第5章 快速傅里叶变换（FFT）

离散傅里叶变换(DFT)在信号的频谱分析,系统的分析、设计与实现方面起着非常重要的作用,一个关键的原因就是计算 DFT 有许多高效快速的算法,快速傅里叶变换(Fast Fourier Transform,FFT)算法就是其中之一。需要强调的是,FFT 并不是一种新的变换算法,只是 DFT 的一种快速实现算法。

由于直接计算 DFT 的运算量太大,很多年一直没有得到实际应用。直到 1965 年,库利(J. W. Cooley)和图基(J. W. Tukey)在论文 *An Algorithm for the Machine Calculation of Complex Fourier Series* 中提出快速算法 FFT,使得 DFT 的运算大为简化,从而使 DFT 真正得到广泛应用。FFT 算法的横空出世,破解了 DFT 运算量偏大的历史性难题,促进了数字信号处理突飞猛进的发展,因此往往把 1965 年视作数字信号处理元年。

本章首先对 DFT 运算复杂度进行分析,找出 DFT 运算的瓶颈所在和提速的可能性,随后介绍将 DFT 在时域逐渐分解为较小点数 DFT 运算的方法,即按时间抽取(Decimation in Time)的 FFT 算法,在 DIT-FFT 算法的基础上介绍按频率抽取(Decimation in Frequency)的 FFT 算法,最后介绍 FFT 算法在工程实现中的一些经验技巧。

5.1 DFT 运算复杂度分析

本节主要用乘法和加法次数来衡量 DFT 的运算复杂度,这是因为在计算机或者 DSP 芯片中,乘法和加法次数直接决定了算法执行效率和速度。

5.1.1 DFT 的运算瓶颈

设 $x(n)$为 N 点有限长序列,其 N 点的 DFT 正变换为

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk},\quad k=0,1,\cdots,N-1 \tag{5.1.1}$$

式中,$W_N^{nk}\triangleq \mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}nk}$ 表示旋转因子。

DFT 反变换为

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-nk},\quad n=0,1,\cdots,N-1 \tag{5.1.2}$$

从 DFT 正变换和反变换公式可以看出,二者差别在于旋转因子指数符号相反,以及差一个常数因子 $1/N$。因此 DFT 正变换和反变换运算量是完全相同的,只需讨论 DFT 正变换的运算量即可。

从 DFT 正变换公式可以看出,每计算一个 $X(k)$,需要进行 N 次复数乘法和 $N-1$ 次复数加法。一共有 N 个 $X(k)$需要计算,就需要 N^2 次复数乘法和 $N(N-1)$次复数加法,这就是完成整个 DFT 正变换需要的计算量。

当 $N\gg 1$ 时,直接计算 DFT,复数乘法次数和复数加法次数都与 N^2 成正比。随着

N 的增大，运算次数会急剧增加。例如，当 $N=8$ 时，需要 64 次复数乘法，当 $N=1024$ 时，就需要 1 048 576 次复数乘法，复数乘法次数达到了百万量级。如果信号要求实时处理，对运算速度的要求实在太高了。

5.1.2 DFT 的运算特点

为了提高 DFT 的运算效率，研究人员对其进行了大量研究，发现在式(5.1.1)中其实存在大量的冗余运算，并且这些冗余运算主要来自于旋转因子的反复相乘。

假定 $x(n)$ 为 4 点的有限长序列，其 4 点的 DFT 结果可以写为

$$\begin{aligned}
X(0)&=x(0)W_4^0+x(1)W_4^0+x(2)W_4^0+x(3)W_4^0\\
X(1)&=x(0)W_4^0+x(1)W_4^1+x(2)W_4^2+x(3)W_4^3\\
X(2)&=x(0)W_4^0+x(1)W_4^2+x(2)W_4^4+x(3)W_4^6\\
X(3)&=x(0)W_4^0+x(1)W_4^3+x(2)W_4^6+x(3)W_4^9
\end{aligned}\tag{5.1.3}$$

如果严格按照式(5.1.1)的定义进行运算，则需要 16 次复数乘法和 12 次复数加法。

旋转因子 W_N^{nk} 具有以下 4 个特性：

共轭对称性 $$(W_N^{nk})^*=W_N^{-nk} \tag{5.1.4}$$

周期性 $$W_N^{nk}=W_N^{(N+n)k}=W_N^{n(N+k)} \tag{5.1.5}$$

可约性 $$W_N^{nk}=W_{mN}^{mnk}=W_{N/m}^{nk/m} \tag{5.1.6}$$

特殊值 $$W_N^0=1,\quad W_N^{N/2}=-1,\quad W_N^{N/4}=-\mathrm{j},\quad W_N^{k+N/2}=-W_N^k \tag{5.1.7}$$

如果充分利用旋转因子的这 4 个特点，则式(5.1.7)可化解为

$$\begin{aligned}
X(0)&=[x(0)+x(2)]+[x(1)+x(3)]\\
X(1)&=[x(0)-x(2)]+[x(1)-x(3)]W_4^1\\
X(2)&=[x(0)+x(2)]-[x(1)+x(3)]\\
X(3)&=[x(0)-x(2)]-[x(1)-x(3)]W_4^1
\end{aligned}\tag{5.1.8}$$

从式(5.1.8)可以看出，计算 4 点的 DFT 结果，此时仅需要 1 次复数乘法和 8 次复数加法，相比按照定义计算的运算量明显降低。

根据式(5.1.1)的定义，DFT 的运算量与 N^2 呈正比，显然 N 越小越有利，因此很自然就希望将大点数的 DFT 分解为若干个小点数的 DFT 来计算。在这种分解过程中，充分利用旋转因子的 4 个特性，为提高 DFT 运算速度提供了可能性，FFT 算法就是基于这种基本思路发展起来的。

为了能够不断地进行分解，FFT 算法要求 DFT 的运算点数 $N=2^L$，L 为正整数。这种 N 为 2 的整数次幂的 FFT 算法，称作基-2 FFT 算法。按照运算特点，基-2 FFT 算法可分为按时间抽取(DIT)和按频率抽取(DIF)两种方案。

5.2 按时间抽取(DIT)的 FFT 算法

5.2.1 算法原理

设 $x(n)$ 为 N 点长序列，$N=2^L$，L 为正整数。按照 n 的奇偶取值将 $x(n)$ 分为两组：

$$\begin{cases} x_0(r)=x(2r) \\ x_1(r)=x(2r+1) \end{cases} \tag{5.2.1}$$

其中 $r=0,1,\cdots,N/2-1$。

则序列 $x(n)$ 的 N 点 DFT 结果为

$$\begin{aligned} X(k)&=\sum_{n=0}^{N-1}x(n)W_N^{nk} \\ &=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2rk}+\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_N^{(2r+1)k} \\ &=\sum_{r=0}^{\frac{N}{2}-1}x_0(r)W_N^{2rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_N^{2rk} \end{aligned} \tag{5.2.2}$$

根据旋转因子的可约性，即 $W_N^{2rk}=W_{N/2}^{rk}$，可得

$$\begin{aligned} X(k)&=\sum_{r=0}^{\frac{N}{2}-1}x_0(r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{rk} \\ &=X_0(k)+W_N^kX_1(k) \end{aligned} \tag{5.2.3}$$

其中，k 的取值范围为 $0\leqslant k\leqslant N/2-1$，$X_0(k)$ 和 $X_1(k)$ 分别是 $x_0(r)$ 和 $x_1(r)$ 的 $N/2$ 点 DFT 结果。

从式(5.2.3)的结果可以看出，一个 N 点的 DFT 可以由两个 $N/2$ 点的 DFT 结果组合得到，但这种方法只得到 $X(k)$ 的前一半结果。要想得到 $X(k)$ 的后一半结果，还需用到旋转因子的周期特性，即 $W_{N/2}^{rk}=W_{N/2}^{r(k+N/2)}$，这样可以得到

$$X_0\left(k+\frac{N}{2}\right)=\sum_{r=0}^{\frac{N}{2}-1}x_0(r)W_{N/2}^{r(k+N/2)}=\sum_{r=0}^{\frac{N}{2}-1}x_0(r)W_{N/2}^{rk}=X_0(k) \tag{5.2.4}$$

同理可得，

$$X_1\left(k+\frac{N}{2}\right)=X_1(k) \tag{5.2.5}$$

式(5.2.4)和式(5.2.5)说明，后半部分 k 值($N/2\leqslant k+N/2\leqslant N-1$)对应的 $X_0(k)$ 和 $X_1(k)$，分别与前半部分 k 值($0\leqslant k\leqslant N/2-1$)对应的 $X_0(k)$ 和 $X_1(k)$ 相等。

结合式(5.2.3)、式(5.2.4)和式(5.2.5)的结论，再利用旋转因子的特殊值，即 $W_N^{k+N/2}=W_N^{N/2}W_N^k=-W_N^k$，就可得出后半部分的 $X(k)$，即

$$X\left(k+\frac{N}{2}\right)=X_0(k)-W_N^kX_1(k) \tag{5.2.6}$$

从式(5.2.3)和式(5.2.6)可以看出，只需要计算前半部分的 $X_0(k)$ 和 $X_1(k)$，就可以得到全部范围内的 $X(k)$，这样就大大降低了运算量。

可用图 5.2.1 来表示式(5.2.3)和式(5.2.6)的运算流图，因为该流图形如蝴蝶，故常称作“蝶形运算”。一个蝶形运算单元需要 1 次复数乘法和 2 次复数加法。如果支路上没有标出系数，则该支路系数默认为 1。

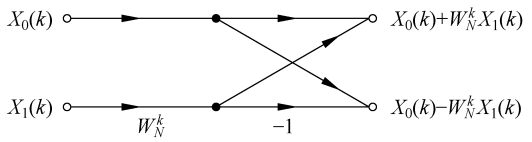

图 5.2.1　蝶形运算单元

图 5.2.2 给出了将 N 点 DFT 分解为两个 $N/2$ 点 DFT 的蝶形运算流图，以 $N=8$ 为例。

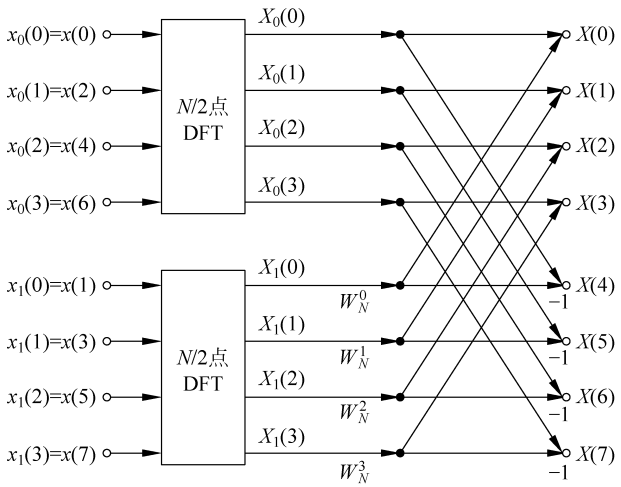

图 5.2.2　按时间抽取，将 N 点 DFT 分解为两个 $N/2$ 点 DFT($N=8$)

由 5.1.1 节的分析可知，直接计算一个 $N/2$ 点的 DFT 运算，需要 $\left(\frac{N}{2}\right)^2$ 次复数乘法和 $\left(\frac{N}{2}\right)\left(\frac{N}{2}-1\right)$ 次复数加法，那么计算两个 $N/2$ 点的 DFT 运算，则需要 $2\times\left(\frac{N}{2}\right)^2=\frac{N^2}{2}$ 次复数乘法和 $N\left(\frac{N}{2}-1\right)$ 次复数加法。此外，根据图 5.2.2，把两个 $N/2$ 点的 DFT 结果合成一个 N 点的 DFT 结果，还需要 $N/2$ 次蝶形运算，即还需要 $\frac{N}{2}$ 次复数乘法和 $2\times\frac{N}{2}=N$ 次复数加法。因此，按照图 5.2.2 的分解方式，总共需要 $\frac{N^2}{2}+\frac{N}{2}$ 次复数乘法和

$N\left(\frac{N}{2}-1\right)+N=\frac{N^2}{2}$次复数加法，运算量相比直接计算 N 点 DFT 减少将近一半。

因为 $N=2^L$，故 $N/2$ 仍是偶数，很自然地就想到还可以把每个 $N/2$ 点 DFT 分解为两个 $N/4$ 点 DFT 来计算，分解和组合的方式与前面介绍的思路完全一致。

按照 r 的奇偶取值将 $x_0(r)$分为两组：

$$\begin{cases} x_2(m)=x_0(2m) \\ x_3(m)=x_0(2m+1) \end{cases} \tag{5.2.7}$$

其中 $m=0,1,\cdots,N/4-1$。

同样可以得到，

$$\begin{aligned} X_0(k)&=\sum_{m=0}^{\frac{N}{4}-1}x_0(2m)W_{N/2}^{2mk}+\sum_{m=0}^{\frac{N}{4}-1}x_0(2m+1)W_{N/2}^{(2m+1)k} \\ &=\sum_{m=0}^{\frac{N}{4}-1}x_2(m)W_{N/4}^{mk}+W_{N/2}^{k}\sum_{m=0}^{\frac{N}{4}-1}x_3(m)W_{N/4}^{mk} \\ &=X_2(k)+W_{N/2}^{k}X_3(k) \end{aligned} \tag{5.2.8}$$

其中 k 的取值范围为 $0\leqslant k\leqslant N/4-1$，$X_2(k)$和 $X_3(k)$分别是 $x_2(m)$和 $x_3(m)$的 $N/4$ 点 DFT 结果。

同理可得后半部分的 $X_0(k)$，即 $N/4\leqslant k+N/4\leqslant N/2-1$，

$$X_0\left(k+\frac{N}{4}\right)=X_2(k)-W_{N/2}^{k}X_3(k) \tag{5.2.9}$$

图 5.2.3 给出 $N=8$ 时，将一个 $N/2$ 点 DFT 分解为两个 $N/4$ 点 DFT 的蝶形运算流图。

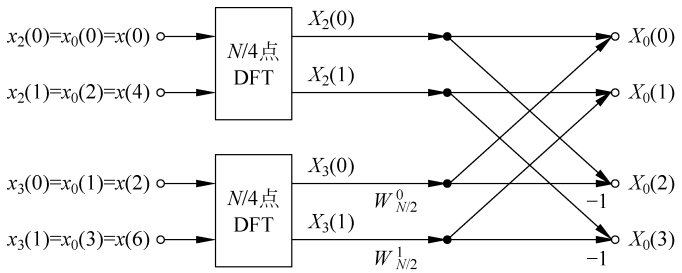

图 5.2.3　按时间抽取，将 $N/2$ 点 DFT 分解为两个 $N/4$ 点 DFT($N=8$)

对 $x_1(r)$也按照 r 的奇偶取值分为两组，

$$\begin{cases} x_4(m)=x_1(2m) \\ x_5(m)=x_1(2m+1) \end{cases} \tag{5.2.10}$$

同理得到

$$X_1(k)=X_4(k)+W_{N/2}^{k}X_5(k) \tag{5.2.11}$$

$$X_1\left(k+\frac{N}{4}\right)=X_4(k)-W_{N/2}^{k}X_5(k) \tag{5.2.12}$$

其中 k 的取值范围仍然为 $0 \leqslant k \leqslant N/4-1$，$X_4(k)$ 和 $X_5(k)$ 分别是 $x_4(m)$ 和 $x_5(m)$ 的 $N/4$ 点 DFT 结果。

根据旋转因子的可约性，将系数统一为 $W_{N/2}^{k}=W_N^{2k}$，则 $N=8$ 点的 DFT 可分解为 4 个 $N/4$ 点的 DFT，如图 5.2.4 所示。

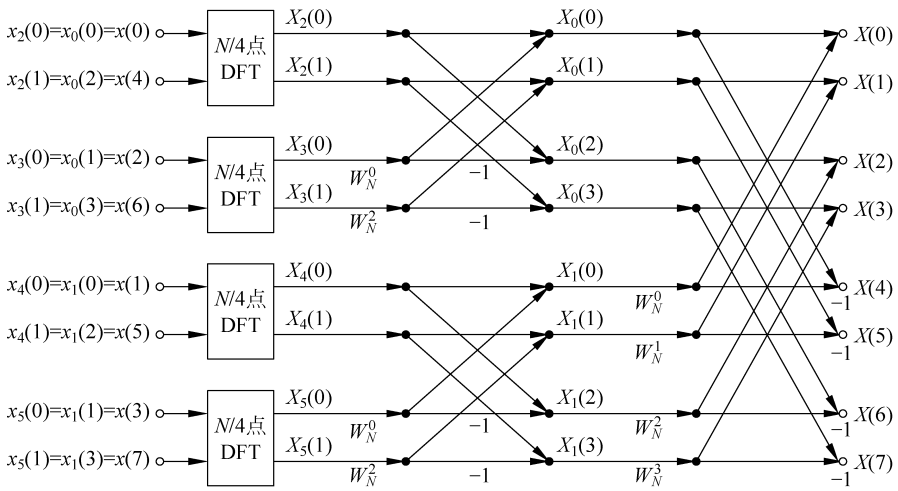

图 5.2.4　按时间抽取，将 N 点 DFT 分解为 4 个 $N/4$ 点 DFT（$N=8$）

同理可知，用 4 个 $N/4$ 点的 DFT 来计算 N 点的 DFT，相比分解成两个 $N/2$ 点 DFT 的方式，运算量又减少将近一半。

如此不断分解下去，一直分解到基本运算单元为 2 点 DFT 为止。对于 $N=8$，图 5.2.4 已经是“分解到底”的情况，此时将“$N/4$ 点 DFT”模块用蝶形运算单元来代替（图 5.2.1），可得完整的 8 点 DIT-FFT 流图，如图 5.2.5 所示。

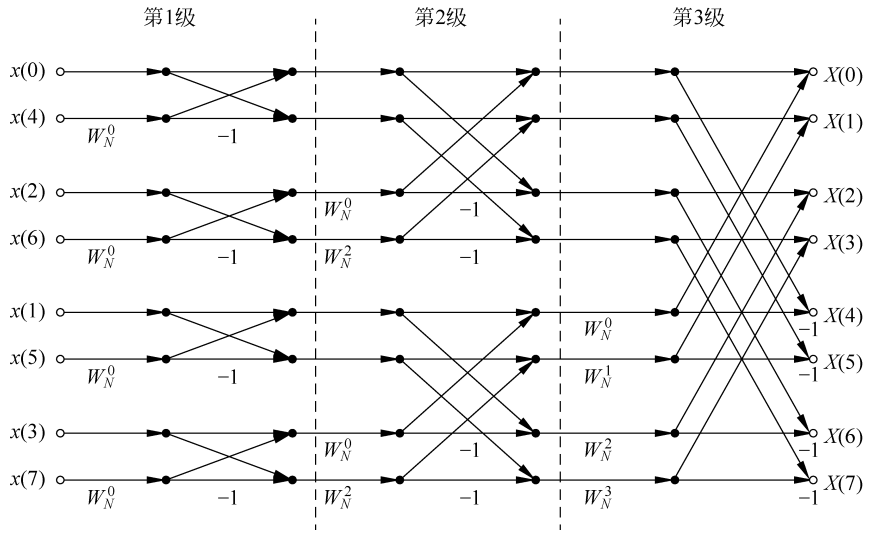

图 5.2.5　$N=8$ 点的基-2 DIT-FFT 流图

由于每一步分解都是按照输入序列的奇偶顺序进行分组的，这就是该方法称作“按时间抽取”的原因。

5.2.2 算法特点

从前面的分析可以看出，基-2 DIT-FFT 算法的推导过程虽然看起来有些复杂，但实现起来却非常有规律，主要有以下这些特点。

1. 同址运算

同址运算也称作原位运算，是指输入和输出在运算过程中占用相同的存储地址，从而可以节约存储空间，降低硬件成本，FFT 运算就是典型的同址运算。

从图 5.2.5 可以看出，FFT 运算由多级蝶形运算构成。若最初的输入数据保存在 $A_0 \sim A_{N-1}$ 的 N 个存储单元中，第 1 级运算完成之后的结果仍可以保存在 $A_0 \sim A_{N-1}$ 存储单元，原来保存在同样位置的输入数据被覆盖，相当于对数据进行了迭代更新。第 1 级的输出为第 2 级的输入，以此类推，最终的 FFT 结果也保存在 $A_0 \sim A_{N-1}$ 存储单元。也就是说，中间过程的所有结果都没有（不需要）被保存，存储数据空间始终为 N 个存储单元。

此外，还需要 $N/2$ 个存储单元用于存放蝶形运算支路的系数，这些系数随着蝶形运算的迭代更新即可，也不需要额外的存储单元。

2. 输入倒序，输出顺序

从图 5.2.5 可以看出，输入数据是按照 $x(0)$、$x(4)$、$x(2)$、$x(6)$、$x(1)$、$x(5)$、$x(3)$、$x(7)$送入 DIT-FFT 流图进行后续运算的。乍一看，这一顺序杂乱无章，其实是很有规律的，对应关系就是二进制的倒位序关系。比如十进制数 $n=6$，用二进制表示为 110，将二进制的 110 高低位顺序反转得到 011，二进制的 011 转换为十进制为 $n=3$，这意味着 $x(6)$需要调整到 $x(3)$的位置上去。通过这种变换关系，$x(3)$也需要调整到 $x(6)$的位置上去，因此调整输入数据位置的过程是成对互换，并不需要额外的存储单元。

这种位置互换关系的根本原因在于“每一步分解都是按照输入序列的奇偶顺序进行分组的”，每次分组都是将二进制最低位进行了移动，分到最后刚好将所有位置全部进行了反转。按位反转在计算机中很容易实现，这也是基-2 FFT 算法受到广泛欢迎的重要原因之一。

表 5.1 以 $N=8$ 为例，给出了输入数据调整位置前后的对应关系。对于一般情况 $N=2^L$，调整位置的过程是完全一致的。

表 5.1 自然顺序与倒位序的对应关系（$N=8$）

自然顺序（十进制）	自然顺序（二进制）	倒位序（二进制）	倒位序（十进制）
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

3. 蝶形运算

从图 5.2.5 可以看出，对于 $N=2^L$ 点长序列，其 DIT-FFT 流图中共有 L 级蝶形运算，每级中都有 $N/2$ 个蝶形运算单元。如果序列长度不是 2 的整数次幂，就补零达到这个要求。

在 DIT-FFT 流图中，蝶形运算类型（即支路系数和数据点间隔）随着迭代次数成倍增加。以 $N=8$ 为例，在第 1 级蝶形运算中，只有一种支路系数，即 W_8^0，并且参与蝶形运算的两个数据点间隔为 1。在第 2 级蝶形运算中，有两种支路系数，即 W_8^0 和 W_8^2，并且参与蝶形运算的两个数据点间隔为 2。在第 3 级蝶形运算中，有四种支路系数，即 W_8^0、W_8^1、W_8^2 和 W_8^3，并且参与蝶形运算的两个数据点间隔为 4。

每一级蝶形运算的上半部分没有旋转因子，只有下半部分有旋转因子，并且这些旋转因子的取值也是非常有规律的。旋转因子 W_N^r 完全由变量 r 决定，而 r 的个数和取值完全由蝶形运算的级数 i 决定，其中 $i=1,2,\cdots,L$，$N=2^L$。r 的个数为 2^{i-1}，取值为 $r=\frac{N}{2^i}p$，$p=0,1,\cdots,(2^{i-1}-1)$，每级旋转因子的具体取值如表 5.2 所示。

表 5.2 DIT-FFT 流图中旋转因子 W_N^r 的取值规律

第 i 级	r 个数	r 取值	旋转因子作用
$i=1$	1 个	$r=0$	2 点 DFT
$i=2$	2 个	$r=0,\frac{N}{4}$	4 点→2 点
$i=3$	4 个	$r=0,\frac{N}{8},\frac{2N}{8},\frac{3N}{8}$	8 点→4 点
…	…	…	…
$i=L$	2^{L-1} 个	$r=0,\frac{N}{2^L},\cdots,\frac{(2^{L-1}-2)N}{2^L},\frac{(2^{L-1}-1)N}{2^L}$	N 点→$N/2$ 点

5.2.3 运算量分析

对于 $N=2^L$ 点长序列，共有 L 级蝶形运算，每级中都有 $N/2$ 个蝶形运算单元，故 DIT-FFT 流图中共有 $\frac{N}{2}L$ 个蝶形运算单元。每个蝶形运算单元需要 1 次复数乘法和 2 次复数加法，因此，实现全部 N 点的 FFT 需要 $\frac{N}{2}L=\frac{N}{2}\log_2 N$ 次复数乘法和 $2\ \frac{N}{2}L=N\log_2 N$ 次复数加法。

从 5.1.1 节的分析可知，直接计算 N 点的 DFT，需要 N^2 次复数乘法和 $N(N-1)$ 次复数加法。由于计算机中，乘法运算更耗时间，因此主要考虑复数乘法次数，表 5.3 将直接计算 DFT 和 FFT 的运算量进行了对比。可以看出，与直接计算 DFT 相比，FFT 算法在运算量上有数量级的下降，并且 N 越大，FFT 算法优势越明显。

表 5.3 直接计算 DFT 和 FFT 算法的复数乘法次数对比

N	直接计算 DFT	FFT 算法
2	4	1
4	16	4
32	1024	80
128	16384	448
256	65536	1024
512	262144	2304
1024	1048576	5120
2048	4194304	11264
4096	16777216	24576

例 5.1 利用 DIT-FFT 流图，计算 $x(n)=\{1,3,5,2\}_0$ 的 DFT 结果。

解：$N=4$ 点的 DIT-FFT 运算流图如图 5.2.6 所示。

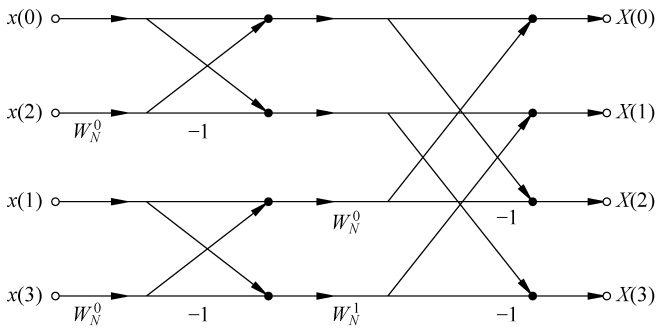

图 5.2.6 $N=4$ 点 DIT-FFT 流图

代入 $x(n)=\{1,3,5,2\}_0$，以及 $W_N^0=1, W_N^1=W_4^1=-\mathrm{j}$，可得运算过程如图 5.2.7 所示。

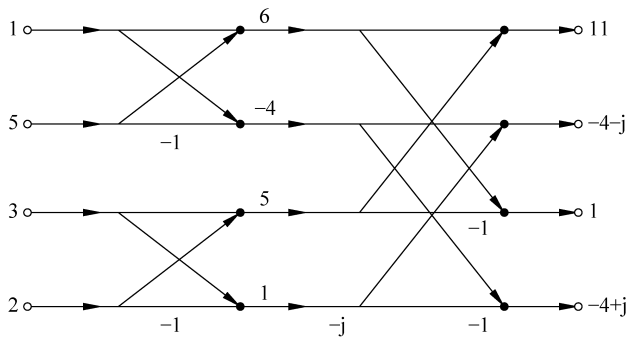

图 5.2.7　例题 5.1 运算过程

从图 5.2.7 的输出结果可以看出，DFT 结果为 $X(k)=\{11,-4-\mathrm{j},1,-4+\mathrm{j}\}_0$。

5.3　按频率抽取(DIF)的 FFT 算法

对于 $N=2^L$ 的情况，另外一种常用的 FFT 算法是按照输出序列 $X(k)$ 的奇偶顺序进行分解的，称作按频率抽取(Decimation in Frequency)的 FFT 算法。

先把输入序列 $x(n)$ 按照前后对半分开，则 N 点的 DFT 可以写成前后两部分

$$\begin{aligned}X(k)&=\sum_{n=0}^{N-1}x(n)W_N^{nk}\\&=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{nk}+\sum_{n=\frac{N}{2}}^{N-1}x(n)W_N^{nk}\\&=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{nk}+\sum_{n=0}^{\frac{N}{2}-1}x\left(n+\frac{N}{2}\right)W_N^{\left(n+\frac{N}{2}\right)k}\\&=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{nk}+W_N^{\frac{N}{2}k}\sum_{n=0}^{\frac{N}{2}-1}x\left(n+\frac{N}{2}\right)W_N^{nk}\end{aligned}\tag{5.3.1}$$

其中 k 的取值范围为 $0\leqslant k\leqslant N-1$，且 $W_N^{\frac{N}{2}k}=(W_N^{\frac{N}{2}})^k=(-1)^k$。

式(5.3.1)右边并不是两个 $N/2$ 点 DFT 之和的形式，因为旋转因子是 W_N^{nk}，而不是 $W_{N/2}^{nk}$，继续整理可得

$$X(k)=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+(-1)^k x\left(n+\frac{N}{2}\right)\right]W_N^{nk}\tag{5.3.2}$$

根据 k 的奇偶取值，可以把 $X(k)$ 分成两部分，

$$\begin{aligned} X(2r) &= \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_N^{2rn} \\ &= \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_{N/2}^{nr} \end{aligned} \tag{5.3.3}$$

$$\begin{aligned} X(2r+1) &= \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^{(2r+1)n} \\ &= \sum_{n=0}^{\frac{N}{2}-1}\left\{\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^{n}\right\}W_{N/2}^{nr} \end{aligned} \tag{5.3.4}$$

其中 r 的取值范围为 $0\leqslant r\leqslant N/2-1$。

在此设

$$\begin{cases} x_0(n)=x(n)+x\left(n+\dfrac{N}{2}\right) \\ x_1(n)=\left[x(n)-x\left(n+\dfrac{N}{2}\right)\right]W_N^n \end{cases} \tag{5.3.5}$$

则式(5.3.3)和式(5.3.4)分别表示 $x_0(n)$和 $x_1(n)$的 $N/2$ 点 DFT 结果，即

$$X(2r)=\mathrm{DFT}[x_0(n)]=\sum_{n=0}^{\frac{N}{2}-1}x_0(n)W_{N/2}^{nr} \tag{5.3.6}$$

$$X(2r+1)=\mathrm{DFT}[x_1(n)]=\sum_{n=0}^{\frac{N}{2}-1}x_1(n)W_{N/2}^{nr} \tag{5.3.7}$$

此时，N 点 DFT 结果 $X(k)$按照 k 的奇偶取值分成了两个 $N/2$ 点的 DFT 来计算。图 5.3.1 给出了将 N 点 DFT 按频率抽取分解为两个 $N/2$ 点 DFT 的蝶形运算流图。

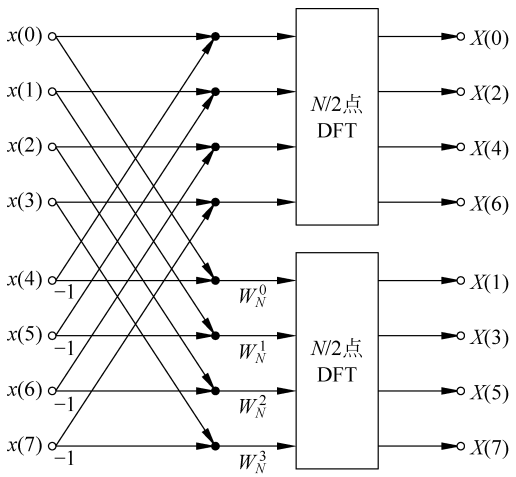

图 5.3.1　按频率抽取，将 N 点 DFT 分解为两个 $N/2$ 点 DFT($N=8$)

与按时间抽取的思路类似，还可以把 $N/2$ 点的 DFT 分解为两个 $N/4$ 点的 DFT，分解依据仍然按照频率顺序的奇偶取值，一直分解到基本运算单元为 2 点 DFT 为止。$N=8$ 点的完整 DIF-FFT 流图，如图 5.3.2 所示。

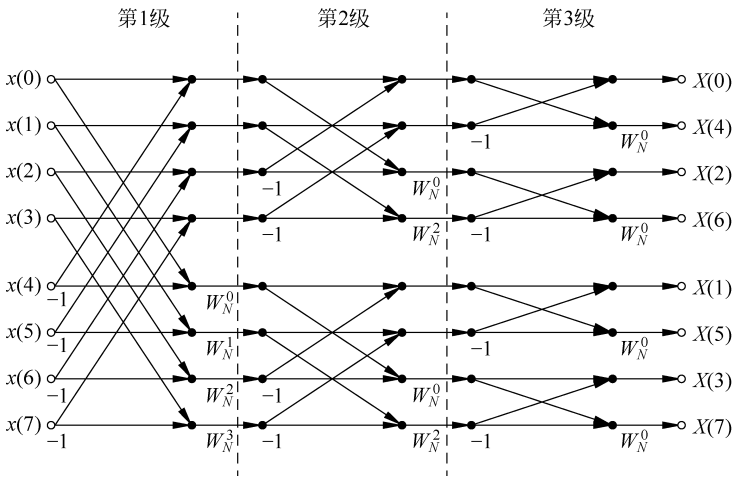

图 5.3.2　$N=8$ 点的基-2 DIF-FFT 流图

从前面的分析可知，基-2 DIT-FFT 算法是根据输入序列的奇偶顺序不断进行分组，基-2 DIF-FFT 算法是根据输出序列的奇偶顺序不断进行分组，这两种计算 DFT 的算法思路和运算流图非常相似，主要有以下特点。

（1）DIF-FFT 算法与 DIT-FFT 算法运算量相同。

如果把 FFT 算法看作是一个系统，那么只需要把 DIT-FFT 流图中输入和输出进行交换，再把所有箭头反向，所有数据就会反向“流动”，新的流动方式即 DIF-FFT 流图。DIF-FFT 流图与 DIT-FFT 流图的这种倒置关系，意味着二者运算量完全相同，即都是 $\frac{N}{2}\log_2 N$ 次复数乘法和 $N\log_2 N$ 次复数加法。

（2）同址运算。

与 DIT-FFT 算法类似，DIF-FFT 运算也满足同址运算的特点，即每一级运算数据都可以保存在同样地址的 N 个存储单元中，且蝶形运算支路的系数也只需要 $N/2$ 个存储单元即可。

（3）输入顺序，输出倒序。

DIF-FFT 流图与 DIT-FFT 流图呈倒置关系，二者的输入和输出进行了交换，当然输入和输出位置关系的特点也进行了交换。

（4）蝶形运算。

与 DIT-FFT 算法类似，DIF-FFT 运算也满足蝶形运算的特点。区别在于：DIT-FFT 流图中蝶形运算类型随着迭代次数成倍增加，DIF-FFT 流图中蝶形运算类型随着迭代次数成倍减少。DIF-FFT 流图的这个特点从图 5.3.2 中也很容易看出来。

因为 DIF-FFT 流图与 DIT-FFT 流图呈倒置关系，即将 DIT-FFT 流图中输入和输

出进行交换，再把所有箭头反向就可得到 DIF-FFT 流图，而旋转因子的位置和取值并不会发生变化，因此不用专门去考虑 DIF-FFT 流图中的旋转因子。

为便于大家学习掌握 DIT-FFT 算法与 DIF-FFT 算法，表 5.4 对它们的特点进行了总结。

表 5.4　DIT-FFT 算法与 DIF-FFT 算法特点

DIT-FFT 算法	DIF-FFT 算法
同址运算	同址运算
输入倒序，输出顺序	输入顺序，输出倒序
蝶形运算	蝶形运算

例 5.2　请利用 DIF-FFT 流图重做例 5.1。

解：$N=4$ 点的 DIF-FFT 运算流图如图 5.3.3 所示。

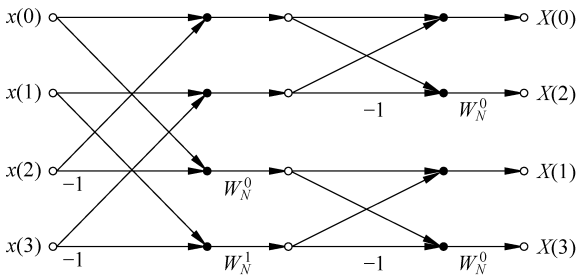

图 5.3.3　$N=4$ 点 DIF-FFT 流图

代入 $x(n)=\{1,3,5,2\}_0$，以及 $W_N^0=1, W_N^1=W_4^1=-\mathrm{j}$，可得运算过程如图 5.3.4 所示。

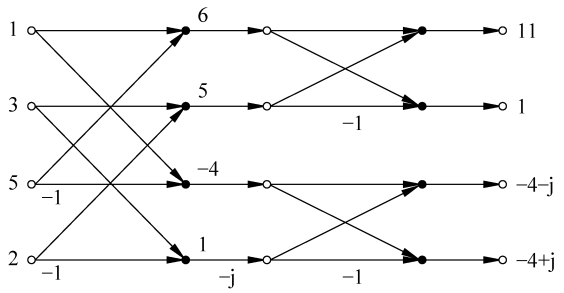

图 5.3.4　例 5.2 运算过程

从图 5.3.4 的输出结果可以看出，DFT 结果为 $X(k)=\{11,-4-\mathrm{j},1,-4+\mathrm{j}\}_0$，注意此时需要将输出结果进行倒序排列。

5.4　FFT 算法的应用技巧

前面主要以数学公式和流图的形式介绍了基-2 DIT-FFT 算法和基-2 DIF-FFT 算法，但在 FFT 算法的硬件和软件实现中还有一些实际问题需要考虑。此外，在实际应用

中如果能够充分利用 FFT 算法的特点，还可以进一步提高处理效率。

1. 查表法计算旋转因子

根据定义，旋转因子 $W_N^r=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}r}$ 需要通过指数函数计算得到。如果在 FFT 运算中旋转因子是实时产生的，就会对 FFT 算法的运算速度产生较大影响。因为在计算机中指数运算耗时较多，并且在 FFT 流图中旋转因子会反复参与运算，使得指数运算反复进行。

一个有效的解决方案就是通过查表法来计算旋转因子，即事先将所有 $N/2$ 个旋转因子计算出来，存储在特定位置成为一个旋转因子表。在 FFT 算法运行时，通过查表来得到所需的旋转因子取值，这样就可使运算速度大为提高。

2. 悄无声息的倒序操作

以 DIT-FFT 算法为例，“输入倒序，输出顺序”为该算法的特点之一，但如果需要事先将输入数据 $x(n)$ 进行倒序操作，这是非常不方便的。

在实际的 FFT 函数或处理器中，不必考虑倒序操作，计算机会通过变址寻址的方式去寻找相应位置的输入数据。因此对于 FFT 函数或处理器而言，都是“输入顺序，输出顺序”的，使用者不必理会内部采用的是 DIT-FFT 流图还是 DIF-FFT 流图，只需要按照自然顺序将时域序列 $x(n)$ 输入即可，输出的也是按照自然顺序排列的频域样本序列 $X(k)$。

3. 用 FFT 实现 IFFT

DFT 算法可以通过 FFT 算法快速实现，其反变换 IDFT 算法也有相应的 IFFT 算法来快速实现。并不需要专门去推导和研究 IFFT 算法，因为用 FFT 算法就可以实现 IFFT 算法。

由式(3.2.7)可知，IDFT 的定义为

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)\mathrm{e}^{\mathrm{j}\frac{2\pi}{N}kn} \tag{5.4.1}$$

其中 n 的取值范围为 $0\leqslant n\leqslant N-1$。

仔细观察 DFT 和 IDFT 公式，二者的最大区别在于旋转因子指数符号相反，互为共轭关系。因此，对式(5.4.1)两边取共轭，可得

$$x^*(n)=\frac{1}{N}\sum_{k=0}^{N-1}X^*(k)\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}kn}=\frac{1}{N}\mathrm{DFT}[X^*(k)] \tag{5.4.2}$$

DFT 运算可由 FFT 运算快速实现，对式(5.4.2)两边再次取共轭可得 $x(n)$，即可用 FFT 算法实现 IFFT 算法，

$$x(n)=\frac{1}{N}\{\mathrm{FFT}[X^*(k)]\}^* \tag{5.4.3}$$

4. 实序列的 FFT

FFT 流图中的旋转因子都为复数，因此无论输入的 $x(n)$ 是复数还是实数，在第 1 级

蝶形运算之后都是复数运算。现实世界中的许多信号都是实信号，如果按照前面介绍的方法进行 FFT 运算，相当于把实信号看作虚部为零的复信号，这样就会“浪费”$x(n)$的虚部信息，降低了运算效率。

提高运算效率的基本思路就是把虚部信息利用起来。对于两个实序列，可以把其中一个实序列作为实部，另外一个实序列作为虚部来进行 FFT 运算；对于一个较长的实序列，可以参照 DIT-FFT 的思想，把偶数序号部分作为实部，奇数序号部分作为虚部来进行 FFT 运算。

第一种情况，在例 3.6 中就已经得到解决。对于实序列 $x_1(n)$和 $x_2(n)$，长度都是 N，将 $x_1(n)$和 $x_2(n)$组合成一个 N 点的复序列 $y(n)=x_1(n)+\mathrm{j}x_2(n)$。

根据例 3.6 的结论可知，$x_1(n)$和 $x_2(n)$的 N 点 FFT 结果为

$$\begin{aligned} X_1(k)&=\frac{1}{2}[Y(k)+Y^*(N-k)] \\ X_2(k)&=\frac{1}{2\mathrm{j}}[Y(k)-Y^*(N-k)] \end{aligned} \tag{5.4.4}$$

其中 k 的取值范围为 $0\leqslant k\leqslant N-1$，$Y(k)$表示复序列 $y(n)$的 FFT 结果。为了能利用基-2 的 FFT 算法，要求序列长度 $N=2^L$，L 为正整数。如果不满足，则补零满足这个条件。

下面分析用组合复数的方法所需要的运算量(仅讨论复数乘法次数)。$X_1(k)$和 $X_2(k)$可以通过式(5.4.4)由蝶形运算实现，1 次蝶形运算需要 1 次复数乘法，故 N 点的 $X_1(k)$和 $X_2(k)$需要 N 次复数乘法。此外，计算 N 点长复序列 $y(n)$的 FFT 需要 $\frac{N}{2}\log_2 N$ 次复数乘法，故组合复数的方法总共需要 $\frac{N}{2}\log_2 N+N$ 次复数乘法。如果直接计算实序列 $x_1(n)$和 $x_2(n)$的 N 点 FFT 结果，则需要 $2\times\frac{N}{2}\log_2 N=N\log_2 N$ 次复数乘法。表 5.5 给出了直接法和组合复数法计算两个 N 点长实序列的 FFT 所需复数乘法次数。

表 5.5　计算两个 N 点长实序列的 FFT 所需复数乘法次数

N	直接法	组合复数法
128	896	576
256	2048	1280
512	4608	2816
1024	10240	6144
2048	22528	13312
4096	49152	28672

第二种情况，对于一个较长的实序列，同样可以利用组合复数法来降低运算量。

对于 $N=2^L$ 点的实序列 $x(n)$，按照 n 的奇偶取值拆分成两个短序列，即

$$\begin{cases} x_0(r)=x(2r) \\ x_1(r)=x(2r+1) \end{cases} \tag{5.4.5}$$

其中 r 的取值范围为 $0 \leqslant r \leqslant N/2-1$。

将 $x_0(r)$ 和 $x_1(r)$ 组合成一个复序列 $y(r)=x_0(r)+\mathrm{j}x_1(r)$，该复序列的长度为 $N/2$。根据前面结论可知，$x_0(r)$ 和 $x_1(r)$ 的 FFT 结果为

$$\begin{aligned} X_0(k) &= \frac{1}{2}[Y(k)+Y^*(N-k)] \\ X_1(k) &= \frac{1}{2\mathrm{j}}[Y(k)-Y^*(N-k)] \end{aligned} \tag{5.4.6}$$

参照 DIT-FFT 的思想，通过蝶形运算可将 $X_0(k)$ 和 $X_1(k)$ 组合得到 $X(k)$，即

$$\begin{aligned} X(k) &= X_0(k)+W_N^k X_1(k) \\ X\left(k+\frac{N}{2}\right) &= X_0(k)-W_N^k X_1(k) \end{aligned} \tag{5.4.7}$$

其中 k 的取值范围为 $0 \leqslant k \leqslant N/2-1$。

下面分析把长实数序列拆开组合成短复数序列，再进行 FFT 运算需要的复数乘法次数。$X(k)$ 可以通过式(5.4.7)由蝶形运算实现，$N/2$ 点长的 $X(k)$ 只需要 $N/2$ 次蝶形运算，故计算 $X(k)$ 需要 $N/2$ 次复数乘法。根据式(5.4.6)，$X_0(k)$ 和 $X_1(k)$ 也可通过蝶形运算实现，$N/2$ 点长的 $X_0(k)$ 和 $X_1(k)$ 需要 $N/2$ 次蝶形运算，即 $N/2$ 次复数乘法。此外，计算 $N/2$ 点长复数序列 $y(n)$ 需要 $\frac{N/2}{2}\log_2\frac{N}{2}=\frac{N}{4}(\log_2 N-1)$ 次复数乘法，故总共需要 $\frac{N}{4}(\log_2 N-1)+\frac{N}{2}+\frac{N}{2}$ 次复数乘法。直接计算 N 点长实序列 $x(n)$ 的 FFT，需要 $\frac{N}{2}\log_2 N$ 次复数乘法。表 5.6 对这两种方法的运算量进行了对比。

表 5.6　计算一个 N 点长实序列的 FFT 所需复数乘法次数

N	直接法	拆分组合复数法
128	448	320
256	1024	704
512	2304	1536
1024	5120	3328
2048	11264	7168
4096	24576	15360

习题

1. 如果通用计算机计算一次复数乘法需要 40ns，计算一次复数加法需要 5ns，用它来计算 512 点的 DFT，请问直接计算需要多少时间？用 FFT 算法来计算需要多少时间？

2. 已知序列 $x(n)$ 和 $y(n)$ 都为 100 点长，用 FFT 来计算这两个序列的线性卷积，假设计算机性能同第 1 题，请问需要多少时间？

3. 已知 1024 点序列 $x(n)$的 DFT 结果为 $X(k)$，假设计算机性能同第 1 题，请问用 FFT 来计算 IDFT 结果需要多少时间？

4. 一个实时卷积器是用 FFT 的重叠保留法分段处理的，假定系统的单位脉冲响应长度为 128，每段 1024 点 FFT 的运算时间为 0.2s，一次复数乘法的时间为 200μs，不计数据采集、存储和加法的运算时间，请问：

(1) 该系统的采样频率最高可以达到多少？

(2) 若两路信号同时卷积，则采样频率最高是多少？

5. 设序列 $x(n)$是一个 M 点的有限长序列，$0\leqslant n\leqslant M-1$，$x(n)$的 z 变换为 $X(z)$。现将 $X(z)$在单位圆上进行 N 点的等间隔采样，试分别针对 $N\leqslant M$ 和 $N>M$ 这两种情况，找出只用一个 N 点 FFT 就能计算 $X(z)$的 N 个采样值的方法。

6. 已知序列 $x(n)=\{1,5,3,4\}_0$

(1) 试给出 DIT-FFT 的信号流图，注意标出节点系数；

(2) 利用流图计算输出结果 $X(k)$；

(3) 计算 $X(k)$的幅度谱和相位谱。

7. 已知 $X(k)=\{11,-4-\mathrm{j},1,-4+\mathrm{j}\}_0$(例 5.1)，请用 DIT-FFT 的信号流图来计算 $x(n)$。

8. 用重叠保留法来完成以下滤波功能，其中 $h(n)$长度为 31，信号 $x(n)$的长度为 19000，请利用 512 点的 FFT 算法来实现滤波运算。

9. 同上题的数据，请采用重叠相加法来实现滤波运算。

10. 请编写重叠相加法的 MATLAB 函数，并用来验证例 4.6 的结果。

11. 请编写重叠保留法的 MATLAB 函数，并用来验证例 4.7 的结果。

12. 请编写利用 FFT 实现 IFFT 运算的 MATLAB 函数，并与 MATLAB 函数 ifft 进行比较，验证 $X(k)=\{11,-4-\mathrm{j},1,-4+\mathrm{j}\}_0$ 的 IFFT 结果。