(i) = matrix3(1);

X(i+8) = matrix3(2);

end

% 时域图

n = 0:N-1;

figure;

stem(n, x);title('输入序列');

% 频域图

figure;

subplot(2,2,1);stem(n,abs(X));title('幅度谱');

subplot(2,2,2);stem(n,real(X));title('频谱实部');

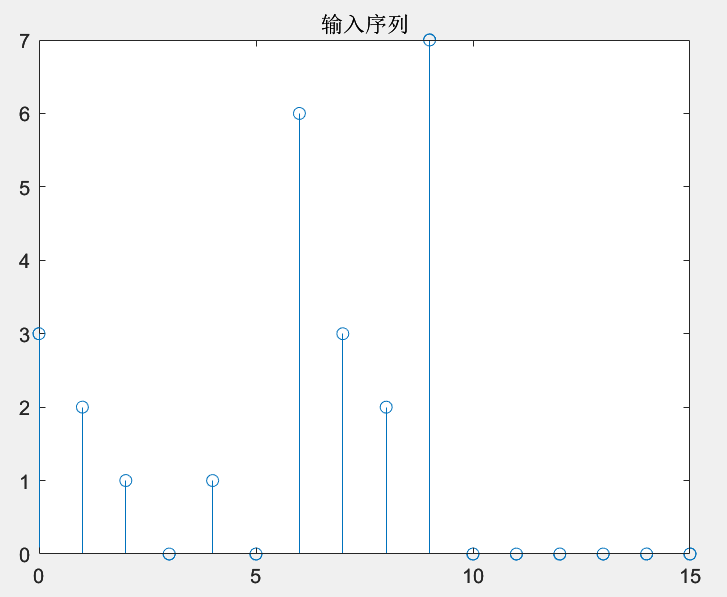
subplot(2,2,3);stem(n,imag(X));title('频谱虚部');

subplot(2,2,4);stem(n,angle(X));title('相角');

4.5 用自己的学号构成的输入序列为（列出数值，画出图形）

数值：x = [3 2 1 0 1 0 6 3 2 7 0 0 0 0 0 0];

图形：



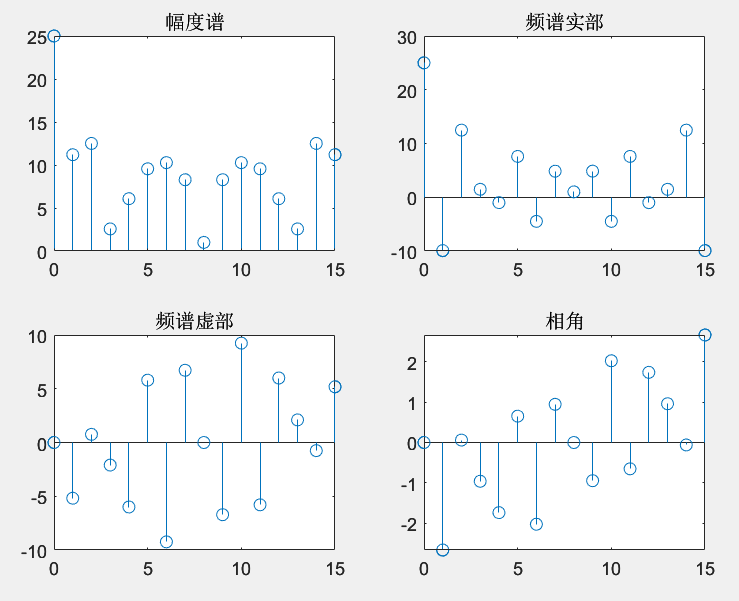
第6页

4.6 对应的基4-FFT输出频谱序列为（列出数值，画出图形）

输出频谱序列：

|  |  |  |  |
| --- | --- | --- | --- |
| X(0) | X(1) | X(2) | X(3) |
| 25.0000000000000 + 0.00000000000000i | -9.92657016602303 - 5.18438060357565i | 12.4852813742386 + 0.757359312880715i | 1.47406644701202 - 2.10198840328326i |
| X(4) | X(5) | X(6) | X(7) |
| -1.00000000000000 - 6.00000000000000i | 7.59700136485346 + 5.79750653332841i | -4.48528137423857 - 9.24264068711929i | 4.85550235415756 + 6.71511433303601i |
| X(8) | X(9) | X(10) | X(11) |
| 1.00000000000000 - 4.89858719658941e-16i | 4.85550235415756 - 6.71511433303601i | -4.48528137423857 + 9.24264068711928i | 7.59700136485346 - 5.79750653332840i |
| X(12) | X(13) | X(14) | X(15) |
| -1.00000000000000 + 6.00000000000000i | 1.47406644701202 + 2.10198840328326i | 12.4852813742386 - 0.757359312880712i | -9.92657016602303 + 5.18438060357565i |

图形：



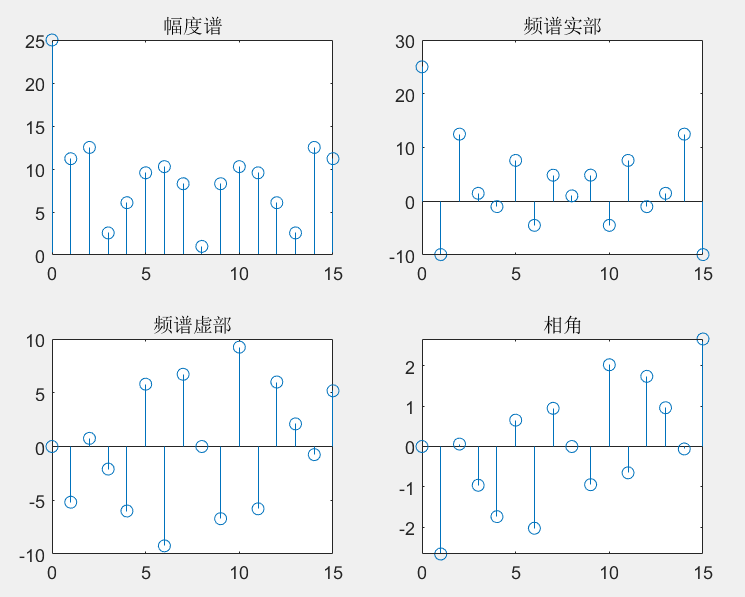
第7页

4.7 对应的基2-FFT输出频谱序列为（列出数值，画出图形）

输出频谱序列：

|  |  |  |  |
| --- | --- | --- | --- |
| X(0) | X(1) | X(2) | X(3) |
| 25.0000000000000 + 0.00000000000000i | -9.92657016602303 - 5.18438060357565i | 12.4852813742386 + 0.757359312880714i | 1.47406644701202 - 2.10198840328326i |
| X(4) | X(5) | X(6) | X(7) |
| -1.00000000000000 - 6.00000000000000i | 7.59700136485346 + 5.79750653332840i | -4.48528137423857 - 9.24264068711928i | 4.85550235415756 + 6.71511433303601i |
| X(8) | X(9) | X(10) | X(11) |
| 1.00000000000000 + 0.00000000000000i | 4.85550235415756 - 6.71511433303601i | -4.48528137423857 + 9.24264068711929i | 7.59700136485345 - 5.79750653332841i |
| X(12) | X(13) | X(14) | X(15) |
| -0.999999999999999 + 6.00000000000000i | 1.47406644701202 + 2.10198840328326i | 12.4852813742386 - 0.757359312880717i | -9.92657016602303 + 5.18438060357566i |

图形：



第8页

**五、结果分析及讨论**

基2-FFT输出频谱序列和基4-FFT输出频谱序列结果一致，利用MATLAB自带的函数：

x = [3 2 1 0 1 0 6 3 2 7 0 0 0 0 0 0];

X = fft(x);

disp(X);

% 时域图

n = 0:15;

figure;

stem(n, x);title('输入序列');

% 频域图

figure;

subplot(2,2,1);stem(n,abs(X));title('幅度谱');

subplot(2,2,2);stem(n,real(X));title('频谱实部');

subplot(2,2,3);stem(n,imag(X));title('频谱虚部');

subplot(2,2,4);stem(n,angle(X));title('相角');

|  |  |  |  |
| --- | --- | --- | --- |
| X(0) | X(1) | X(2) | X(3) |
| 25.0000000000000 + 0.00000000000000i | -9.92657016602303 - 5.18438060357565i | 12.4852813742386 + 0.757359312880714i | 1.47406644701202 - 2.10198840328326i |
| X(4) | X(5) | X(6) | X(7) |
| -1.00000000000000 - 6.00000000000000i | 7.59700136485346 + 5.79750653332841i | -4.48528137423857 - 9.24264068711929i | 4.85550235415756 + 6.71511433303601i |
| X(8) | X(9) | X(10) | X(11) |
| 1.00000000000000 + 0.00000000000000i | 4.85550235415756 - 6.71511433303601i | -4.48528137423857 + 9.24264068711929i | 7.59700136485346 - 5.79750653332841i |
| X(12) | X(13) | X(14) | X(15) |
| -1.00000000000000 + 6.00000000000000i | 1.47406644701202 + 2.10198840328326i | 12.4852813742386 - 0.757359312880714i | -9.92657016602303 + 5.18438060357565i |

经验证，基2-FFT输出频谱序列和基4-FFT输出频谱序列计算结果正确。

第9页

与直接计算长序列的离散傅里叶变换(DFT)相比，基4-FFT算法有效地降低了计算复杂度。通过将序列分解为更短的序列并对其应用DFT，该算法显著减少了所需的计算量。为了验证实现的准确性，比较从基4-FFT实现中获得的频谱序列与基2-FFT实现中获得的频谱序列，可以发现两种算法产生相同的结果，证实了基-4 FFT实现的正确性。这个实验加深了我对FFT算法、其基本原理和计算效率的理解。