浙江大学实验报告

专业：信息工程
姓名：姚桂涛
学号：3190105597
日期：2021 年 11 月 2 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 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、流图结构简述如下

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：

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

且有：

$$\begin{aligned} X^{(0)}(k) &= \mathrm{DFT}_{4^{m-1}}\left\{x^{(0)}(n)\right\} \\ X^{(1)}(k) &= \mathrm{DFT}_{4^{m-1}}\left\{x^{(1)}(n)\right\} \\ X^{(2)}(k) &= \mathrm{DFT}_{4^{m-1}}\left\{x^{(2)}(n)\right\} \\ X^{(3)}(k) &= \mathrm{DFT}_{4^{m-1}}\left\{x^{(3)}(n)\right\} \end{aligned} \qquad 0 \leqslant k \leqslant N-1 = 4^m - 1$$

令 $0 \leqslant k \leqslant 4^{m-1}-1$，则有：

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

1.2　流图结构

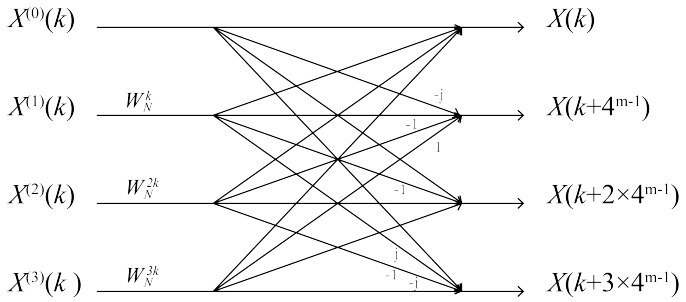


图 1: 基 4 DIT-FFT 中的蝶形运算

对于 N 点，基 4DIT-FFT 的整体计算过程如下：

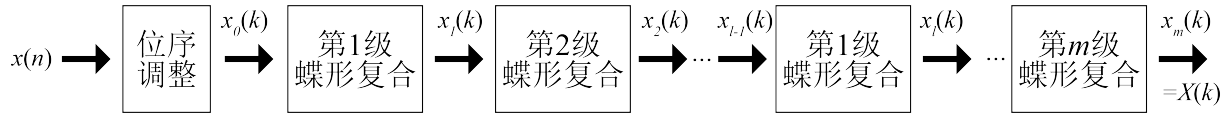


图 2: 基 4 DIT-FFT 计算过程

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）

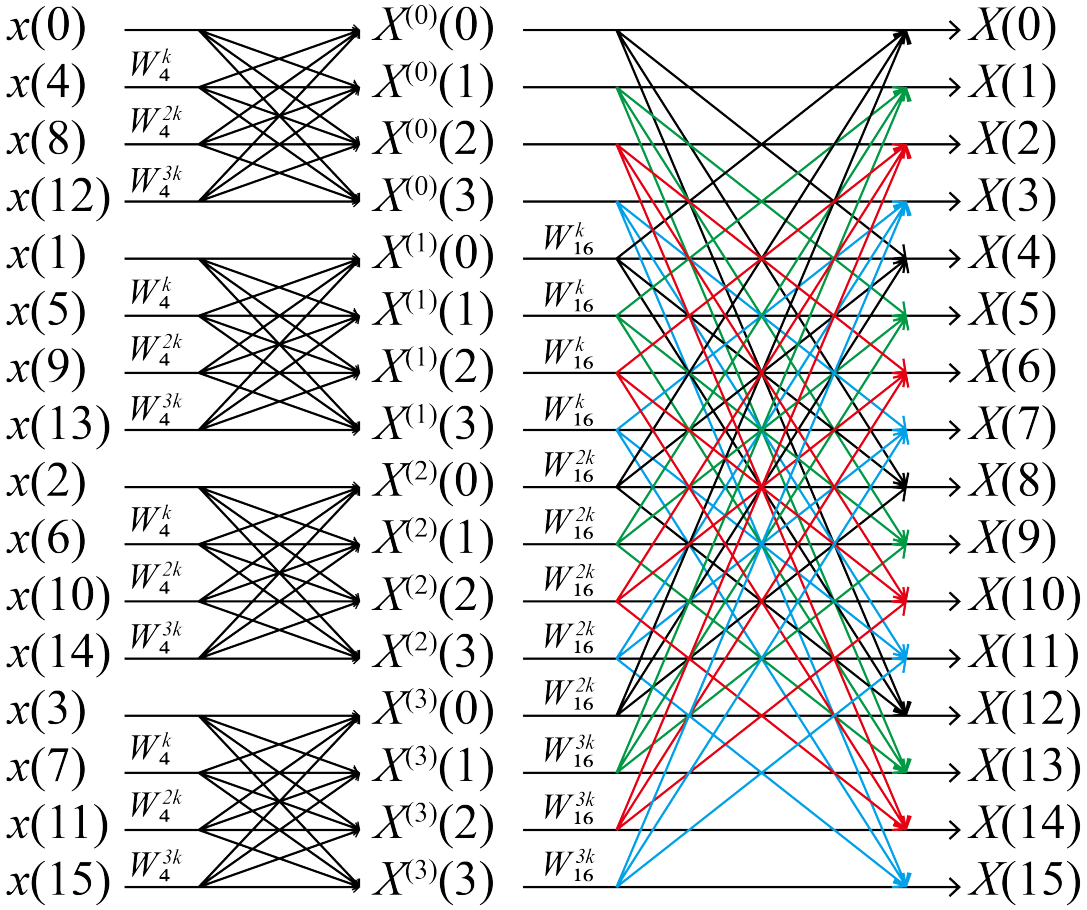


图 3: 16 点基 4-FFT 算法的流图

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

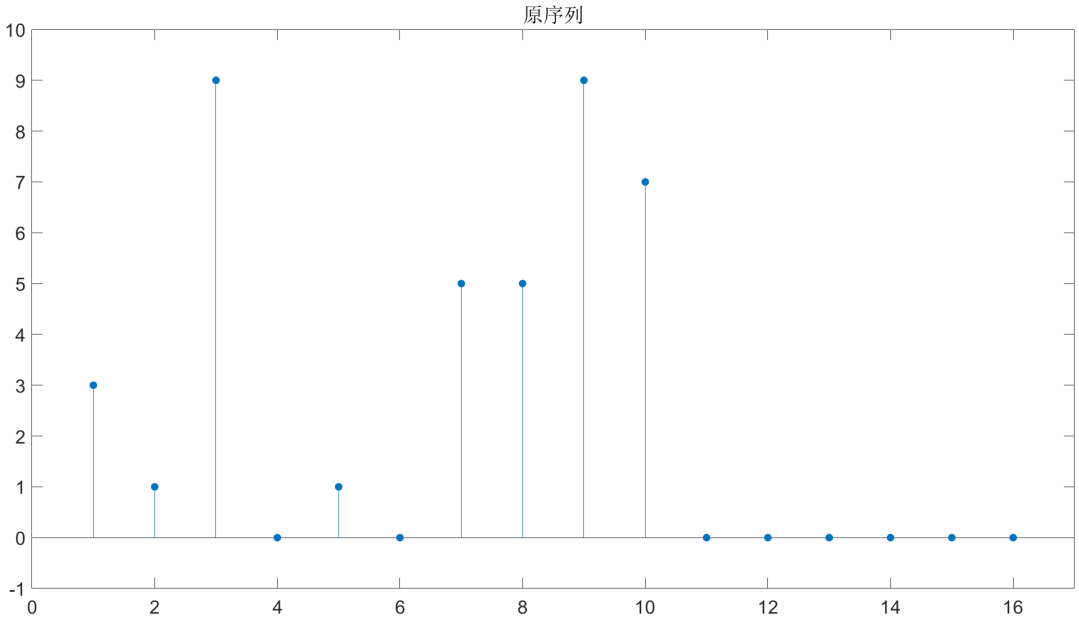
studentname.m

```
% 原始序列
xn = [3,1,9,0,1,0,5,5,9,7,0,0,0,0,0,0];
n = 1:16;

% 绘制原始序列
f1 = figure(1);
    set(gcf,'outerposition',get(0,'screensize'));
    stem(n, real(xn), 'filled');
    title('原序列');
    set(gca,'FontSize',16);
    axis([0 17 -1 10]);
    saveas(f1, 'exp3_1','png');
% 基4 DFT
```

```
N = 16;
W = exp(-1j*2*pi/N);
W4 = dftmtx(4);

% 第一级蝶形运算
X1 = W4*[xn(1); xn(5); xn(9); xn(13)];
X2 = W4*[xn(2); xn(6); xn(10); xn(14)];
X3 = W4*[xn(3); xn(7); xn(11); xn(15)];
X4 = W4*[xn(4); xn(8); xn(12); xn(16)];

% 第二级蝶形运算
Xk = zeros(1, 16);
for k = 0:3
    temp = W4*[
        X1(k + 1);
        X2(k + 1)*(W^k);
        X3(k + 1)*(W^(2*k));
        X4(k + 1)*(W^(3*k))];
    Xk(k + 1) = temp(1);
    Xk(k + 5) = temp(2);
    Xk(k + 9) = temp(3);
    Xk(k + 13) = temp(4);
end

f2 = figure(2);
set(gcf,'outerposition',get(0,'screensize'));
    stem(n, real(Xk), 'filled');
    title('基4—FFT算法输出频谱实部');
    set(gca,'FontSize',16);
    axis([0 17 -16 43]);
    saveas(f2, 'exp3_2','png');

f3 = figure(3);
set(gcf,'outerposition',get(0,'screensize'));
    stem(n, imag(Xk), 'filled');
    title('基4—FFT算法输出频谱虚部');
    set(gca,'FontSize',16);
    saveas(f3, 'exp3_3','png');

% 对比直接调用FFT函数结果
Xk2 = fft(xn);
f4 = figure(4);
set(gcf,'outerposition',get(0,'screensize'));
    stem(n, real(Xk2), 'filled');
    title('直接调用FFT函数频谱实部');
    set(gca,'FontSize',16);
    axis([0 17 -16 43]);
    saveas(f4, 'exp3_4','png');
```

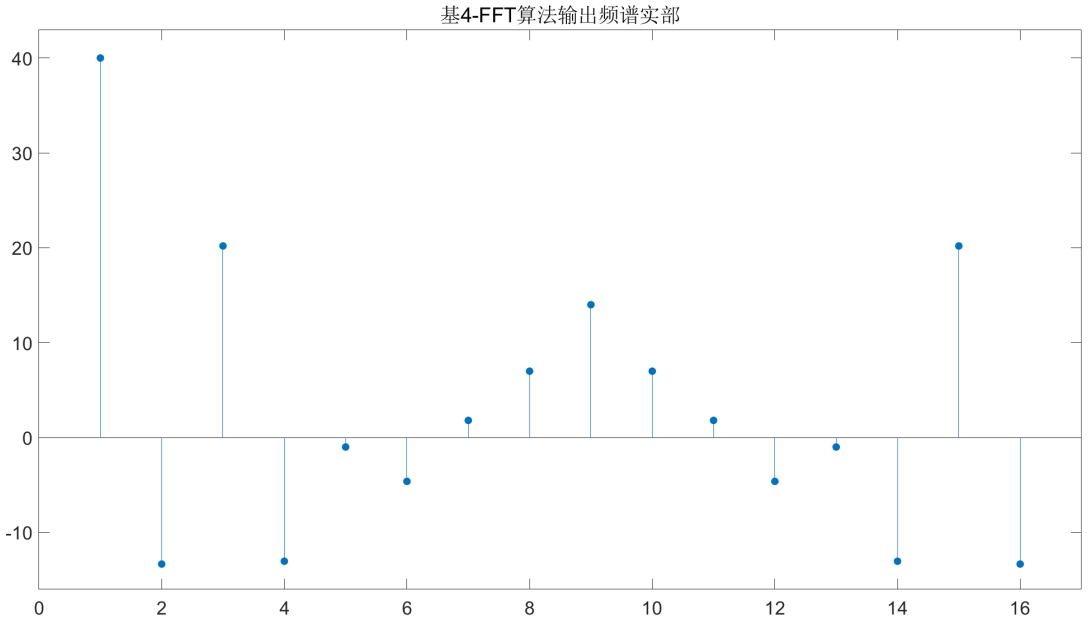
4. 用自己的学号构成的输入序列为（列出数值，插入图形）

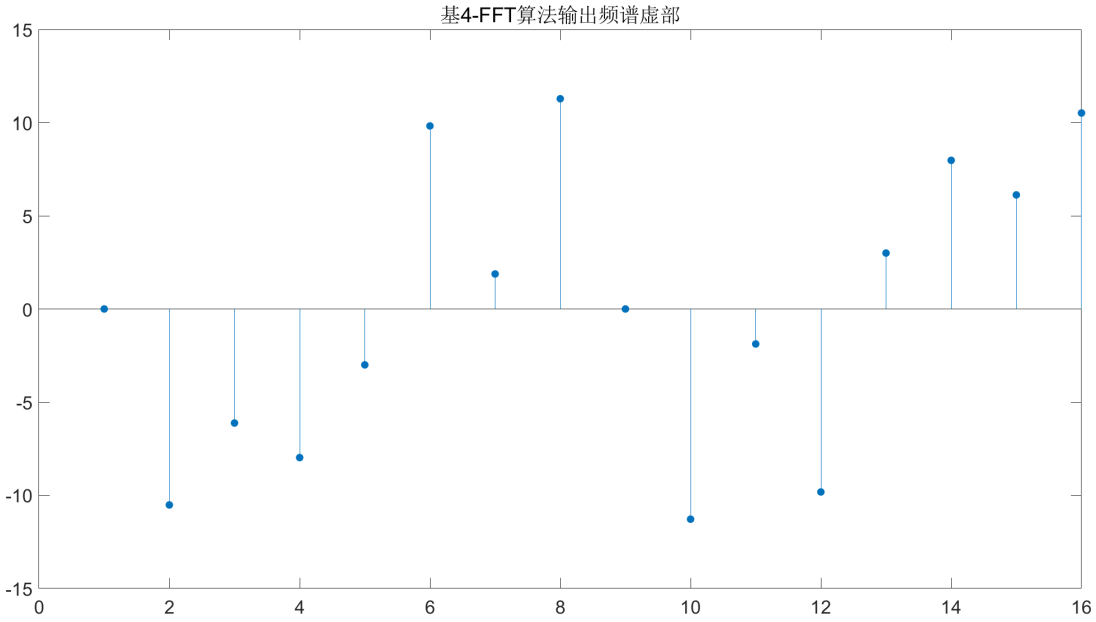
x(n) = [3, 1, 9, 0, 1, 0, 5, 5, 9, 7, 0, 0, 0, 0, 0, 0]



5. 对应的输出频谱序列为（列出数值，插入图形）

Xk =
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

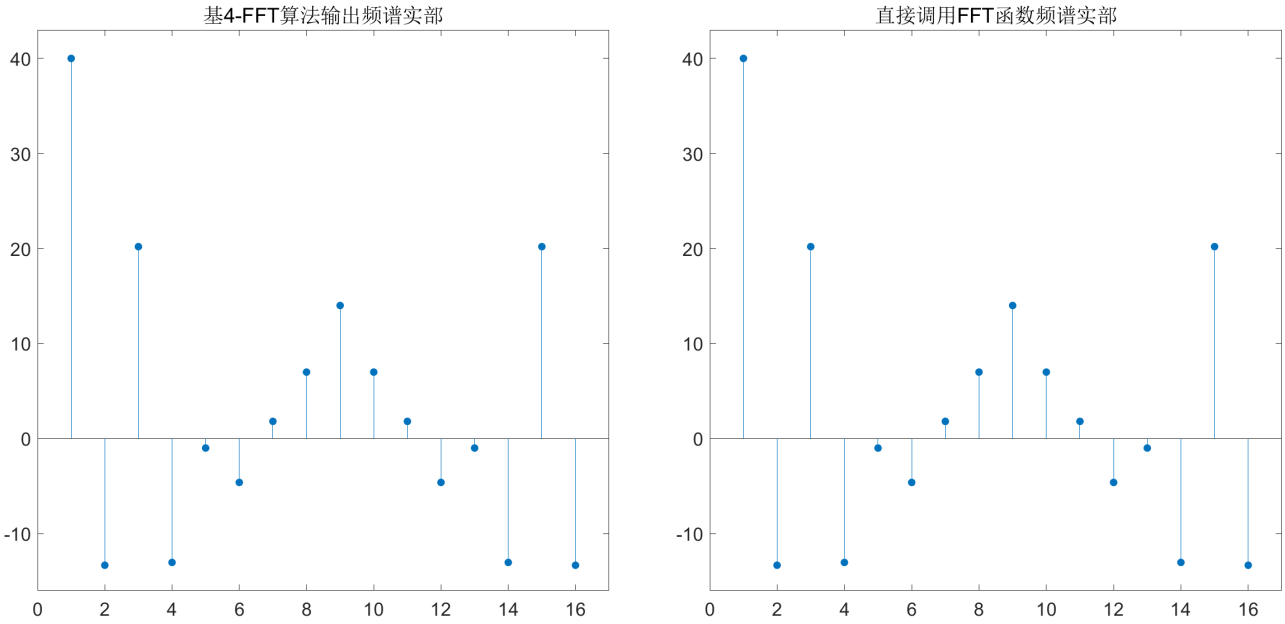


图 4: 对比图

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。