

# 浙江大学实验报告

专业： 信息工程  
姓名： \_\_\_\_\_  
学号： \_\_\_\_\_  
日期： 2023 年 6 月 4 日  
地点： \_\_\_\_\_

课程名称： 数字信号处理      指导老师： 徐元欣      成绩： \_\_\_\_\_  
实验名称： 基 4-FFT 算法编程      实验类型： 设计      同组学生姓名： \_\_\_\_\_

## 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，使得计算量大大下降。基 4-FFT 是混合基 FFT 的一个特例。

通过编写基 4-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 程序（studentname.m 文件）。

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  $\text{stem}(X)$  显示频谱图形。

## 三、 主要仪器设备

MATLAB 编程。

## 四、 操作方法和实验步骤

（参见“二、实验内容和步骤”）

## 五、 实验数据记录和处理

### 1. 基 4-FFT 算法思路、流程图简述如下

#### 1.1 算法思路

该算法在时域上按  $n$  的特点对序列  $x(n)$  进行不断的以 4 为基数的分组以及位序调整，进而通过逐级的蝶形复合处理，间接地完成高点数 DFT 的计算，由此达到降低运算量以及节省存储空间的目的。

令序列  $x(n)$  的  $N$  点 DFT 结果为  $X(k)$ ，且有  $N = 4^m$ ，按  $((4))_n$  的结果对序列  $x(n)$  分组得：

$$\begin{aligned}x^{(0)}(n) &= x(4n) \\x^{(1)}(n) &= x(4n + 1) \\x^{(2)}(n) &= x(4n + 2) \\x^{(3)}(n) &= x(4n + 3)\end{aligned} \quad 0 \leq n \leq \frac{N}{4} - 1$$

且有：

$$\begin{aligned} X^{(0)}(k) &= \text{DFT}_{4^{m-1}} \{x^{(0)}(n)\} \\ X^{(1)}(k) &= \text{DFT}_{4^{m-1}} \{x^{(1)}(n)\} \\ X^{(2)}(k) &= \text{DFT}_{4^{m-1}} \{x^{(2)}(n)\} \\ X^{(3)}(k) &= \text{DFT}_{4^{m-1}} \{x^{(3)}(n)\} \end{aligned} \quad 0 \leq k \leq N-1 = 4^m - 1$$

令  $0 \leq k \leq 4^{m-1} - 1$ ，则有：

$$\begin{cases} X(k) = X^{(0)}(k) + W_N^k X^{(1)}(k) + W_N^{2k} X^{(2)}(k) + W_N^{3k} X^{(3)}(k) \\ X(k + 4^{m-1}) = X^{(0)}(k) - jW_N^k X^{(1)}(k) - W_N^{2k} X^{(2)}(k) + jW_N^{3k} X^{(3)}(k) \\ X(k + 2 \times 4^{m-1}) = X^{(0)}(k) - W_N^k X^{(1)}(k) + W_N^{2k} X^{(2)}(k) - W_N^{3k} X^{(3)}(k) \\ X(k + 3 \times 4^{m-1}) = X^{(0)}(k) + jW_N^k X^{(1)}(k) - W_N^{2k} X^{(2)}(k) - jW_N^{3k} X^{(3)}(k) \end{cases}$$

## 1.2 流图结构

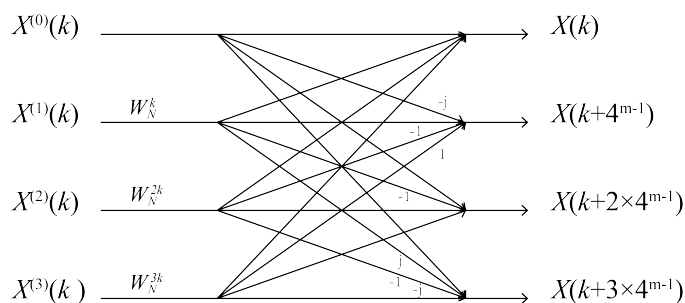


图 1: 基 4 DIT-FFT 中的蝶形运算

对于  $N$  点，基 4DIT-FFT 的整体计算过程如下：

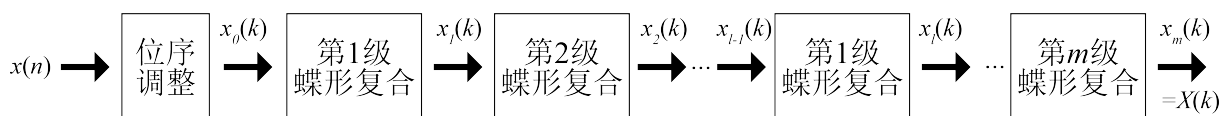


图 2: 基 4 DIT-FFT 计算过程

2. 16 点基 4-FFT 算法的流图绘出如下（因绘图限制，省略系数-1, -j, j, 具体系数对应项见上一蝶形图）

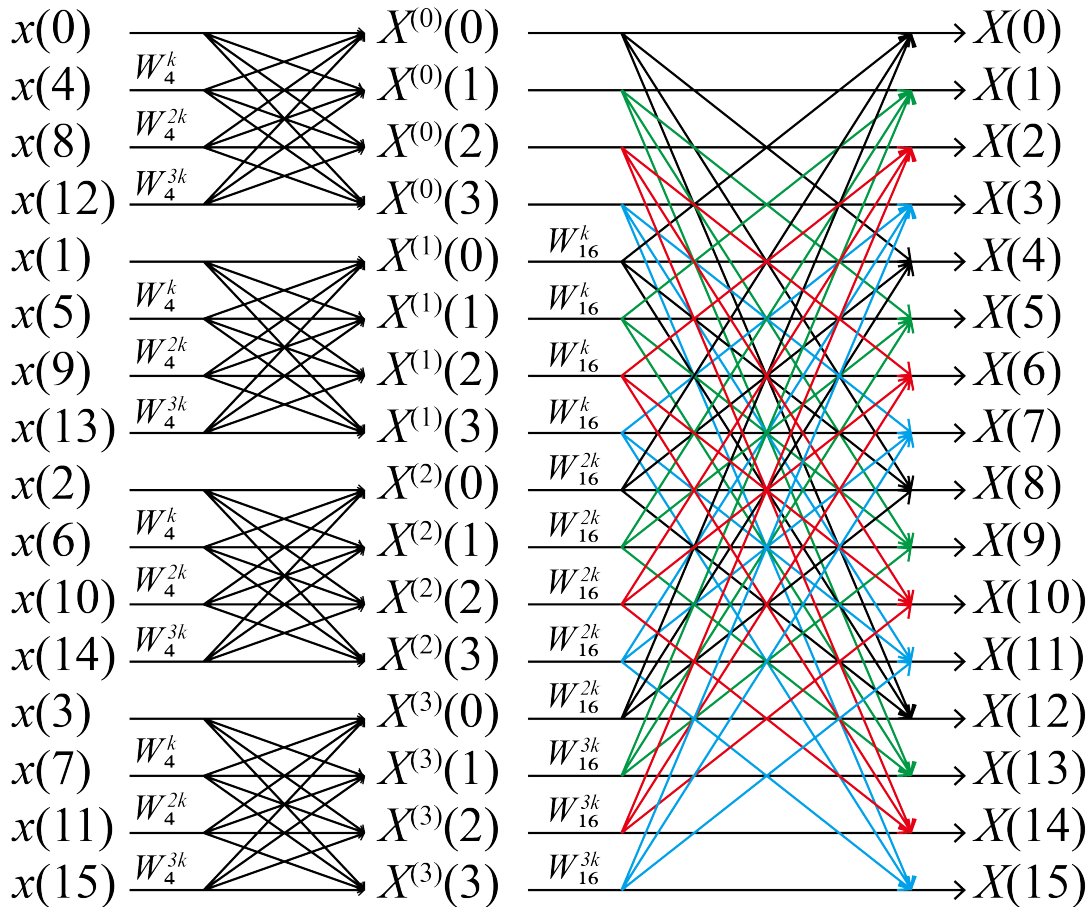


图 3: 16 点基 4-FFT 算法的流图

3. 16 点基 4-FFT 算法的 MATLAB 程序 (studentname.m) 列出如下

studentname.m

```

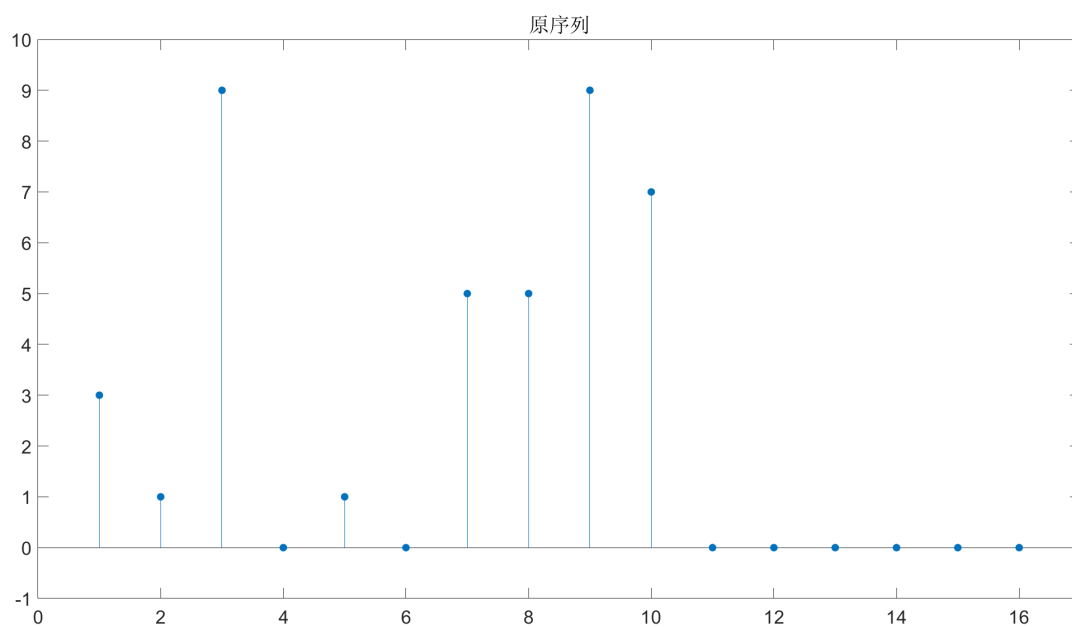
1 % 原始序列
2 xn = [3,1,9,0,1,0,5,5,9,7,0,0,0,0,0,0];
3 n = 1:16;
4
5 % 绘制原始序列
6 f1 = figure(1);
7 set(gcf,'outerposition',get(0,'screensize'));
8 stem(n, real(xn), 'filled');
9 title('原序列');
10 set(gca,'FontSize',16);
11 axis([0 17 -1 10]);
12 saveas(f1, 'exp3_1', 'png');
13 % 基4 DFT

```

```
14 N = 16;
15 W = exp(-1j*2*pi/N);
16 W4 = dftmtx(4);
17
18 % 第一级蝶形运算
19 X1 = W4*[xn(1); xn(5); xn(9); xn(13)];
20 X2 = W4*[xn(2); xn(6); xn(10); xn(14)];
21 X3 = W4*[xn(3); xn(7); xn(11); xn(15)];
22 X4 = W4*[xn(4); xn(8); xn(12); xn(16)];
23
24 % 第二级蝶形运算
25 Xk = zeros(1, 16);
26 for k = 0:3
27     temp = W4*[
28         X1(k + 1);
29         X2(k + 1)*(W^k);
30         X3(k + 1)*(W^(2*k));
31         X4(k + 1)*(W^(3*k))];
32     Xk(k + 1) = temp(1);
33     Xk(k + 5) = temp(2);
34     Xk(k + 9) = temp(3);
35     Xk(k + 13) = temp(4);
36 end
37
38 f2 = figure(2);
39 set(gcf, 'outerposition', get(0, 'screensize'));
40 stem(n, real(Xk), 'filled');
41 title('基4-FFT算法输出频谱实部');
42 set(gca, 'FontSize', 16);
43 axis([0 17 -16 43]);
44 saveas(f2, 'exp3_2', 'png');
45
46 f3 = figure(3);
47 set(gcf, 'outerposition', get(0, 'screensize'));
48 stem(n, imag(Xk), 'filled');
49 title('基4-FFT算法输出频谱虚部');
50 set(gca, 'FontSize', 16);
51 saveas(f3, 'exp3_3', 'png');
52
53 % 对比直接调用FFT函数结果
54 Xk2 = fft(xn);
55 f4 = figure(4);
56 set(gcf, 'outerposition', get(0, 'screensize'));
57 stem(n, real(Xk2), 'filled');
58 title('直接调用FFT函数频谱实部');
59 set(gca, 'FontSize', 16);
60 axis([0 17 -16 43]);
61 saveas(f4, 'exp3_4', 'png');
```

## 4. 用自己的学号构成的输入序列为（列出数值，插入图形）

$$x(n) = [3, 1, 9, 0, 1, 0, 5, 5, 9, 7, 0, 0, 0, 0, 0, 0]$$



## 5. 对应的输出频谱序列为（列出数值，插入图形）

$X_k =$

列 1 至 4

$40.0000 + 0.0000i$  ,  $-13.3342 - 10.5168i$  ,  $20.1924 - 6.1213i$  ,  $-13.0379 - 7.9756i$

列 5 至 8

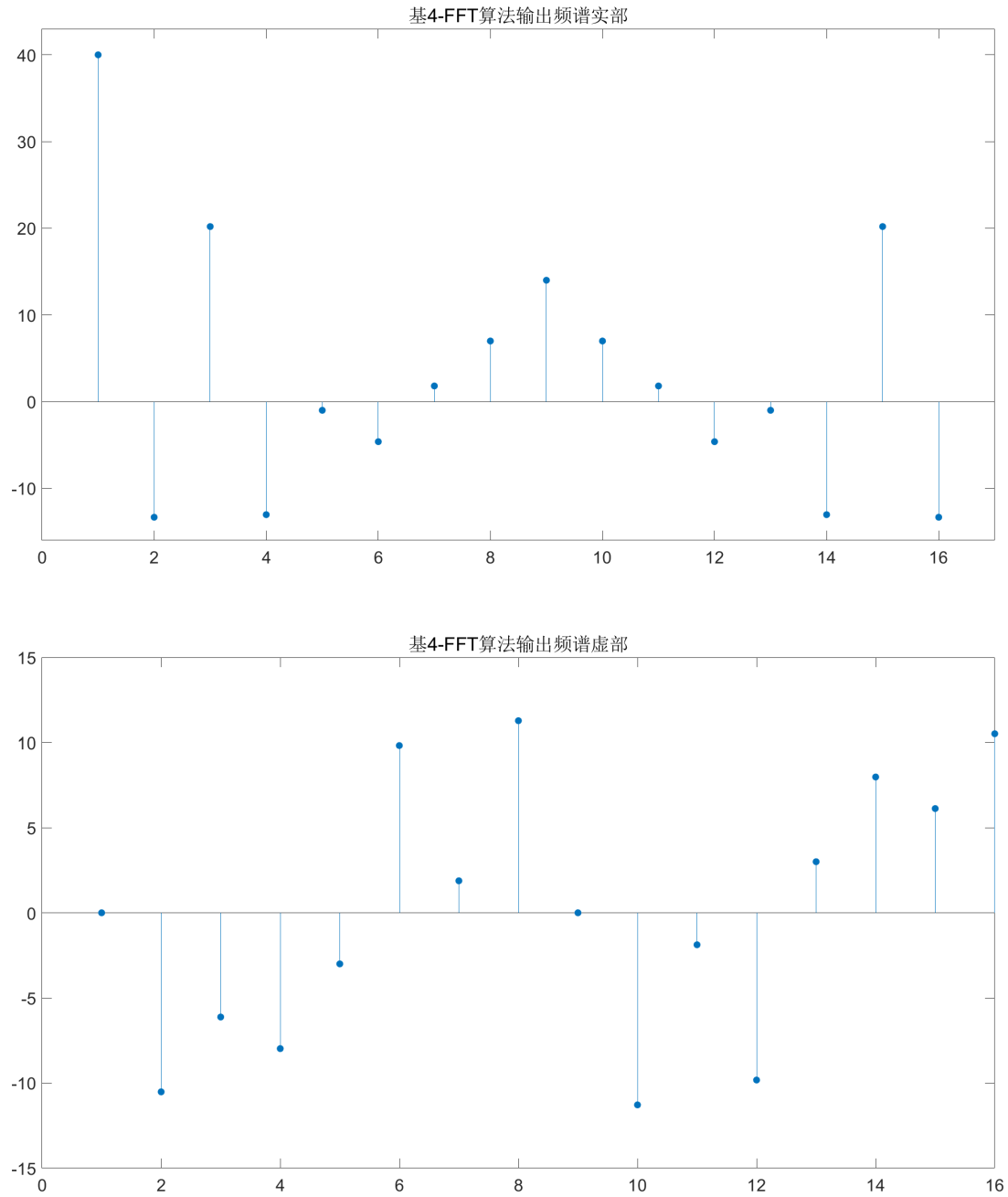
$-1.0000 - 3.0000i$  ,  $-4.6189 + 9.8234i$  ,  $1.8076 + 1.8787i$  ,  $6.9911 + 11.2822i$

列 9 至 12

$14.0000 + 0.0000i$  ,  $6.9911 - 11.2822i$  ,  $1.8076 - 1.8787i$  ,  $-4.6189 - 9.8234i$

列 13 至 16

$-1.0000 + 3.0000i$  ,  $-13.0379 + 7.9756i$  ,  $0.1924 + 6.1213i$  ,  $-13.3342 + 10.5168i$



## 六、 实验结果与分析

为了验证基 4-FFT 算法的正确性，我们直接调用了 Matlab 的 `fft` 函数进行了计算，其结果如下：Xk2 = 列 1 至 4

40.0000 + 0.0000i , -13.3342 - 10.5168i , 20.1924 - 6.1213i , -13.0379 - 7.9756i

列 5 至 8

-1.0000 - 3.0000i , -4.6189 + 9.8234i , 1.8076 + 1.8787i , 6.9911 + 11.2822i

列 9 至 12

$14.0000 + 0.0000i$ ,  $6.9911 - 11.2822i$ ,  $1.8076 - 1.8787i$ ,  $-4.6189 - 9.8234i$

列 13 至 16

$-1.0000 + 3.0000i$ ,  $-13.0379 + 7.9756i$ ,  $20.1924 + 6.1213i$ ,  $-13.3342 + 10.5168i$

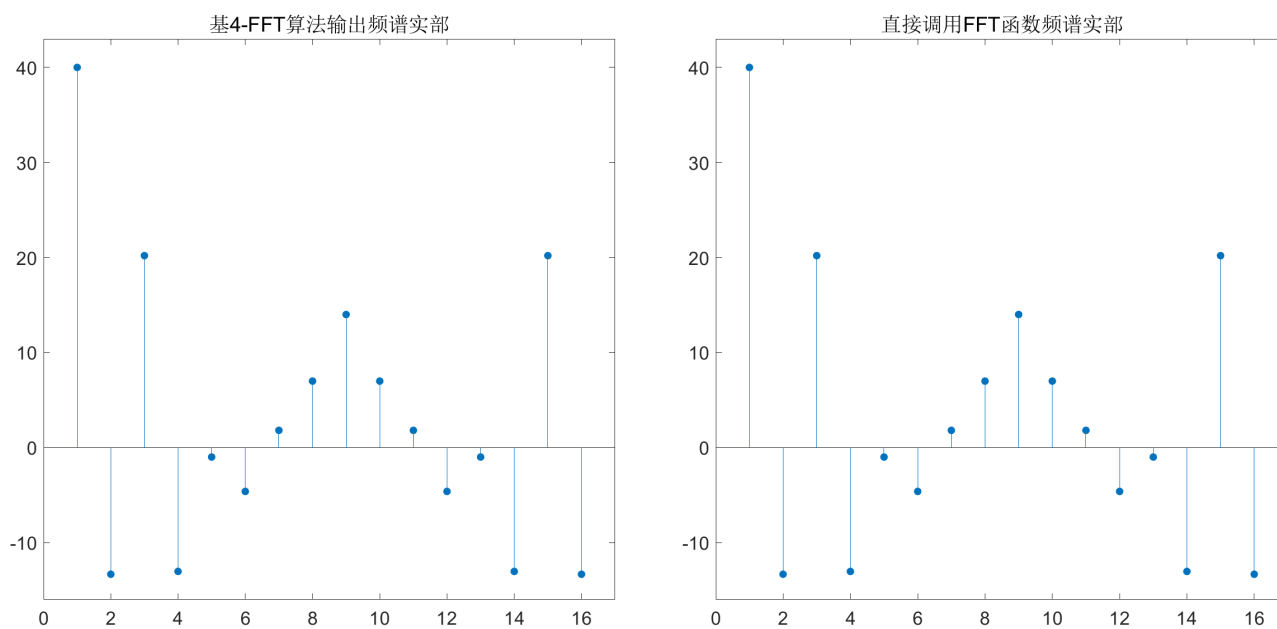


图 4: 对比图

可看出 Matlab 的 `fft` 所得结果和自己的基于 4-FFT 算法所得结果一致，说明自己的结果的正确性。