第5章 快速傅里叶变换

快速傅里叶变换（Fast Fourier Transform）是指在经典离散傅里叶变换（DFT）的基础上，以降低运算量和提高运算效率为目的，对DFT进行的各种改进，各种改进算法均称为傅里叶变换的快速算法，简称FFT。采用快速算法可以使离散傅里叶变换所需要的乘法次数大为减少，而且序列的点数越大，FFT算法对计算量的减少就越显著。


视频讲解

5.1 引言

快速傅里叶变换并不是一种新的变换，而是对DFT的一种算法改进。因此，FFT算法的基础是DFT，核心是对DFT算法进行高效改进。

有限长序列的离散傅里叶变换的时域和频域均为离散形式，可以采用计算机进行计算。DFT在数字信号处理中应用非常广泛。例如，FIR滤波器设计需要由$h(n)$求$H(k)$以及由$H(k)$求$h(n)$，这些都需要进行DFT计算。信息通信、图像传输、雷达、声呐等领域需要进行信号频谱分析，这也需要进行DFT计算。在数字系统分析与设计中，也经常要进行DFT计算。然而，在20世纪60年代中期以前，由于DFT的计算量非常大，且计算机的运算效率低，故很少采用计算机进行DFT运算。1965年，库利（J. W. Cooley）和图基（J. W. Tukey）在《计算数学》（*Mathematics of Computation*）杂志上发表了著名的“机器计算傅里叶级数的一种算法”的论文，提出了DFT的一种快速算法，傅里叶变换的快速算法才引起广泛重视，从此，快速傅里叶变换的应用情况发生了根本性的变化，FFT经过不断完善和改进，其运算时间一般可缩短至两个数量级以上，运算时间的有效缩短有效推动了FFT的广泛应用。

党的二十大报告提出“实现高水平科技自立自强，进入创新型国家前列”，实现这一目标，需要我们发奋努力，在科学和技术的各个领域不断创新、不断突破。快速傅里叶变换具有非常广泛的应用，尖端医疗器械、先进电力控制设备、5G通信技术、星空探测、航母控制系统、高铁信号处理系统等都离不开FFT。

5.2 DFT的运算量及改进的途径

5.2.1 DFT算法运算量的估计

设序列$x(n)$为N点有限长序列，其DFT算法如下。

正变换（DFT）：

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk},\quad k=0,1,\cdots,N-1 \tag{5-1}$$

逆变换(IDFT)：

$$x(n)=\frac{1}{N}\sum_{n=0}^{N-1}X(k)W_N^{-nk},\quad n=0,1,\cdots,N-1 \tag{5-2}$$

比较 DFT 和 IDFT 可以发现，二者的算法结构非常类似，差别很小，它们的差异性主要体现在如下两点。

(1) 正变换和逆变换算法中 W_N 的指数符号相反；

(2) 正变换求和符号前的系数为 1，而逆变换求和符号前的乘数因子为 $1/N$。

由上述分析可知，正变换和逆变换的运算量几乎相等，因此，一般以 DFT 正变换为例进行运算量的估算。由于运算结构相同，故逆变换可以根据正变换的运算量进行准确估计。

为了提高适应性，一般假定序列 $x(n)$在复数范围内取值。实际上，无论 $x(n)$是实数还是复数，W_N^{nk} 一般都是复数、因而 $X(k)(k=,0,1,2,\cdots,N-1)$一般也是复数。

根据 DFT 正变换公式，对于 N 点频域序列 $X(k)$而言，每计算一个 $X(k)$值，需进行 N 次复数乘法，即求和符号中 $x(n)$与 W_N^{nk} 有 N 次复数乘，$(N-1)$次复数加；而 $k=0,1,2,\cdots,N-1$，即 $X(k)$一共有 N 个值，因此 DFT 正变换的全部运算共有 N^2 次复数乘法，$(N(N-1))$次复数加法。

复数运算是以实数运算为基础来完成的，因此，DFT 正变换可写为如下形式。

$$\begin{aligned}X(k)&=\sum_{n=0}^{N-1}x(n)W_N^{nk}=\sum_{n=0}^{N-1}\{\mathrm{Re}[x(n)]+\mathrm{jIm}[x(n)]\}\{\mathrm{Re}[W_N^{nk}]+\mathrm{jIm}[W_N^{nk}]\}\\&=\sum_{n=0}^{N-1}\mathrm{Re}[x(n)]\mathrm{Re}[W_N^{nk}]-\mathrm{Im}[x(n)]\mathrm{Im}[W_N^{nk}]+\\&\quad\mathrm{j}(\mathrm{Re}[x(n)]\mathrm{Im}[W_N^{nk}]+\mathrm{Im}[x(n)]\mathrm{Re}[W_N^{nk}]\end{aligned} \tag{5-3}$$

根据式(5-3)，可得：

(1) 一次复数乘法＝4 次实数乘法＋2 次实数加法；

(2) 一次复数加法＝2 次实数加法。

因此，每计算一个 $X(k)$需 $4N$ 次实数乘法以及 $2N(2N-1)$次实数加法。所以 N 点的 DFT 总计算量包括 $4N^2$ 次实数乘法和 $2N(2N-1)$次实数加法，如表 5-1 所示。

表 5-1　N 点 DFT 运算量表

计算方式	复数乘法	复数加法	实数乘法	实数加法
运算量	N^2	N^2-N	$4N^2$	$4N^2-2N$

上述运算量估计是纯理论估计值，它包含了乘以常数 1 等特殊情况，与实际运算量会略有差异。这是因为式(5-1)中时域因子 n 和频域因子 k 取不同值时，W_N^{nk} 可能等于 1、−1 或 j，这时无须乘法运算，如：

$$W_N^0=1,\quad W_N^{N/2}=-1,\quad W_N^{N/4}=-\mathrm{j}$$

这些特殊情况会使实际运算量略小于运算量的理论值，当 N 越大时，这种特例所占的比例就越小。因此，为了统一和方便比较，DFT 运算量估计一般不考虑上述特殊情况。

根据表 5-1，完成一个长度为 N 点的 DFT 计算，乘法次数和加法次数都与 N^2 成正比，

当 N 很大时，运算量是非常大的。例如，若 $N=8$，则 DFT 运算量为 64 次复数乘法；若 $N=1024$，则 DFT 运算量为 1048576 次复数乘法，如果系统对实时性要求很高，这样巨大的运算量对计算机的速度要求非常高。若采样点数 N 进一步增大，则运算量以平方关系快速增长，因此，需要对 DFT 的算法进行改进，以有效降低 DFT 算法的运算量，提高运算速度。


视频讲解

5.2.2　降低运算量的途径

任何改进都必须从可改之处着手，从最有效之处改进，DFT 算法的改进也不例外。根据 5.2.1 节的分析可知，影响 DFT 运算量的因素有如下两个。

(1) W_N^{nk} 等于 1、−1 等常数时可适当降低运算量。

(2) 序列 $x(n)$ 的长度 N（即点数）对运算量有显著影响。

下面分析如何利用这两个因素减少运算量。

1. 利用 W_N^{nk} 减少运算量

利用 W_N^{nk} 等于 1、−1 等常数，以及 W_N^{nk} 的相关性质，可以适当降低运算量，主要包括如下特性。

(1) $W_N^{nk}=1,-1,\cdots$ 等常数，如，$W_N^0=1$，$W_N^{N/2}=-1$，$W_N^{N/4}=-\mathrm{j}$。

(2) W_N^{nk} 的对称性。

$$(W_N^{nk})^*=W_N^{-nk}$$

(3) W_N^{nk} 的周期性。

$$W_N^{nk}=W_N^{(n+N)k}=W_N^{n(k+N)}$$

(4) W_N^{nk} 的可约性。

$$W_N^{nk}=W_{mN}^{mnk},\quad W_N^{nk}=W_{N/m}^{nk/m}$$

由此可得

$$W_N^{n(N-k)}=W_N^{(N-n)k}=W_N^{-nk},\quad W_N^{n/2}=-1,\quad W_N^{(k+N/2)}=-W_N^k$$

利用上述特性，可以使 DFT 运算中某些项无须乘法运算，也可以使某些项合并运算。

2. 降低序列长度 N 减少运算量

由于运算量与 N^2 成正比，因此，减小 N 对减少 DFT 运算量是最有效的，它将使得运算量以平方的速度快速下降。然而，N 的减小是受限的，即受到奈奎斯特采样定理约束，序列长度 N 不能无原则缩短，工程上，一般应在满足采样定理的基础上，选择符合采样要求的最短序列。

DFT 的运算量与 N^2 成正比，N^2 是非线性函数，N 减小时运算量显著下降，即点数少的序列 DFT 运算量比点数多的序列的运算量要小很多。因此，当点数 N 不能进一步减小时，降低运算量的思路是将 N 点的序列 $x(n)$ 分解为 2 个 $N/2$ 点的序列，这样两个短序列的复数乘法运算量均为

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

经过一次分解，复数乘法运算量大约降低了一半。以此思路继续分解，每一个 $N/2$ 点的序列又可以分解为 2 个 $N/4$ 点的 DFT，运算量可进一步减小，以此类推，对每一个新产生的短序列继续分解，直至分解到全部子序列均为 2 点为止。

DFT 算法改进主要依据这一思路展开，因此快速算法理论上主要可以分成两大类，一

种是按时间抽取(Decimation-In-Time,DIT)法,将长序列分解为短序列,另一种是按频率抽取(Decimation-In-Frequency,DIF)法,将长序列分解为短序列。

5.3 按时间抽取(DIT)基-2 FFT 算法

FFT 快速算法的思路是将长序列分解为短序列,并根据 DFT 变换的奇、偶、虚、实的对称性及相关特性,对算法进行改进从而得到快速算法。FFT 算法并没有在理论上对傅里叶变换有新的推进,但在推动傅里叶变换的应用方面却取得了巨大的进步。根据将序列分解为短序列的方法不同而产生了不同的 FFT 算法,典型的算法主要包括 DIT 基-2(按时间抽取)和 DIF 基-2(按频率抽取)算法。

5.3.1 DIT 算法原理

对序列 $x(n)$按序号的奇、偶抽取得到两个子序列,并不断进行分组而实现的 FFT 算法称为 **DIT 基-2 FFT 算法**。该算法由库利-图基首先提出,故又称为库利-图基算法。

设序列 $x(n)$的点数为 $N=2^L$,L 为正整数。若序列点数不满足条件,可以从序列末端补零点,使之满足要求。N 为 2 的整数幂的 FFT 称为基-2 FFT。


视频讲解

1. 序列 $x(n)$第一次分解

将长度为 $N=2^L$ 的序列 $x(n)$ $(n=0,1,\cdots,N-1)$,根据时域序号 n 的奇、偶分为如下两组(两个子序列):

$$\begin{cases} x_1(r)=x(2r) \\ x_2(r)=x(2r+1) \end{cases} \quad r=0,1,\cdots,\frac{N}{2}-1 \tag{5-4}$$

于是,DFT 正变换公式可以化为

$$\begin{aligned} X(k)=\mathrm{DFT}[x(n)]&=\sum_{n=0}^{N-1}x(n)W_N^{nk}=\sum_{\substack{n=0\\ n\text{为偶数}}}^{N-1}x(n)W_N^{nk}+\sum_{\substack{n=0\\ n\text{为奇数}}}^{N-1}x(n)W_N^{nk} \\ &=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2rk}+\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_N^{(2r+1)k}=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)(W_N^2)^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_2(r)(W_N^2)^{rk} \end{aligned} \tag{5-5}$$

由于 $W_N^2=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}\cdot 2}=W_{N/2}$,式(5-5)进一步可以化为

$$X(k)=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_{N/2}^{rk} \tag{5-6}$$

由于 $x_1(r)$、$x_2(r)$均为 $N/2$ 点的时域序列,其对应的 DFT 如下:

$$\begin{cases} X_1(k)=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{rk}=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_{N/2}^{rk} \\ X_2(k)=\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_{N/2}^{rk}=\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_{N/2}^{rk} \end{cases} \tag{5-7}$$

因此，式(5-6)可表示为如下形式：

$$X(k)=X_1(k)+W_N^kX_2(k) \tag{5-8}$$

$X_1(k)$、$X_2(k)$都是长度为 $N/2$ 点的序列 $x_1(r)$、$x_2(r)$的标准 DFT 正变换，至此，一个 N 点 DFT 已分解成两个 $N/2$ 点的 DFT。

(1) $X_1(k)$、$X_2(k)$都是 $N/2$ 点的标准 DFT 正变换，因此，一个 N 点 DFT 已分解成两个 $N/2$ 点的 DFT。

(2) 式(5-8)中 $x_1(r)$、$x_2(r)$以及 $X_1(k)$、$X_2(k)$都是 $N/2$ 点的序列，即 $r,k=0,1,\cdots,N/2-1$，而 $X(k)$却有 N 点，因此，根据式(5-8)只能计算 $X(k)$的前一半点的结果，而 $X(k)$后一半点的结果需要根据 DFT 的周期性以及 W_N^{nk} 的周期性对式(5-8)进行改进。

根据 W_N^{nk} 周期性

$$W_{N/2}^{rk}=W_{N/2}^{r\left(k+\frac{N}{2}\right)} \tag{5-9}$$

代入式(5-7)可得

$$X_1\left(\frac{N}{2}+k\right)=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{r\left(\frac{N}{2}+k\right)}=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{rk}=X_1(k) \tag{5-10}$$

对于 $X_2(k)$，同样可得

$$X_2\left(\frac{N}{2}+k\right)=X_2(k) \tag{5-11}$$

由于 $W_N^{N/2}=-1$，可得

$$W_N^{\left(\frac{N}{2}+k\right)}=W_N^{N/2}W_N^k=-W_N^k \tag{5-12}$$

因此，将式(5-10)～式(5-12)代入式(5-8)，可得后一半点的公式如下：

$$\begin{aligned}X\left(k+\frac{N}{2}\right)&=X_1\left(k+\frac{N}{2}\right)+W_N^{\left(k+\frac{N}{2}\right)}X_2\left(k+\frac{N}{2}\right)\\&=X_1(k)-W_N^kX_2(k),\quad k=0,1,\cdots,\frac{N}{2}-1\end{aligned} \tag{5-13}$$

由此可得计算全部 $X(k)$值的算法如下：

$$\begin{cases}X(k)=X_1(k)+W_N^kX_2(k),\quad k=0,1,\cdots,\dfrac{N}{2}-1\\X\left(k+\dfrac{N}{2}\right)=X_1(k)-W_N^kX_2(k),\quad k=0,1,\cdots,\dfrac{N}{2}-1\end{cases} \tag{5-14}$$

根据式(5-14)，由长度为 $N/2$ 点的 $X_1(k)$和 $X_1(k)$，即可求出 N 点 $X(k)$的全部值。

式(5-14)的运算规律可用图 5-1 的蝶形运算流图进行描述，式(5-14)中的乘法运算，对应于系数为 W_N^k 的蝶形支路。采用计算流图表示算法结构更简明扼要，也更加形象化，若支路上标有系数，则表示信号与支路系数相乘，若支路没有标注系数，则表示该支路的系数为 1。

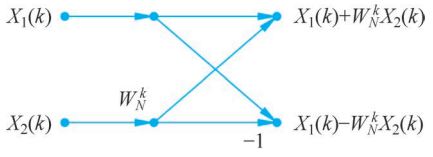


图 5-1　按时间抽取法蝶形运算流图

图 5-1 是一个基于 DIT 基-2 快速算法的一般蝶形运算图，若以 $k=0,1,2,\cdots,\left(\frac{N}{2}-1\right)$代

入图 5-1,则可得到 FFT 快速运算的第一次分解蝶形运算图。为便于深入理解 FFT 快速算法原理,以 $N=2^3=8$ 为例,详细分析图 5-1 所包含的全部碟形运算。

当 $k=0$ 时,碟形运算如下:

$$\begin{cases} X(0)=X_1(0)+W_N^0X_2(0) \\ X(4)=X_1(0)-W_N^0X_2(0) \end{cases}$$

当 $k=1$ 时,碟形运算如下:

$$\begin{cases} X(1)=X_1(1)+W_N^1X_2(1) \\ X(5)=X_1(1)-W_N^1X_2(1) \end{cases}$$

当 $k=2$ 时,碟形运算如下:

$$\begin{cases} X(2)=X_1(2)+W_N^2X_2(2) \\ X(6)=X_1(2)-W_N^2X_2(2) \end{cases}$$

当 $k=3$ 时,碟形运算如下:

$$\begin{cases} X(3)=X_1(3)+W_N^3X_2(3) \\ X(7)=X_1(3)-W_N^3X_2(3) \end{cases}$$

这就是图 5-1 或式(5-14)所包含的全部蝶形运算,对于上述表达式中的 $X_1(k)$、$X_2(k)$ $(k=0,1,2,3)$的计算,依然采用传统 DFT 公式进行计算。根据上面 $k=0,1,2,3$ 的 4 个表达式以及式(5-14),可以得到 8 点 DFT 经过第一次按序号 n 的奇偶分组(分解)之后计算 $X(k)$的过程,如图 5-2 所示。

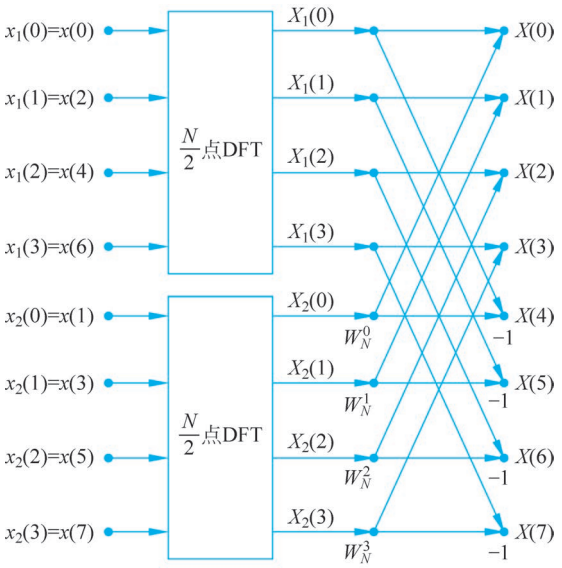


图 5-2 一个 N 点 DFT 分解为两个 N/2 点 DFT 算法过程

图 5-2 中,标注 DFT 的两个方框表示依然采用 DFT 公式进行计算,即 $X_1(0)\sim X_1(3)$ 采用式(5-7)中的第一式计算,$X_2(0)\sim X_2(3)$采用式(5-7)中的第二式计算,而最右端一列的 $X(0)\sim X(3)$采用式(5-14)中的第一式计算,$X(4)\sim X(7)$采用式(5-14)中的第二式计算。可以看出,每一个蝶形运算仅需一次复数乘法 $X_2(k)W_N^k$ 及两次复数加法运算。

图 5-2 完整地表示了一个 N 点 DFT 分解成两个 $N/2$ 点 DFT 的过程，该图完全等价地描述了式(5-14)的全部计算过程。根据第一次分解过程，若直接计算 $N/2$ 点 DFT，则每一个 $N/2$ 点 DFT 有 $\left(\frac{N}{2}\right)^2=\frac{N^2}{4}$ 次复数乘法，$\frac{N}{2}\left(\frac{N}{2}-1\right)$ 次复数加法，两个 $N/2$ 点 DFT 共需 $2\times\left(\frac{N}{2}\right)^2=\frac{N^2}{2}$ 次复数乘法和 $N\left(\frac{N}{2}-1\right)$ 次复数加法。

此外，采用蝶形运算将两个 $N/2$ 点 DFT 组合成一个 N 点 DFT 时，有 $N/2$ 个蝶形运算，还有 $N/2$ 次复数乘法及 $2\times N/2=N$ 次复数加法。因而第一次分解完成之后，复数运算量如下。

复数乘法运算量：

$$\frac{N^2}{2}+\frac{N}{2}+\frac{N(N+1)}{2}\approx\frac{N^2}{2}(\text{次})$$

复数加法运算量：

$$N\left(\frac{N}{2}-1\right)+N=\frac{N^2}{2}$$

由此可知，计算 N 点序列 $x(n)$ 的频域序列 $X(k)$，在经过第一次分解之后，运算量大约为直接采用 DFT 运算量的一半。


视频讲解

2. 序列 $x(n)$ 第二次分解

N 点序列分解为两个 $N/2$ 点短序列可以降低运算量，以此类推，由于 $N=2^L$，$N/2$ 仍是偶数，对 $x_1(r)$、$x_2(r)$ 两个 $N/2$ 点子序列应继续按序号 r 的奇偶分别分解为两个 $N/4$ 点子序列。

1) 对 $x_1(r)$ 分解

按第一次分解的原理，先将 $x_1(r)$ 分解为 $x_3(r)$、$x_4(r)$。

$$\left.\begin{aligned}x_1(2l)&=x_3(l)\\x_1(2l+1)&=x_4(l)\end{aligned}\right\},\quad l=0,1,\cdots,\frac{N}{4}-1 \tag{5-15}$$

$$\begin{aligned}X_1(k)&=\sum_{l=0}^{\frac{N}{4}-1}x_1(2l)W_{N/2}^{2lk}+\sum_{l=0}^{\frac{N}{4}-1}x_1(2l+1)W_{N/2}^{(2l+1)k}\\&=\sum_{l=0}^{\frac{N}{4}-1}x_3(l)W_{N/4}^{lk}+W_{N/2}^{k}\sum_{l=0}^{\frac{N}{4}-1}x_4(l)W_{N/4}^{lk}\\&=X_3(k)+W_{N/2}^{k}X_4(k),\quad k=0,1,\cdots,\frac{N}{4}-1\end{aligned} \tag{5-16}$$

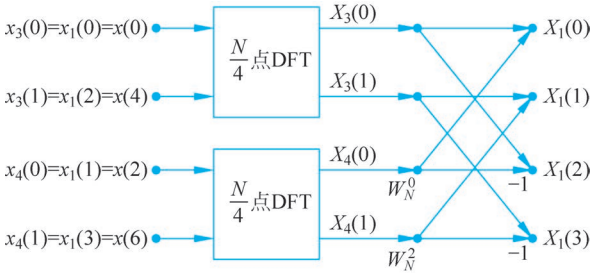
式(5-16)可以求出 $X_1(k)$ 前一半点的值 $\left(k=0,1,\cdots,\frac{N}{4}-1\right)$，对于 $X_1(k)$ 的后一半点，根据第一次分解的方法，可得

$$X_1\left(\frac{N}{4}+k\right)=X_3(k)-W_{N/2}^{k}X_4(k),\quad k=0,1,\cdots,\frac{N}{4}-1$$

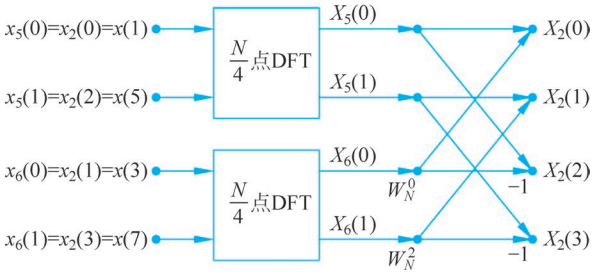
式中

$$\begin{cases} X_3(k)=\sum\limits_{l=0}^{\frac{N}{4}-1}x_3(l)W_{N/4}^{lk} \\ X_4(k)=\sum\limits_{l=0}^{\frac{N}{4}-1}x_4(l)W_{N/4}^{lk} \end{cases} \tag{5-17}$$

与第一次分解类似，对 $x_1(r)$ 的分解过程也可以用蝶形图进行形象的表述，图 5-3(a)给出了当 $N=8$ 时，将一个 $N/2$ 点的 DFT 分解成两个 $N/4$ 点 DFT，并由这两个 $N/4$ 点 DFT 组合成一个 $N/2$ 点 DFT 的过程。



(a) 序列$x_1(r)$的蝶形运算



(b) 序列$x_2(r)$的蝶形运算

图 5-3　N/2 点 DFT 分解为两个 N/4 点 DFT

2）对 $x_2(r)$ 分解

对 $x_2(r)$ 也可进行同样的分解：

$$\begin{cases} X_2(k)=X_5(k)+W_{N/2}^{k}X_6(k) \\ X_2\left(\dfrac{N}{4}+k\right)=X_5(k)-W_{N/2}^{k}X_6(k) \end{cases},\quad k=0,1,\cdots,\frac{N}{4}-1$$

式中

$$\begin{cases} X_5(k)=\sum\limits_{l=0}^{\frac{N}{4}-1}x_2(2l)W_{N/4}^{lk}=\sum\limits_{l=0}^{\frac{N}{4}-1}x_5(l)W_{N/4}^{lk} \\ X_6(k)=\sum\limits_{l=0}^{\frac{N}{4}-1}x_2(2l+1)W_{N/4}^{lk}=\sum\limits_{l=0}^{\frac{N}{4}-1}x_6(l)W_{N/4}^{lk} \end{cases} \tag{5-18}$$

与 $x_1(r)$ 类似，$x_2(r)$ 的分解过程也可以用蝶形图描述，图 5-3(b)给出了当 $N=8$ 时，序列 $x_2(r)$ 分解成两个 $N/4$ 点的 DFT，并由这两个 $N/4$ 点 DFT 组合成一个 $N/2$ 点 DFT 的过程。

至此，一个 N 点（$N=8$）DFT 分解成了四个 $\frac{N}{4}=2$ 点的 DFT。经过两次分解之后，由 $x(n)$ 计算 $X(k)$ 的运算过程如图 5-4 所示。为了统一支路上的乘数因子（又称为旋转因子，也称为支路系数），根据 W_N^{nk} 的可约性，对 W_N^{nk} 类型的支路系数进行了适当的运算，如 $W_{N/2}^{k}=W_N^{2k}$。图 5-4 中，标注 DFT 的方框，无论序列长度是多少，均表示采用经典 DFT 计算。

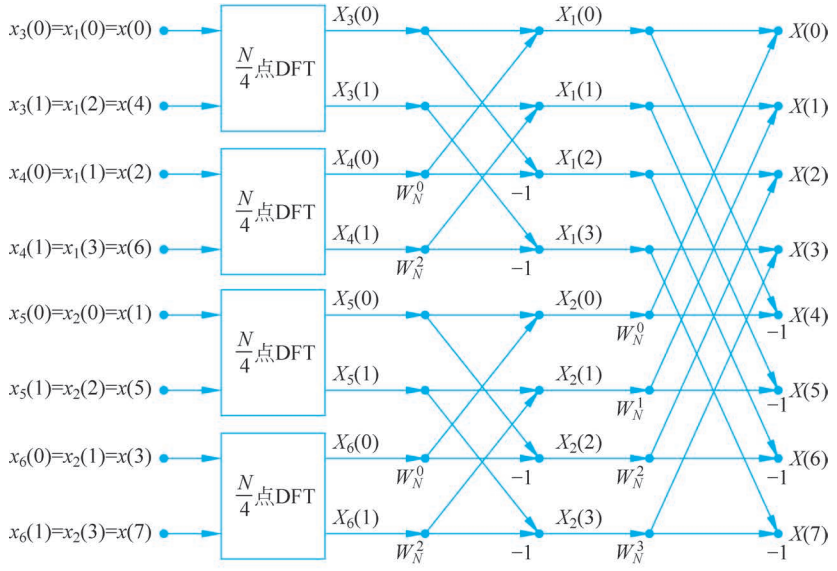


图 5-4　一个 N 点 DFT 分解为四个 $N/4$ 点 DFT（$N=8$）

显然，序列 $x(n)$ 经过第二次分解之后，由四个 $N/4$ 点的 DFT 以及右边的两级蝶形组合运算来计算 N 点 DFT，其运算量比只有第一次分解的运算量又降低了大约一半。

3. 序列 $x(n)$ 第三次分解

以 $N=2^3=8$ 为例，经过两次分解以后，图 5-4 左边一列为四个 $N/4$ 点的 DFT，4 个子序列的长度均为 2 点，最左列每一框图都是 2 点 DFT 运算，其输出分别为 $X_3(k)$、$X_4(k)$、$X_5(k)$、$X_6(k)$，$k=0,1$，其中 $X_3(k)$、$X_4(k)$ 依据式（5-17）进行计算，$X_5(k)$、$X_6(k)$ 依据式（5-18）进行计算。

根据 DFT 的公式，展开 2 点 DFT 公式可以发现，完成 2 点 DFT 的计算只需 1 次复数乘法（由于 $W_2^0=1$，实际上无须乘运算）和 2 次加法运算，将图 5-4 左上第一个方框中的 $N/4$ 点 DFT 按公式展开，形式如下：

$$\begin{cases} X_3(0)=x(0)+W_2^0x(4) \\ X_3(1)=x(0)-W_2^0x(4) \end{cases} \tag{5-19}$$

以 $N=8$ 为例，根据式（5-19）可得 2 点长序列的蝶形算法流图，即计算 $X_3(k)$ 的蝶形运算流图如图 5-5 所示。

当 $N=8$ 时，图 5-4 左列其他 3 个 DFT（分别计算 $X_4(k)$、$X_5(k)$、$X_6(k)$）也都是 2 点 DFT，都可以将 2 点 DFT 框图直接展开，并以蝶形运算表示，由此可得出当 $N=8$ 时 FFT 的完整碟形运算图，如图 5-6 所示。

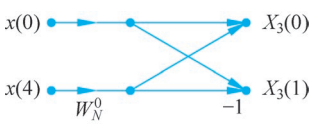


图 5-5　2 点 DFT 运算流图

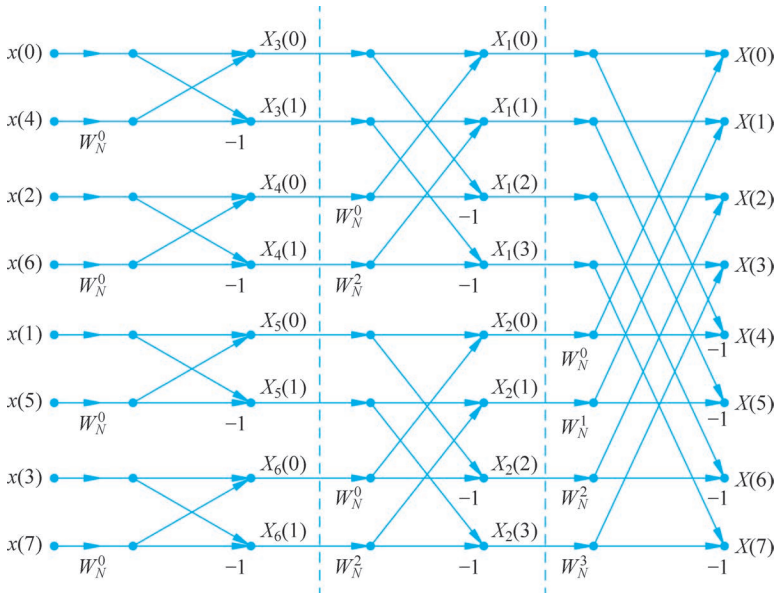


图 5-6 N=8 时(DIT)FFT 运算流图

图 5-6 称为 FFT 快速运算的完整蝶形运算图($N=8$ 时),FFT 蝶形图由 $x(n)$ 计算 $X(k)$ 是从左至右按列(图中虚线)进行计算的,因此,当 $N=8$ 时只需 $L=3$ 级($2^L=2^3$)运算。

第 1 级运算。按照左边第一列 4 个蝶形图进行计算,序列 $x(n)$ 经过分组排队后,以 $[x(0),x(4),x(2),x(6),x(1),x(5)\ x(3),x(7)]$ 的顺序输入,按照蝶形运算规律,计算出第一级运算的输出值 $[X_3(0),X_3(1),X_4(0),X_4(1),X_5(0),X_5(1),X_6(0),X_6(1)]$。

第 2 级运算。按第二列 4 个蝶形图进行计算,这时第一级的输出就是第二级的输入值,依据第二列的 4 个蝶形图(上下两组,每组两个蝶形图)进行计算,得出输出值 $[X_1(0),X_1(1),X_1(2),X_1(3),X_2(0),X_2(1),X_2(2),X_2(3)]$。

第 3 级运算。按第三列 4 个蝶形图进行计算,根据输入 $[X_1(0),X_1(1),X_1(2),X_1(3),X_2(0),X_2(1),X_2(2),X_2(3)]$ 按最右一列的 4 个蝶形图计算出 8 点频域序列 $X(k),k=0,1,2,\cdots,N-1$。

上述 3 级计算中,每一级计算均包含 $N/2$ 次复数乘法运算,且每一个基本蝶形运算仅包含一个乘法,因此,若 $N=8$,依据图 5-6 所示的 FFT 蝶形运算计算 DFT,则仅需 12 次复数乘法运算(包括乘 $W_N^0=1$)。

上述 3 级蝶形运算中,每一级基本蝶形图的系数具有明显的规律性。

(1) 第 1 级蝶形运算的系数为 $W_N^0=W_8^0$。

(2) 第 2 级蝶形运算的系数为 W_N^0,W_8^2。

(3) 第 3 级蝶形运算的系数为 W_N^0,W_8^1,W_N^2,W_N^3。

以此类推,大家也可以得出当 $N=16$ 时,DIT 基-2 FFT 快速算法完整的蝶形运算流图以及与流图相对应的 FFT4 级运算过程,同样的方法,也可以得出 $N=2^L$ 的完整 FFT 算法流程图以及 FFT 的 L 级运算过程。

DIT 基-2 算法的每一次分解都是按输入序列 $x(n)$ 的序号 n 是偶数还是奇数进行分解,因此称为按时间抽取法。

5.3.2　DIT 算法的运算量

根据 DIT 算法的 FFT 流图，当 $N=2^L$ 时，蝶形流图从左往右共有 L 级蝶形运算，每一级都包含 $N/2$ 个基本蝶形运算单元，每个基本蝶形运算单元包含一次复数乘法、二次复数加法。因此，一个完整蝶形流图在从左至右的 L 级运算中，每一级都需要 $N/2$ 次复数乘法和 N 次复数加法，这样 L 级完整蝶形运算的总运算量如下。

复数乘法次数：

$$m_T=\frac{N}{2}L=\frac{N}{2}\log_2 N \tag{5-20}$$

复数加法次数：

$$a_T=NL=N\log_2 N \tag{5-21}$$

通常情况下，考虑到 $W_N^0=1$、$W_N^{N/2}=-1$ 和 $W_N^{N/4}$ 等因素，实际复数乘法运算量一般略低于理论估计值。通过分析 N 点时域序列的 FFT 算法过程可以得出，发生 $W_N^0=1$ 的情况共有 $N-1\left(1+2+4+\cdots+2^{L-1}=\sum_{i=0}^{L-1}2^i=2^L-1=N-1\right)$ 次，这些特殊系数都不用乘法运算，对于工程信号，一般 N 都比较大，特殊系数在运算量中占比较低。因此，对快速算法运算量的估计一般均按理论值，而不考虑系数为 ±1 等特例。

由于计算机进行乘法运算所需时间比加法运算所需时间长很多，因此，也可以只用复数乘法次数表示运算量的大小。FFT 算法与 DFT 算法的运算量比较如表 5-2 所示。

表 5-2　DFT 算法与 FFT 算法运算量比较

N	N^2	$\frac{N}{2}\log_2 N$	$N^2\Big/\left(\frac{N}{2}\log_2 N\right)$
2	4	1	4.0
4	16	4	4.0
8	64	12	5.4
16	256	32	8.0
32	1024	80	12.8
64	4096	192	21.4
138	16384	448	36.6
256	65536	1024	64.0
512	262144	2304	113.8
1024	1048576	5120	204.8
2048	4194304	11264	372.4

DFT 的复数乘法次数是 N^2，FFT 复数乘法次数为 $\frac{N}{2}\log_2 N$，直接计算 DFT 与用 FFT 算法的运算量（复数乘法次数）之比为

$$\frac{N^2}{\frac{N}{2}L}=\frac{N^2}{\frac{N}{2}\log_2 N}=\frac{2N}{\log_2 N} \tag{5-22}$$

根据式(5-20)和式(5-22)可以得出 FFT 算法与 DFT 算法的运算量与序列长度 N 的关

系曲线，如图 5-7 所示，由该图可以看出，FFT 算法在节省运算量上具有巨大的优越性，且 N 越大，FFT 快速算法节约运量的效果就越明显，当 $N=1024$ 时，DFT 算法的运算量约为 FFT 运算量的 205 倍。

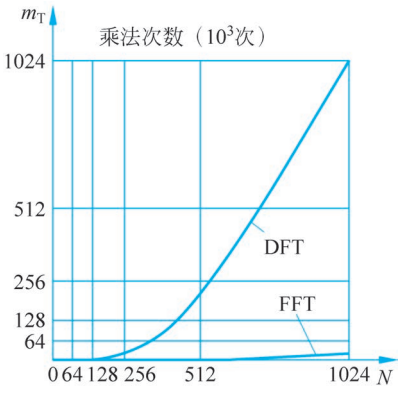


图 5-7 DFT 与 FFT 乘法运算量比较

5.3.3 DIT 基-2 算法的特点

DIT 基-2 算法具有一系列的特点，例如，根据快速算法的分解过程，可以很方便地得出 $N=4$ 点和 $N=8$ 点 FFT 算法流图的递推关系，同样也可以得出 $N=2^{L-1}$ 点与 $N=2^L$ 点 FFT 算法流图之间的递推关系。


视频讲解

1. 倒位序规律

在图 5-6 中，序列经过 $L=3$ 级分解，对于 $N=8$ 点的序列，FFT 的输出序列 $X(k)$ 为正常顺序排列，顺序为 $X(0),X(1),\cdots,X(7)$，但输入序列 $x(n)$ 经过 $L=3$ 次排序以后，已经不是自然顺序，而是按 $x(0),x(4),x(2),x(6),x(1),x(5)\ x(3),x(7)$ 的顺序排列。如果 $N=16,32,\cdots,1024$，则随着 N 的不断增大，输入信号 $x(n)$ 被分组的次数越多，经过不断分组以后，$x(n)$ 的输入顺序也越复杂。若依据逐次奇偶排序的方法计算输入的顺序，则既耗时间，又容易产生错误。

当 N 很大时，在完成各次分组排队之后，$x(n)$ 的输入顺序有没有规律可循呢？答案是肯定的，这一规律就是倒位序规律。

为了探讨输入的排序规律，可以从 $x(n)$ 的分组方法和二进制原理的内在关系入手。

回顾 $x(n)$ 按序号 n 的偶和奇分组的过程，以 $N=8$ 为例，n 可以用三位二进制数表示，$x(n)$ 的序号可表示为 $(n_2n_1n_0)_2$，第一次分组，序号 n 为偶数组成 $x_1(r)$，在上半部分；n 为奇数组成 $x_2(r)$，在下半部分。

从二进制视角看，第一次分组完全由二进制数 n 的最低位 n_0 决定，$n_0=0$ 时 n 为偶数，$n_0=1$ 时 n 为奇数。也就是说，$x(n)$ 按序号 n 的偶和奇分组实际上就是按序号二进制数的最低位 n_0 进行分组。

第一次分组之后，得到两个长度为 4 点的子序列，这时 n_0 已经确定。第二次分组是对 4 点子序列 $x_1(r)$、$x_2(r)$ 进行分组，这时是对 $(n_2n_1n_0)_2$ 中 n_2n_1 进行排队。因此，第二次分组对 $x_1(r)$、$x_2(r)$ 按偶序号和奇序号排队，和第一次分组的道理一样，这时是对二进制数 n_2n_1 进行排队，低位是 n_1，因此由 n_1 决定第二次分组，$n_1=0$ 时 r 为偶数，$n_1=1$ 时 r 为奇数。

同理，第三次分组的奇偶由最低位 n_2 确定，若 $N=8$，则只有 n_2，因此，$n_2=0$ 对应于偶数，$n_2=1$ 对应于奇数。

以 $N=8$ 为例，按序号 n 的偶序号和奇序号进行分组的过程，可用如图 5-8 所示的二进制树状图进行描述和解释。

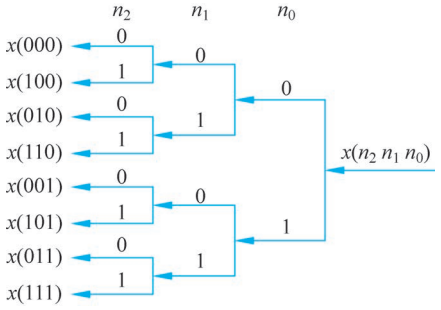


图 5-8　按序号分组形成倒位序的机制图

图 5-8 中，n_0、n_1、n_2 所在的列，正好对应了 $N=8$ 点序列的 3 次分组排队的过程，而最左边的 $x(000)$，$x(100)$，…，$x(111)$ 则是 $x(n)$ 三次分组后的顺序，观察括号内的三位二进制序号，若从右往左看，则为序列的正常顺序；而从左往右看，则为**倒位序**。

因此，倒位序就是 DIT 基-2 算法的输入顺序，由 $N=8$ 点推而广之，若序号 n 的正常二进制数为 $n_0n_1n_2\cdots n_{L-1}$，则 FFT 算法的输入为 $n_{L-1}\cdots n_2n_1n_0$，即序号正常二进制的倒位序，根据倒位序规律，可得 DIT 基-2 FFT 算法的输入顺序如表 5-3 所示。

表 5-3　DIT 基-2 FFT 算法倒位序规律表（$N=8$）

自 然 顺 序		倒　位　序	
自然顺序（n）	二 进 制 数	倒位序二进制数	倒位序顺序（$\hat{n}$）
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

FFT 实际运算中，一般先按自然顺序将输入序列 $x(n)$ 存入存储单元，为了得到倒位序的顺序，可以通过程序实现，以 $N=8$ 为例，若正常二进制数为 $(n_2n_1n_0)$，倒位序二进制数则为 $(n_0n_1n_2)$，用程序实现这一功能非常简单。

2. 原位运算

从图 5-6 可以看出，FFT 运算具有明显的规律，L 级运算中的每一级（每一列）运算都由 $N/2$ 个蝶形图组成，每一个基本蝶形运算完成如下迭代运算：

$$\begin{cases} X_m(k)=X_{m-1}(k)+X_{m-1}(j)W_N^r \\ X_m(j)=X_{m-1}(k)-X_{m-1}(j)W_N^r \end{cases} \tag{5-23}$$

式中，m 表示第 m 列，k、j 表示 FFT 蝶形运算图的第 k 行和第 j 行。

因此,式(5-23)对应的基本蝶形运算如图 5-9 所示,包括一次复数乘和两次复数加。

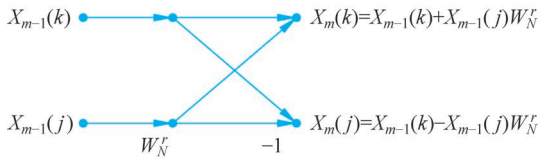


图 5-9 按时间抽取蝶形运算结果

根据图 5-6,第 $m-1$ 列的两个节点 k 和 j 的节点变量进行蝶形运算后,得到结果为第 m 列的 k、j 两节点的节点变量。在算法的实现上,FFT 蝶形运算是从左至右,按列进行运算。以 $N=8$ 为例,即先由输入序列 $x(n)$,分别采用第一级(列)的 4 个蝶形运算,求出 8 个数据;然后以这 8 个数据作为输入,采用第二级的 4 个蝶形运算图,求出 8 个数据;再采用第三列蝶形运算图,求出最终的 $X(k)$的全部 8 个值。

每一个基本蝶形运算的两个输出值存入下一级蝶形运算图的两个输入存储单元。因此,在编写程序代码时,某一列的 N 个数据存入存储单元之后,进入下一列蝶形运算,同样得到 N 个运算结果,仍可存储在同一组存储器中,直至计算出最后结果,无须开设更多的存储单元,这种优化和节省内存的方式称为 FFT 的**原位运算**(in-place computation)。

采用原位运算进行优化编程仅需 N 个存储单元,代码设计只需输入序列 $x(n)(n=0,1,\cdots,N-1)$的 N 个存储单元,以及用于存储每一列蝶形运算支路系数 $W_N^r(r=0,1,\cdots,N/2-1)$的 $N/2$ 个存储单元即可,降低了对内存资源的占用,提高了程序运行效率。

3. 蝶形运算两节点间的“距离”

DIT 基-2 FFT 算法是以蝶形图为单元的运算结构,在由计算机实现通用算法时,需要对各节点定位,以图 5-6 所示的 $N=8$ 点蝶形运算为例,序列 $x(n)$的输入顺序是倒位序,$X(k)$输出顺序为自然顺序。在该蝶形图中,从左至右共有 3 列,$m=1$ 时为第 1 级(即第 1 列),每个蝶形图两节点间的“距离”为 1;$m=2$ 时,第 2 级运算每个蝶形图两节点间的“距离”为 2;第 3 级每个蝶形图两节点间的“距离”为 4。

由此可得,当 $N=2^L$ 时,FFT 蝶形运算图输入为倒位序,输出正常顺序时,用 d 表示第 m 级运算每个蝶形图两节点间的“距离”,则 d 可用下式表示:

$$d=2^{m-1}$$

式中,m 表示 FFT 完整蝶形运算流图中列的序号,即第 m 列。

4. W_N^r 的确定

在 FFT 完整蝶形运算图中,第 m 级蝶形运算两节点间的“距离”为 2^{m-1},因而式(5-23)可写成如下形式:

$$\begin{cases} X_m(k)=X_{m-1}(k)+X_{m-1}(k+2^{m-1})W_N^r \\ X_m(k+2^{m-1})=X_{m-1}(k)-X_{m-1}(k+2^{m-1})W_N^r \end{cases} \tag{5-24}$$

观察图 5-6,分析 FFT 蝶形运算流程可以发现,运算流图中的系数因子 W_N^r,第一列仅一个因子 W_N^0;第二列有两个因子 W_N^0,W_N^2;……,以此类推,最后一列有 $N/2$ 个 W_N^r 系数因子,分别为 W_N^0,W_N^1,…,$W_N^{(\frac{N}{2}-1)}$。

对于程序实现,通过数学推导可以得到 r 值的递推算法,具体步骤如下:

(1) 将式(5-24)蝶形运算两节点的第一个节点标号 k 表示成 L 位二进制数；

(2) 将二进制数左移 $L-m$ 位，右边空位补零(即乘以 2^{L-m})；

(3) 将所得二进制数转换为十进制数，该数即为 r 值。

5.3.4　DIT 基-2 算法其他形式流图

图 5-6 所示的蝶形运算图描述了 FFT 快速算法的顺序结构和运算量，对于 FFT 算法流程图，若各节点的连接支路以及支路系数不变，则无论各节点和支路如何变形(改变空间位置)，其算法和运算量均不会改变。即仅改变节点和支路的空间位置，蝶形图的运算结果不会改变，但由于支路物理位置改变，输入数据的提取方式(顺序)和输出的顺序可能会发生变化。以图 5-6 为例，在不改变支路连接关系和支路乘数因子(旋转因子)的条件下，可以得到多种其他形式的流图。

1. 输入自然顺序、输出倒位序

对于图 5-6 所示蝶形运算流图，按如下两个步骤改变支路的物理位置(不改变节点的链接支路)。

(1) 将第 2 根水平线和第 5 根水平线平移互换位置，即 $x(4)$ 水平相连的所有节点和 $x(1)$ 水平相连的所有节点(包括输入数据和输出数据节点)互换位置。

(2) 将第 4 根水平线和第 7 根水平线平移互换位置，即 $x(6)$ 水平相连的所有节点和 $x(3)$ 水平相连的所有节点互换位置，该互换也包括输入数据和输出数据节点。

图 5-6 所示蝶形运算图经过上述两个步骤的变化，可得如图 5-10 所示的蝶形运算流图。根据碟形运算的规律，图 5-10 与图 5-6 的蝶形图运算结构完全相同，运算量也完全相同。不同点主要体现在如下两方面：

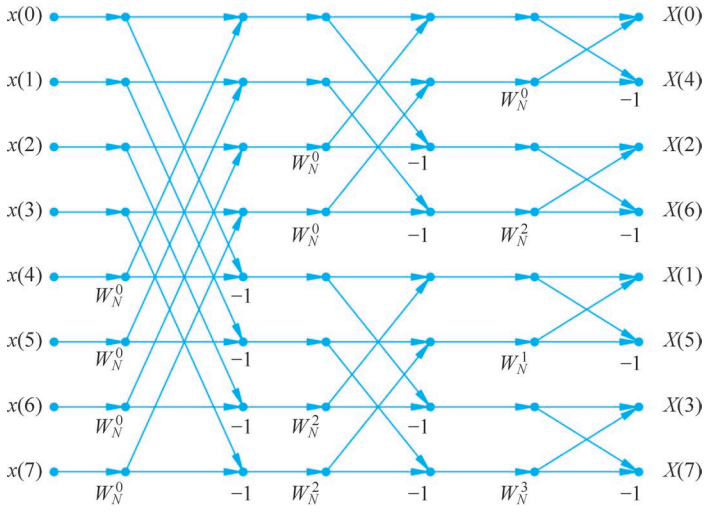


图 5-10　DIT 基-2 FFT 算法输入自然顺序、输出倒位序的 FFT 流图

(1) 图 5-10 和图 5-6 的输入顺序不一样，图 5-6 输入为倒位序，输出为自然顺序，而图 5-10 输入为自然顺序，输出却是倒位序。

(2) 支路乘数因子的顺序不同，图 5-6 的最后一列从上到下，使用系数的顺序为 W_N^0，W_N^1，W_N^2，W_N^3，前一列系数是后一列系数中的偶数幂系数(如 W_N^0，W_N^2)；而图 5-10 的最后

一列使用系数的顺序为 $W_N^0, W_N^2, W_N^1, W_N^3$，前一列系数是后一列系数的前一半，即仅使用前一半系数 W_N^0, W_N^2，顺序为 $W_N^0, W_N^0, W_N^2, W_N^2$。该运算流程图就是库利和图基最初得出的按时间抽取法算法流程图。

2. 其他形式的流图

如果在不改变图 5-10 各节点的支路连接数量和连接方式的前提下，继续改变某些节点和支路的空间位置，则可以得到输入和输出均为正常顺序的 FFT 蝶形运算图，如图 5-11 所示。该算法流程图的优点是输入和输出均为正常顺序，无须按倒位序排序。缺点是程序设计时不能进行原位运算，N 个输入数据必须开设 $2N$ 个以上的复数存储单元。

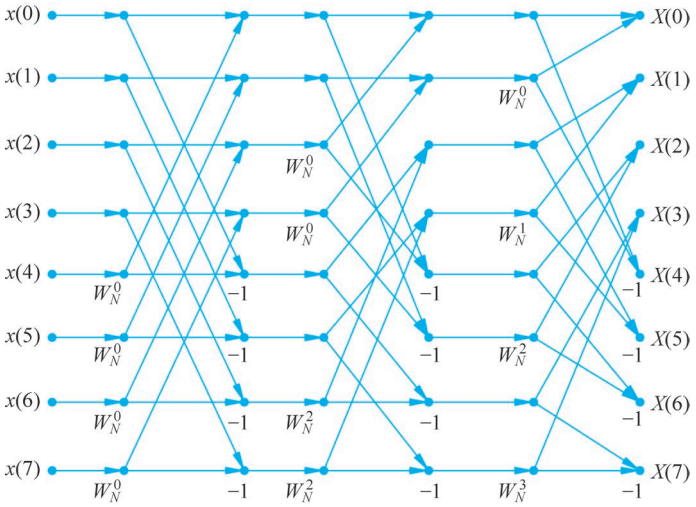


图 5-11　输入和输出均为正常顺序的 FFT 流图

FFT 蝶形运算图除了图 5-10 和图 5-11 所示的变化之外，还可以在保持各节点连接支路的数量和连接方式不变的前提下，再做进一步的变化，即图 5-6 和图 5-10，都可以进一步演变为其他形式，得到其他形式的 FFT 蝶形运算图。

【例 5-1】 用蝶形图计算 FFT 示例。

已知序列 $x(n)$ 如下：

$$x(n)=[1,2,1,1]$$

根据 DIT 基-2 快速算法原理，给出基于序列 $x(n)$ 的 DIT 基-2 快速运算蝶形图，并对序列 $x(n)$ 进行快速 DFT 计算(即 FFT 计算)。

解： 序列 $x(n)$ 为 4 点有限长序列，根据 DIT 基-2 算法原理，其完整蝶形图如图 5-12 所示。

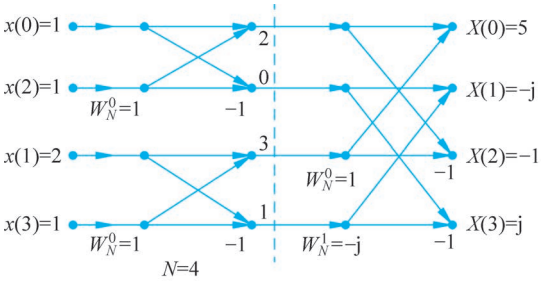


图 5-12　DIT 基-2($N=4$)FFT 蝶形运算图

$N=4=2^2$，共有两级蝶形运算（见图 5-12），支路系数（旋转因子）的数值已计算并标注在图 5-12 中，输入序列 $x(n)$ 的数据为倒位序。

虚线左边为第一级蝶形运算，从上往下四个节点的计算结果如下：

$$1+1=2,\quad 1-1=0,\quad 2+1=3,\quad 2-1=1$$

虚线右边为第二级蝶形运算，第二级运算结果如下：

$$X(0)=2+3=5,\quad X(1)=0-\mathrm{j}=-\mathrm{j}$$

$$X(2)=2-3=-1,\quad X(3)=0+\mathrm{j}=\mathrm{j}$$

计算过程表明，4 点蝶形运算中，由于 $W_4^0=1$，$W_4^1=-\mathrm{j}$，4 点 FFT 实际上没有乘法运算，非常节省运算量。

5.4　按频率抽取（DIF）基-2 FFT 算法

在 DIT 基-2 算法中，对序列 $x(n)$ 的分组采用按序号 n 的奇偶分解序列，得到 FFT 算法。如果对 $x(n)$ 不按奇偶分组，而是直接将序列的前一半和后一半分为两组，得到两个新的子序列，然后对新的子序列继续均分，一直分解到所有子序列为 2 点为止，这样就得到了一种新的 FFT 快速算法，这就是按频率抽取 FFT 算法，即 **DIF 基-2 FFT 算法**。

该方法对 $x(n)$ 按序号的中间位置对半均分，实际上形成了输出 $X(k)$ 按频域序号 k 的奇、偶分组排序。该算法由桑德-图基首先提出和完善，故又称为桑德-图基算法。

5.4.1　DIF 算法原理

设序列 $x(n)$ 的点数为 $N=2^L$，L 为正整数，先将序列 $x(n)$ 按前一半、后一半分为两个子序列。因此，DFT 公式可进行如下形式的变化：

$$\begin{aligned}X(k)&=\sum_{n=0}^{N-1}x(n)W_N^{nk}=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{nk}+\sum_{n=\frac{N}{2}}^{N-1}x(n)W_N^{nk}\\&=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{nk}+\sum_{n=0}^{\frac{N}{2}-1}x\left(n+\frac{N}{2}\right)W_N^{(n+\frac{N}{2})k}\\&=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)W_N^{Nk/2}\right]W_N^{nk},\quad k=0,1,\cdots,N-1\end{aligned}\tag{5-25}$$

由于 $W_N^{\frac{Nk}{2}}=(-1)^k$，代入式(5-25)，可得

$$X(k)=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+(-1)^kx\left(n+\frac{N}{2}\right)\right]W_N^{nk},\quad k=0,1,\cdots,N-1\tag{5-26}$$

若 k 为偶数，$(-1)^k=1$；若 k 为奇数，$(-1)^k=-1$，因此，根据频域序号 k 的奇偶情况可将式(5-26)表示为

$$X(2r)=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_N^{2nr}=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_{N/2}^{nr}\tag{5-27}$$

$$X(2r+1)=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^{n(2r+1)}=\sum_{n=0}^{\frac{N}{2}-1}\left\{\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^n\right\}W_{N/2}^{nr} \tag{5-28}$$

令

$$\begin{cases}x_1(n)=x(n)+x\left(n+\dfrac{N}{2}\right)\\ x_2(n)=\left[x(n)-x\left(n+\dfrac{N}{2}\right)\right]W_N^n\end{cases}\quad n=0,1,\cdots,\frac{N}{2}-1 \tag{5-29}$$

即 $x(n)$前半区间某一序列值与间距为 $N/2$ 的后半区间的对应序列值之和组成了新的序列 $x_1(n)$,$x(n)$前半区间某一序列值与间距为 $N/2$ 的后半区间对应的序列值之差再与 W_N^n 相乘组成了新的序列 $x_2(n)$。新序列 $x_1(n)$和 $x_2(n)$均为 $N/2$ 点序列。

于是,式 (5-27)和式(5-28)可简化为

$$\begin{cases}X(2r)=\sum\limits_{n=0}^{\frac{N}{2}-1}x_1(n)W_{N/2}^{nr}\\ X(2r+1)=\sum\limits_{n=0}^{\frac{N}{2}-1}x_2(n)W_{N/2}^{nr}\end{cases}\quad r=0,1,\cdots,\frac{N}{2}-1 \tag{5-30}$$

式(5-30)中的第一式是关于序列 $x_1(n)$的 $N/2$ 点 DFT 运算,第二式为关于序列 $x_2(n)$的 $N/2$ 点 DFT 运算。

式(5-29)所表示的运算关系可以用如图 5-13 所示的蝶形运算图表示。

这是 DIF 基本蝶形运算图,与图 5-1 进行对比可以看出,它与 DIT 蝶形图类似,DIF 的每一个基本蝶形运算仅包括一次复数乘法及两次复数加法。

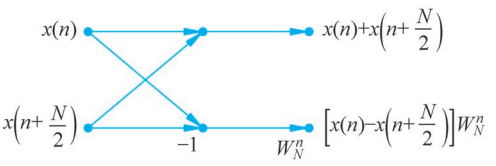


图 5-13 DIF 基本蝶形运算图

根据图 5-13,若 $N=2^3=8$ 时,以 $n=0,1,2,3$ 代入,可以得出 DIF 基-2 第一次分解的运算过程,结合式(5-30)可得第一次分解 DIF 蝶形运算如图 5-14 所示,N 点 DFT 分解为左边的 4 个基本蝶形图和两个 $N/2$ 点的 DFT。第一次分解完成之后,其计算过程是从左往右按列计算,即第一列按 4 个蝶形图计算,右边 DFT 方框依然按 DFT 公式计算。

与 DIT 算法的推导过程类似,第一次分解以后,图 5-14 的左边包含了 4 个基本蝶形运算,即 $x(0)$与 $x(4)$、$x(1)$与 $x(5)$、$x(2)$与 $x(6)$、$x(3)$与 $x(7)$组成了 4 个基本蝶形运算,每一个蝶形运算包含一次复数乘法。

至此,一个 N 点 DFT 分解成了两个 $N/2$ 点 DFT,若直接计算 $N/2$ 点 DFT,则每一个 $N/2$ 点 DFT 的复数乘法次数如下:

$$\left(\frac{N}{2}\right)^2=\frac{N^2}{4}$$

复数加法次数如下:

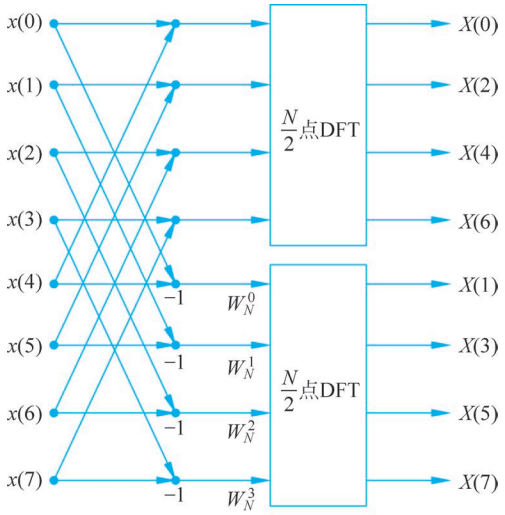


图 5-14　N 点 DFT 分解为两个 N/2 点 DFT(DIF N=8)

$$\frac{N}{2}\left(\frac{N}{2}-1\right)$$

因此,图 5-14 中两个 $N/2$ 点 DFT 共需$\frac{N^2}{2}$次复数乘法和 $N\left(\frac{N}{2}-1\right)$次复数加法。此外,左边第一列 $N/2$ 个蝶形运算共有 $N/2$ 次复数乘法和 $2\times N/2=N$ 次复数加法。因而第一次分解全部完成之后,复数总运算量如下。

复数乘法次数:

$$\frac{N^2}{2}+\frac{N}{2}+\frac{N(N+1)}{2}\approx\frac{N^2}{2}$$

复数加法次数:

$$N\left(\frac{N}{2}-1\right)+N=\frac{N^2}{2}$$

因此,DIF 经过第一次分解,其运算量与 DIT 的第一次分解的运算量完全一样,约为直接采用 DFT 运算量的一半。

既然长序列分解为短序列可以降低运算量,那么接下来继续进行第二次分解,与 DIT 的分解类似,由于 $N=2^L$,故 $N/2$ 仍是偶数,对 $x_1(n)$和 $x_2(n)$两个 $N/2$ 点的序列按前一半后一半继续分解,得到两个 $N/4$ 点的子序列。以 $x_1(n)$为例,按序号从中间均分为两组,每组 $N/4$ 点。

于是,一个 $N/2$ 点 DFT 被分解为两个 $N/4$ 点 DFT。当 $N=8$ 时,分解过程如图 5-15 所示。

对于 $N=2^L$ 序列,分解共需进行 L 次,即直到序列被分解为每一个子序列均为 2 点为止。2 点 DFT 正好组成一个基本蝶形运算,而且乘数因子 $W_N^0=1$,无须乘法运算。当 $N=8$ 时,DIF 基-2 FFT 完整蝶形运算图如图 5-16 所示。

图 5-16 称为 DIF 基-2 FFT 完整蝶形运算图($N=8$ 时),与 DIT 快速算法类似,由 $x(n)$计算 $X(k)$是从左至右按列进行计算,因此,8 点 FFT 共需要 3 级计算。

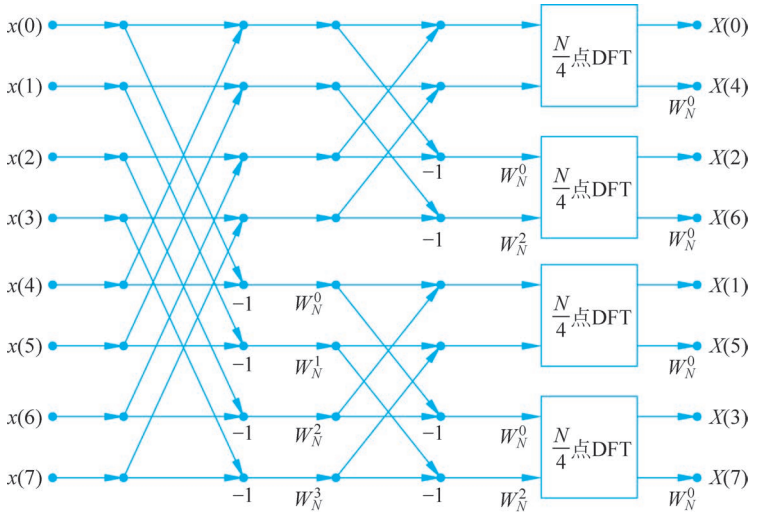


图 5-15　N 点 DFT 分解为 4 个 N/4 点 DFT(DIF N=8)

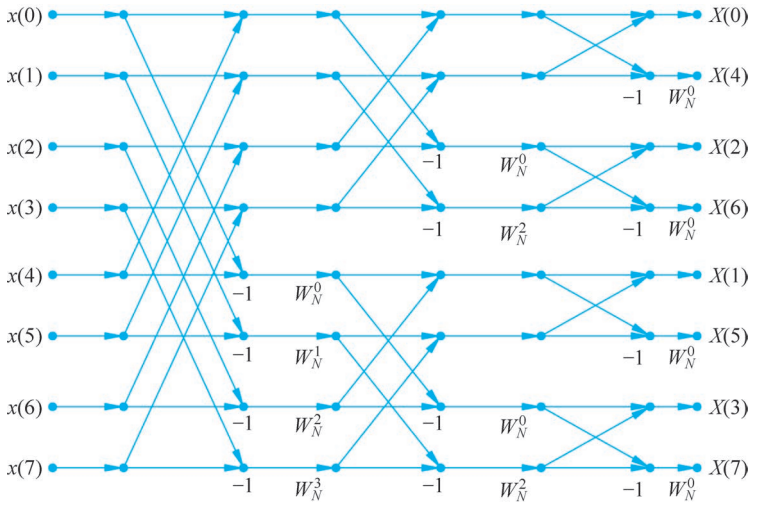


图 5-16　DIF 基-2 FFT 算法流图(N=8)

第 1 级运算,按左边第一列 4 个蝶形图进行计算,得到 8 个中间值;第 2 级运算,以第 1 级的结果为输入,按第二列 4 个蝶形图计算,得到 8 个中间值;以此类推进行第 3 级运算,直至完成最后一级运算,得出全部 $X(k)$,$k=0,1,2,\cdots,N-1$。

若 $N=8$,上述 3 级计算中,每一级计算均包含 $\frac{N}{2}=4$ 次复数乘法运算,即每一个基本蝶形运算包含一个乘法,因此,依据图 5-16 所示的 FFT 蝶形运算计算 DFT,仅需 12 次复数乘法运算(包括乘 $W_N^0=1$)。

上述 3 级蝶形运算中,每一级基本蝶形图的系数也具有如下规律。

(1) 第 1 级蝶形运算的系数为 W_N^0,W_N^1,W_N^2,W_N^3。

(2) 第 2 级蝶形运算的系数为 W_N^0,W_8^2。

(3) 第 3 级蝶形运算的系数均为 W_N^0。

以此类推，也可以得出 $N=2^L$，DIF 基-2 FFT 快速算法完整的蝶形运算流图以及 FFT 的 L 级运算过程。

5.4.2　DIF 基-2 算法的特点

DIF 基-2 算法具有与 DIT 算法类似的特点，如根据序列分解过程及原理，可以很方便地得出 $N=4$ 点和 $N=8$ 点 FFT 流程图的递推关系，进而可得出 $N=2^{L-1}$ 点与 $N=2^L$ 点 FFT 算法流图之间的递推关系，以及 DIF 基-2 FFT 算法也可实现原位运算等。

1. DIF 的原位运算

从图 5-16 可以看出，DIF 与 DIT 的 FFT 算法结构相似，也具有明显的规律性，当 $N=8$ 时，从左至右，共包括三级蝶形运算。若 $N=2^L$，则包括 L 级(列)运算，每级都包含 $N/2$ 个基本蝶形图。每一个蝶形结构完成如下基本迭代运算：

$$\begin{aligned} X_m(k) &= X_{m-1}(k)+X_{m-1}(j) \\ X_m(j) &= [X_{m-1}(k)-X_{m-1}(j)]W_N^r \end{aligned} \tag{5-31}$$

式中，m 表示 DIF 基-2 完整蝶形运算的第 m 列，k、j 分别表示第 k 行和第 j 行，式(5-31)可以用如图 5-17 所示的蝶形图表示，共包括一次复数乘法和两次复数加法。

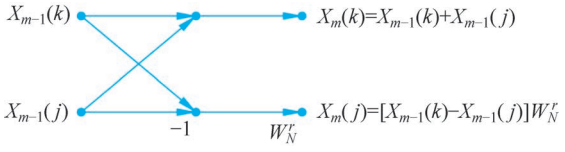


图 5-17　DIF 蝶形运算结构

在编程实现 DIF 的 FFT 算法时，同样可以实现原位运算，即仅需 N 个存储单元和 $N/2$ 个乘数因子存储单元即可实现从输入到输出的运算过程。

2. 两节点间的“距离”

与 DIT 基-2 FFT 算法类似，编程实现算法时，也需要对各节点进行定位，从图 5-16 可以看出，其输入数据为正常顺序，输出为倒位序。该蝶形图从左至右共有 3 级，$m=1$ 时为第 1 级(即第 1 列)，每个蝶形图两节点间的“距离”为 4；$m=2$ 时为第 2 级(第 2 列)，每个蝶形图两节点间的“距离”为 2；第 3 级每个蝶形图两节点间的“距离”为 1。

若 $N=2^L$，用 d 表示第 m 级运算蝶形图两节点间的“距离”，则 d 可用下式表示：

$$d=2^{L-m}=\frac{N}{2^m} \tag{5-32}$$

式中，$N=2^L$，m 表示 FFT 完整蝶形运算流图中列的顺序，即第 m 列。

3. W_N^r 的计算

对 DIF 的完整蝶形图进行第 m 级运算时，基本蝶形运算两节点间的“距离”为 2^{L-m}，因而第 m 级的蝶形计算可表示为

$$\begin{cases} X_m(k)=X_{m-1}(k)+X_{m-1}\left(k+\dfrac{N}{2^m}\right) \\ X_m\left(k+\dfrac{N}{2^m}\right)=\left[X_{m-1}(k)-X_{m-1}\left(k+\dfrac{N}{2^m}\right)\right]W_N^r \end{cases} \tag{5-33}$$

用代码实现 DIF 基-2 算法时，对于 $m=0,1,\cdots,L$ 的不同级运算，可以得出支路乘数因子 W_N^r 中指数 r 的递推算法，具体步骤如下：

(1) 将式(5-33)蝶形运算两节点中的第一个节点标号 k 表示为 L 位二进制数；

(2) 该二进制数左移$(m-1)$位，即乘以 2^{m-1}，移位时右边空位补零；

(3) 将所得二进制数转为十进制数，即为 r 值。

5.4.3 DIT 与 DIF 的比较

1. DIT 与 DIF 的不同点

(1) DIT 基-2 FFT 的输入序列是倒位序，输出是自然顺序，DIF 正好相反，但这不是绝对的，无论是 DIT 还是 DIF，都可以通过改变蝶形运算图支路的空间位置实现输入或输出的重新排列，两者的输入或输出顺序都可以改变为自然顺序或倒位序。

(2) DIT 的基本蝶形(见图 5-1)与 DIF 的基本蝶形(见图 5-17)不同，DIT 是先乘法运算，再进行代数和运算，而 DIF 则是先求代数和，再乘法运算。这是 DIT 与 DIF 本质的不同点。

2. DIT 与 DIF 的相同点

(1) 两者的运算量相同，即都有 L 级(列)运算，每一级运算均包括 $N/2$ 个基本蝶形运算，总共需要 $m_T=\frac{N}{2}\log_2 N$ 次复数乘法和 $a_T=N\log_2 N$ 次复数加法。

(2) DIT 基-2 FFT 和 DIF 基-2 FFT 都可通过编程实现原位运算。

(3) DIT 与 DIF 的基本蝶形运算流图互为转置关系。若将 DIT 的基本蝶形运算流图进行转置，就得到 DIF 的基本蝶形图，反之亦然。转置是指将流图的所有支路都改变方向，输入和输出互换。经转置运算，由图 5-1 可以得到图 5-17，反之也可以由图 5-17 得到图 5-1。

由此可知，根据输入和输出是按自然顺序还是按倒位序排列的不同，有四种原位运算的 FFT 算法流图，DIT 与 DIF 各两种，DIT 与 DIF 算法的特点如表 5-4 所示。

表 5-4 DIT 与 DIF 算法的特点($N=2^L$)

	按时间抽取(DIT)	
	输入自然顺序、输出倒位序	输入倒位序、输出自然顺序
m 级蝶形图两节点间的距离	$2^{L-m}=\frac{N}{2^m}$	2^{m-1}
第 m 级蝶形运算公式	$X_m(k)=X_{m-1}(k)+X_{m-1}\left(k+\frac{N}{2^m}\right)W_N^r$ $X_m\left(k+\frac{N}{2^m}\right)=X_{m-1}(k)-X_{m-1}\left(k+\frac{N}{2^m}\right)W_N^r$	$X_m(k)=X_{m-1}(k)+X_{m-1}(k+2^{m-1})W_N^r$ $X_m(k+2^{m-1})=X_{m-1}(k)-X_{m-1}(k+2^{m-1})W_N^r$
r 值计算(W_N^r)	1. k 表示为 L 位二进制数 2. 右移 $L-m$ 位，左边空出位补 0 3. 补 0 后倒位序排列即为 r 值	1. k 表示为 L 位二进制数 2. 左移 $L-m$ 位，右边空出位补 0 3. 补 0 后的结果即为 r 值
	按频率抽取(DIF)	
	输入自然顺序、输出倒位序	输入倒位序，输出自然顺序
m 级蝶形图两节点间的距离	$2^{L-m}=\frac{N}{2^m}$	2^{m-1}

续表

	按频率抽取(DIF)	
	输入自然顺序、输出倒位序	输入倒位序，输出自然顺序
第 m 级蝶形运算公式	$X_m(k)=X_{m-1}(k)+X_{m-1}\left(k+\frac{N}{2^m}\right)W_N^r$ $X_m\left(k+\frac{N}{2^m}\right)=$ $\left[X_{m-1}(k)-X_{m-1}\left(k+\frac{N}{2^m}\right)\right]W_N^r$	$X_m(k)=X_{m-1}(k)+X_{m-1}(k+2^{m-1})W_N^r$ $X_m(k+2^{m-1})=[X_{m-1}(k)-X_{m-1}(k+2^{m-1})]W_N^r$
r 值计算(W_N^r)	1. k 表示为 L 位二进制数 2. 左移 $m-1$ 位，右边空出位补 0 3. 补 0 后的结果即为 r 值	1. k 表示为 L 位二进制数 2. 右移 $m-1$ 位，左边空出位补 0 3. 补 0 后倒位序排列即为 r 值

5.4.4　IDFT 快速算法

离散傅里叶逆变换的快速运算称为 IFFT。DFT 和 IDFT 的算法结构类似，差异非常小。

DFT 正变换：

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk},\quad k=0,1,\cdots,N-1 \tag{5-34}$$

DFT 逆变换：

$$x(n)=\mathrm{IDFT}[X(k)]=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-nk},\quad k=0,1,\cdots,N-1 \tag{5-35}$$

正变换和逆变换的差别主要表现在两方面，一是正逆变换中的 W_N 指数的符号相反；二是逆变换算法包含常数因子 $1/N$，而正变换的因子为 1。

由于正变换和逆变换算法的内核要素相同，因此，无论是 DIT 基-2 FFT 还是 DIF 基-2 FFT 算法，若解决了逆变换与正变换的两个不同点，则正变换的快速算法也能用于逆变换。

解决上述指数符号和系数的差异问题，有如下两种方法：

(1) 研究 FFT 快速算法运算流图，解决指数符号问题，实现 IFFT 算法；

(2) 研究 DFT 和 IDFT 公式，利用复数性质改逆变换公式，实现 IFFT 运算。

1. 基于运算流图的 IFFT

(1) 指数符号问题。

由于正变换和逆变换的算法结构基本相同，以 DIT 基-2 FFT 算法为例，根据 DIT 基-2 快速算法原理，只需将正变换快速算法蝶形运算中的支路增益系数 W_N^{nk} 置换为 W_N^{-nk} 即可。类似地，对于 DIF 基-2 快速算法，对其每一级蝶形做同样的改进之后也可以用于 IFFT 运算。

(2) 系数 $1/N$ 的处理。

在乘数因子 W_N^{nk} 解决之后，可以将所得的运算结果均乘以常数 $1/N$，或者在每一级蝶形运算的各支路上增加支路系数 1/2，则可以解决逆变换系数 $1/N$ 的问题。

也就是说，对正变换的 FFT 运算进行上述改进之后，按时间抽取和按频率抽取的 FFT 算法均可用于逆变换的计算。在运算中注意将按时间抽取的流图(见图 5-6)或者按频率抽

取的流图(见图 5-16)中的支路乘数因子 W_N^{nk} 改变为 W_N^{-nk},并在每列(级)运算中乘以 1/2,$\left(\frac{1}{2}\right)^L=\frac{1}{N}$。以 $N=8$ 点的 DIT 基-2 FFT 为例,IFFT 算法流图如图 5-18 所示。

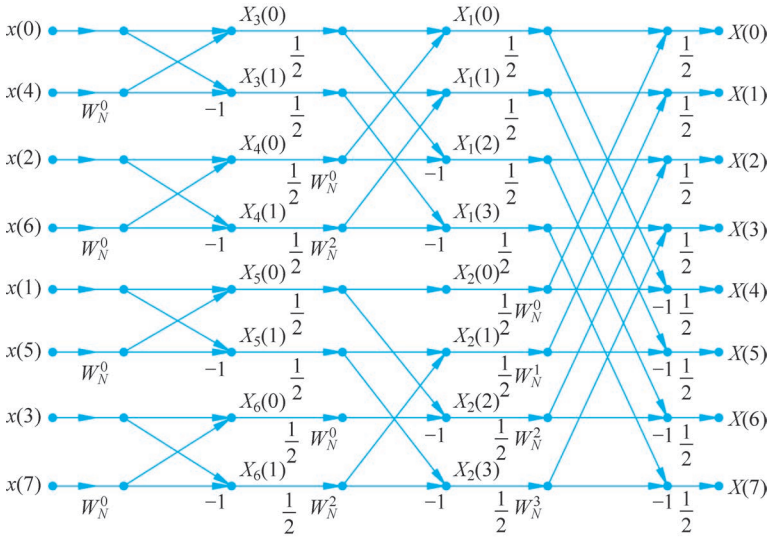


图 5-18 DIT 逆变换快速算法完整蝶形图($N=8$)

2. 基于公式改进的 IFFT

根据上述 IFFT 算法的实现过程可知,对正变换 FFT 的代码进行相应的修改即可用于逆变换的实现。这种方法虽然工作量小、实现方便,但需要修改 FFT 正变换程序的内部代码,即修改 W_N^{nk} 的指数符号以及对每一级运算乘以 1/2 才能实现逆变换的快速算法。

显然,修改既有程序内部的代码对程序员的要求较高,稍有不慎,容易引起其他问题。如果不修改 DFT 正变换的内部代码,直接将正变换程序用于逆变换运算,则可以从 DFT 的公式入手,通过对变换公式进行适当变换,利用复数的性质,使逆变换与正变换的因子 W_N^{nk} 的符号相同,这样就无须修改 FFT 正变换程序的内部代码,也可以直接将正变换程序用于逆变换计算。

先对 DFT 逆变换进行共轭运算

$$x^*(n)=\frac{1}{N}\sum_{k=0}^{N-1}X^*(k)W_N^{nk}$$

再次进行共轭运算

$$x(n)=\frac{1}{N}\left[\sum_{k=0}^{N-1}X^*(k)W_N^{nk}\right]^*=\frac{1}{N}\{\mathrm{DFT}[X^*(k)]\}^* \tag{5-36}$$

此时,式(5-36)的 W_N^{nk} 指数的符号与正变换完全一致,因此,若先对 $X(k)$进行共轭运算,就可以直接应用正变换快速算法的程序进行逆变换计算,然后将变换结果再进行一次共轭运算,并乘以 $1/N$,就完成了逆变换的快速运算。

该方法的优点是无须对正变换 FFT 程序的内部代码进行修改,只需运算前先对逆变换的输入数据进行共轭运算,再将计算结果进行一次共轭运算并除以 N,就完成了求逆变换 $x(n)$的全部计算。

5.5　基-4 FFT 算法

基-2 FFT 算法的基本方法是先将 N 点序列一分为二，一个 N 点 DFT 被分解为两个 $N/2$ 点 DFT 的组合，再将 $N/2$ 点 DFT 又一分为二，分解为 4 个 $N/4$ 点的 DFT，直至全部分解为 2 点 DFT 为止。基-4 FFT 的思路与基-2 FFT 的思路基本类似，不同点是前者将 N 点序列一分为四，再对各子序列继续一分为四，直至分解到全部子序列为 4 点为止。

5.5.1　基-4 FFT 原理

根据基-2 FFT 算法原理，对于基-4 FFT 算法，序列 $x(n)$ 的点数应满足 $N=4^L$。基-4 FFT 算法的基本思路是将 N 点序列一分为四，即一个 N 点 DFT 分解为四个 $N/4$ 点 DFT 的线性组合，再将每一个 $N/4$ 点 DFT 又一分为四，分解为 4 个 $N/16$ 点的 DFT，直至全部分解为 4 点 DFT 为止。而对于 4 点 DFT 运算，则采用基-2 FFT 运算，并以此作为基-4 FFT 最后一次分解的基本运算单元。因此，类似于 DIT 基-2 的分解过程，DIT 基-4 快速算法先将长度为 $N=4^L$ 的序列 $x(n)$，按如下方式分为四个子序列：

$$\begin{cases} x_0(r)=x(4r) \\ x_1(r)=x(4r+1) \\ x_2(r)=x(4r+2) \\ x_3(r)=x(4r+3) \end{cases}, \quad r=0,1,\cdots,\frac{N}{4}-1 \tag{5-37}$$

于是，DFT 公式可以改变为如下形式：

$$\begin{aligned} X(k)&=\mathrm{DFT}[x(n)]=\sum_{n=0}^{N-1}x(n)W_N^{nk} \\ &=\sum_{((n))_4=0}^{N-1}x(n)W_N^{nk}+\sum_{((n))_4=1}^{N-1}x(n)W_N^{nk}+\sum_{((n))_4=2}^{N-1}x(n)W_N^{nk}+\sum_{((n))_4=3}^{N-1}x(n)W_N^{nk} \\ &=\sum_{r=0}^{\frac{N}{4}-1}x(4r)W_N^{4rk}+\sum_{r=0}^{\frac{N}{4}-1}x(4r+1)W_N^{(4r+1)k}+\sum_{r=0}^{\frac{N}{4}-1}x(4r+2)W_N^{(4r+2)k}+\sum_{r=0}^{\frac{N}{4}-1}x(4r+3)W_N^{(4r+3)k} \\ &=\sum_{r=0}^{\frac{N}{4}-1}x_0(r)W_N^{4rk}+\sum_{r=0}^{\frac{N}{4}-1}x_1(r)W_N^{(4r+1)k}+\sum_{r=0}^{\frac{N}{4}-1}x_2(r)W_N^{(4r+2)k}+\sum_{r=0}^{\frac{N}{4}-1}x_3(r)W_N^{(4r+3)k} \end{aligned}$$

式中，$((n))_4$ 表示 n 除以 4 的余数。根据 W_N 的可约性，上式可以进一步化为

$$X(k)=\sum_{r=0}^{\frac{N}{4}-1}x_0(r)W_{N/4}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{4}-1}x_1(r)W_{N/4}^{rk}+W_N^{2k}\sum_{r=0}^{\frac{N}{4}-1}x_2(r)W_{N/4}^{rk}+W_N^{3k}\sum_{r=0}^{\frac{N}{4}-1}x_3(r)W_{N/4}^{rk} \tag{5-38}$$

根据 $N/4$ 点 DFT 公式，式(5-38)可以表示为

$$X(k)=X_0(k)+W_N^kX_1(k)+W_N^{2k}X_2(k)+W_N^{3k}X_3(k), \quad 0\leqslant k\leqslant\frac{N}{4}-1 \tag{5-39}$$

式中，$X_0(k)$、$X_1(k)$、$X_2(k)$、$X_3(k)$ 分别为 $N/4$ 点时域序列 $x_0(r)$、$x_1(r)$、$x_2(r)$、$x_3(r)$ 对应的 DFT，即

$$X_0(k)=\sum_{r=0}^{\frac{N}{4}-1}x_0(r)W_{N/4}^{rk},\quad X_1(k)=\sum_{r=0}^{\frac{N}{4}-1}x_1(r)W_{N/4}^{rk}$$

$$X_2(k)=\sum_{r=0}^{\frac{N}{4}-1}x_2(r)W_{N/4}^{rk},\quad X_3(k)=\sum_{r=0}^{\frac{N}{4}-1}x_3(r)W_{N/4}^{rk}$$

类似 DIT 基-2 算法的第一次分解，式(5-39)只能计算出 $X(k)$ 前四分之一的值，$X(k)$ 其余值的计算需根据 W_N 的周期性以及有限长序列隐含的周期性，由式(5-39)直接得出：

$$X\left(k+\frac{N}{4}\right)=X_0(k)+W_N^{k+\frac{N}{4}}X_1(k)+W_N^{2\left(k+\frac{N}{4}\right)}X_2(k)+W_N^{3\left(k+\frac{N}{4}\right)}X_3(k)$$

$$X\left(k+\frac{N}{2}\right)=X_0(k)+W_N^{k+\frac{N}{2}}X_1(k)+W_N^{2\left(k+\frac{N}{2}\right)}X_2(k)+W_N^{3\left(k+\frac{N}{2}\right)}X_3(k)$$

$$X\left(k+\frac{3N}{4}\right)=X_0(k)+W_N^{k+\frac{3N}{4}}X_1(k)+W_N^{2\left(k+\frac{3N}{4}\right)}X_2(k)+W_N^{3\left(k+\frac{3N}{4}\right)}X_3(k)$$

$$0\leqslant k\leqslant\frac{N}{4}-1$$

根据 W_N 的特性，上述公式可以简化为如下形式：

$$X\left(k+\frac{N}{4}\right)=X_0(k)-\mathrm{j}W_N^kX_1(k)-W_N^{2k}X_2(k)+\mathrm{j}W_N^{3k}X_3(k)\tag{5-40}$$

$$X\left(k+\frac{N}{2}\right)=X_0(k)-W_N^kX_1(k)+W_N^{2k}X_2(k)-W_N^{3k}X_3(k)\tag{5-41}$$

$$X\left(k+\frac{3N}{4}\right)=X_0(k)+\mathrm{j}W_N^kX_1(k)-W_N^{2k}X_{23}(k)-\mathrm{j}W_N^{3k}X_3(k)\tag{5-42}$$

$$0\leqslant k\leqslant\frac{N}{4}-1$$

由式(5-39)和式(5-40)～式(5-42)可得如图 5-19 所示的 DIT 基-4 快速算法基本蝶形运算图。

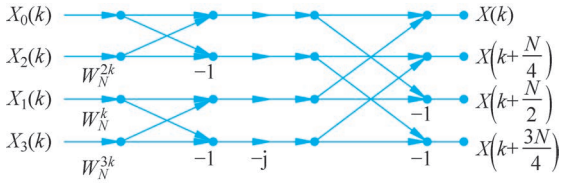


图 5-19 DIT 基-4 快速算法基本蝶形图

为了直观和深入理解基-4 FFT 算法原理，现以 $N=16$ 为例分析基-4 快速运算流图的规律。由于 $N=16=4^L$，即 $L=2$，根据式(5-39)和式(5-40)～式(5-42)对序列 $x(n)$ 进行第一次分解，分别以 $k=0,1,2,3$ 代入，可得当 $N=16$ 时第一次分解的蝶形运算图，如图 5-20 所示。经过第一次分解之后，输出为正常顺序，输入顺序为四进制倒位序。

由于 $N=16=4^2$ 仅需 2 次分解，经过第一次分解之后，对于 4 个 $\frac{N}{4}$ 点的 DFT，各子序列长度均为 4 点，正好是 DIT 基-4 快速运算的基本单元。根据基-4 算法的原则，4 点长的子序列 DFT 采用基-2 FFT 运算，这是基-4 FFT 的最后一次分解。根据图 5-19 或式(5-39)和式(5-40)～式(5-42)的运算规律，这时 k 只有一个取值，即 $k=0$，由于需按基-2 运算分组，

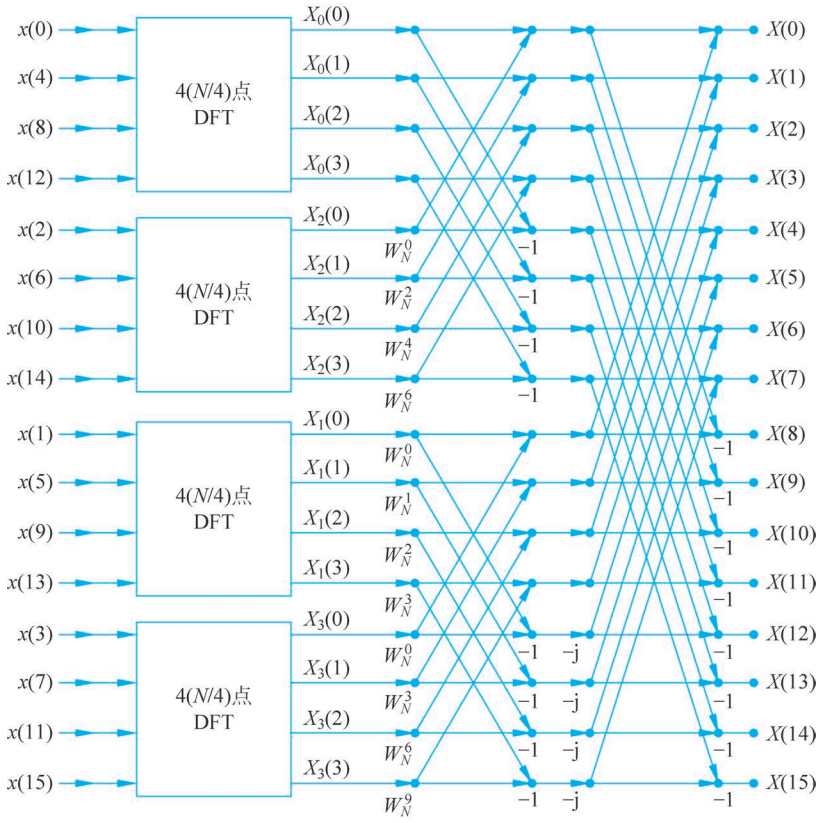


图 5-20　DIT 基-4 快速算法第一次分解蝶形图

因此需要按倒位序排列。以图 5-20 左上角第一个 4 点 DFT 为例，其运算可展开如下：

$$\begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ X_0(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^2 & W_4^1 & W_4^3 \\ W_4^0 & W_4^0 & W_4^2 & W_4^2 \\ W_4^0 & W_4^2 & W_4^3 & W_4^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(2) \\ x(1) \\ x(3) \end{bmatrix}$$

由于 $W_4^0=1$、$W_4^1=-\mathrm{j}$、$W_4^2=-1$、$W_4^3=\mathrm{j}$，并考虑到 2 点 DFT 仍需按偶序号和奇序号排列，因此矩阵表达式可以进一步简化为如下形式：

$$\begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ X_0(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & \mathrm{j} & -\mathrm{j} \\ 1 & -1 & 1 & -1 \\ 1 & -1 & \mathrm{j} & -\mathrm{j} \end{bmatrix} \begin{bmatrix} x(0) \\ x(2) \\ x(1) \\ x(3) \end{bmatrix} \tag{5-43}$$

根据式(5-43)，可得第一列左上角 4 点基-4 快速算法的蝶形运算过程如图 5-21 所示。

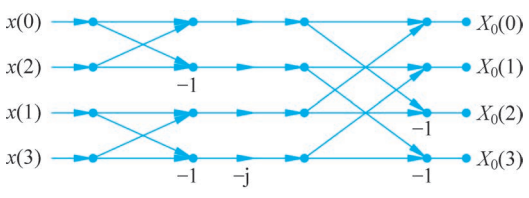


图 5-21　基-4 FFT 基本运算的信号流图

蝶形图的计算是从左往右按列依次运算，根据图 5-21 可知，基-4 FFT 的第一级运算具有如下特点：

(1) 一个基-4 算法包含有 2 点 DFT 蝶形结构。

(2) 第一级 4 点基-4 FFT 运算完全不需要复数乘法运算,虽然流图中有支路系数 $-j$,但实际上这是交换复数的实部和虚部,不用乘法运算也可完成。

根据第一次分解和第二次分解过程可以得出当 $N=16$ 时 DIT 基-4 FFT 的完整蝶形运算流图如图 5-22 所示,序列 $x(n)$ 经过两次分解之后,输出为正常顺序,输入为四进制倒位序,图中虚线框内为一个 4 点基-4 FFT 流图的基本单元。

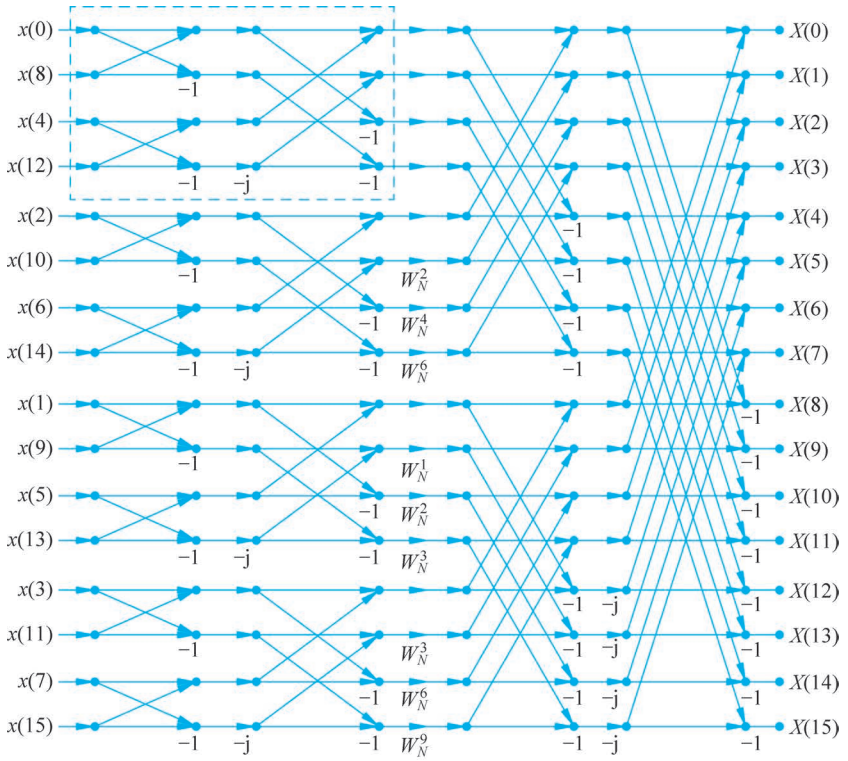


图 5-22 DIT 基-4 FFT 流图

5.5.2 基-4 FFT 运算量

根据基-4 FFT 算法原理,对于第一级基本的 4 点 FFT,实际上无须乘法运算。根据图 5-22 所示的算法流图,支路有乘旋转因子才包含复数乘法运算,每一个 4 点 DFT 仅有 3 次需乘旋转因子,另一乘 $W_N^0=1$。每一级基-4 FFT 包含 $N/4$ 个 4 点 DFT,因此,每一级共包含 $3\times N/4$ 次复数乘法运算,由于 $N=4^L$,共有 L 级运算,$L=(\log_2 N)/2$,第一级运算的因子为 ±1 和 j,实际上没有乘法运算,因此,基-4 FFT 算法总的复数乘法次数如下。

(1) 复数乘法次数。

$$m_{\mathrm{T}}=\frac{3}{4}N(L-1)=\frac{3}{4}N\left(\frac{1}{2}\log_2 N-1\right)\approx\frac{3}{8}N\log_2 N,\quad L\gg1 \tag{5-44}$$

由此可知,基-4 FFT 与基-2 FFT 相比乘法运算量更少,运算效率更高。由于一些特定的乘数因子无须相乘,如当 $N=16$ 时,$W_{16}^0=1$、$W_{16}^4=-j$、$W_{16}^8=-1$ 等,这些乘数因子实际上无须乘法运算,故根据图 5-22,当 $N=16$ 时,基-4 FFT 仅需 8 次复数乘法运算。

对于复数加法运算量，根据运算流图 5-22 可知，求和节点和基-2 FFT 的数量相等，由此可得，基-4 FFT 所需的复数加法次数和基-2 FFT 完全相等。

(2) 复数加法次数。

$$a_{\mathrm{T}} = N\log_2 N \tag{5-45}$$

5.6　分裂基 FFT 算法

基-2 FFT 算法的复数乘法运算量为$\frac{N}{2}\log_2 N$，为了进一步寻找更高效的算法，1984 年，有学者在基-2 和基-4 算法的基础上提出了分裂基算法(split-radix)。根据现有公开文献和资料，若序列点数满足 $N=2^L$，分裂基 FFT 算法被认为是目前运算量最优的 FFT 算法，这是因为该算法所包含的复数乘法次数最少，并可实现原位运算。

5.6.1　分裂基算法原理

分裂基算法又称为基-2 基-4 混合基算法，它既与基-2 算法相关，也与基-4 算法相关。无论是按时间抽取还是按频率抽取，在基-2 FFT 快速算法的计算过程中(见图 5-2 和图 5-4)，每一级运算中的奇序号(DIT 观察输入序号，DIF 观察输出序号)均乘以一个旋转因子，偶序号则不用乘旋转因子。由于基-4 FFT 比基-2 FFT 更高效，故分裂基算法的基本方法是对偶序号采用基－2 FFT 算法，对奇序号采用基-4 FFT 算法，从而进一步降低复数乘法的运算量。

与 DIT 和 DIF 基-2 FFT 算法类似，分裂基 FFT 算法也要求 $N=2^L$ (L 为正整数)。

DFT 正变换公式如下：

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

根据分裂基算法思想，一个 N 点的 DFT 应分解为一个 $N/2$ 点的 DFT 和两个 $N/4$ 点的 DFT。这种分解既有基-2 的部分，又有基-4 的部分，故称为分裂基分解。因此，时域序列 $x(n)$按序号 n 分解为如下三个子序列：

$$x_1(r)=x(2r),\quad 0\leqslant r\leqslant \frac{N}{2}-1$$

$$\begin{cases}x_2(l)=x(4r+1)\\ x_3(l)=x(4l+3)\end{cases},\quad 0\leqslant r\leqslant \frac{N}{4}-1$$

序列分解以后，DFT 公式可以改变为如下形式：

$$\begin{aligned}X(k)&=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2rk}+\sum_{r=0}^{\frac{N}{4}-1}x(4r+1)W_N^{(4r+1)k}+\sum_{r=0}^{\frac{N}{4}-1}x(4r+3)W_N^{(4r+3)k}\\&=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{4}-1}x_2(r)W_{N/4}^{rk}+W_N^{3k}\sum_{r=0}^{\frac{N}{4}-1}x_3(l)W_{N/4}^{rk}\\&=X_1(k)+W_N^kX_2(k)+W_N^{3k}X_3(k)\end{aligned}\tag{5-46}$$

式中

$$X_1(k)=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{rk}=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_{N/2}^{rk}$$

$$X_2(k)=\sum_{r=0}^{\frac{N}{4}-1}x_2(r)W_{N/4}^{rk}=\sum_{r=0}^{\frac{N}{4}-1}x(4r+1)W_{N/4}^{rk}$$

$$X_3(k)=\sum_{r=0}^{\frac{N}{4}-1}x_3(r)W_{N/4}^{rk}=\sum_{r=0}^{\frac{N}{4}-1}x(4r+3)W_{N/4}^{rk}$$

$X(k)$为 N 点 DFT，而 $X_1(k)$为 $x(n)$中由偶序号组成的 $N/2$ 点 DFT；$X_2(k)$、$X_3(k)$均为 $x(n)$中由奇序号组成的 $N/4$ 点 DFT，根据 W_N^{nk}、$X_1(k)$、$X_2(k)$和 $X_3(k)$所隐含的周期特性可得

$$X_1(k)=X_1\left(k+\frac{N}{2}\right) \tag{5-47}$$

$$X_2(k)=X_2\left(k+\frac{N}{4}\right) \tag{5-48}$$

$$X_3(k)=X_3\left(k+\frac{3N}{4}\right) \tag{5-49}$$

将 $X(k)$分为如下四个区间：

$$\begin{cases}X(k)=X_1(k)+W_N^kX_2(k)+W_N^{3k}X_3(k)\\ X\left(k+\dfrac{N}{4}\right)=X_1\left(k+\dfrac{N}{4}\right)-\mathrm{j}W_N^kX_2(k)+\mathrm{j}W_N^{3k}X_3(k)\\ X\left(k+\dfrac{N}{2}\right)=X_1(k)-W_N^kX_2(k)-W_N^{3k}X_3(k)\\ X\left(k+\dfrac{3}{4}N\right)=X_1\left(k+\dfrac{N}{4}\right)+\mathrm{j}W_N^kX_2(k)-\mathrm{j}W_N^{3k}X_3(k)\end{cases} \tag{5-50}$$

$$0\leqslant k\leqslant\frac{N}{4}-1$$

式(5-50)的运算关系可表示为如图 5-23 所示的分裂基蝶形运算图。

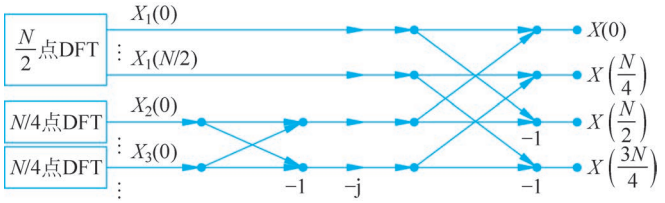


图 5-23 DIT 分裂基 FFT 算法的第一级运算流图

由此可得分裂基 FFT 快速算法的基本蝶形运算单位可表示为如图 5-24 所示的形式。

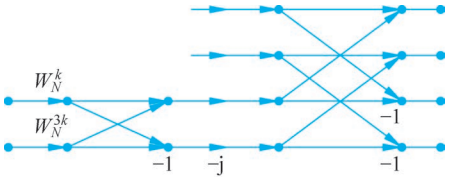


图 5-24 分裂基 FFT 算法的基本蝶形运算

与基-2 算法类似，以 $N=16$ 为例，基于式(5-50)可得出 DIT 分裂基 FFT 算法第一次分解的蝶形运算过程，如图 5-25 所示。

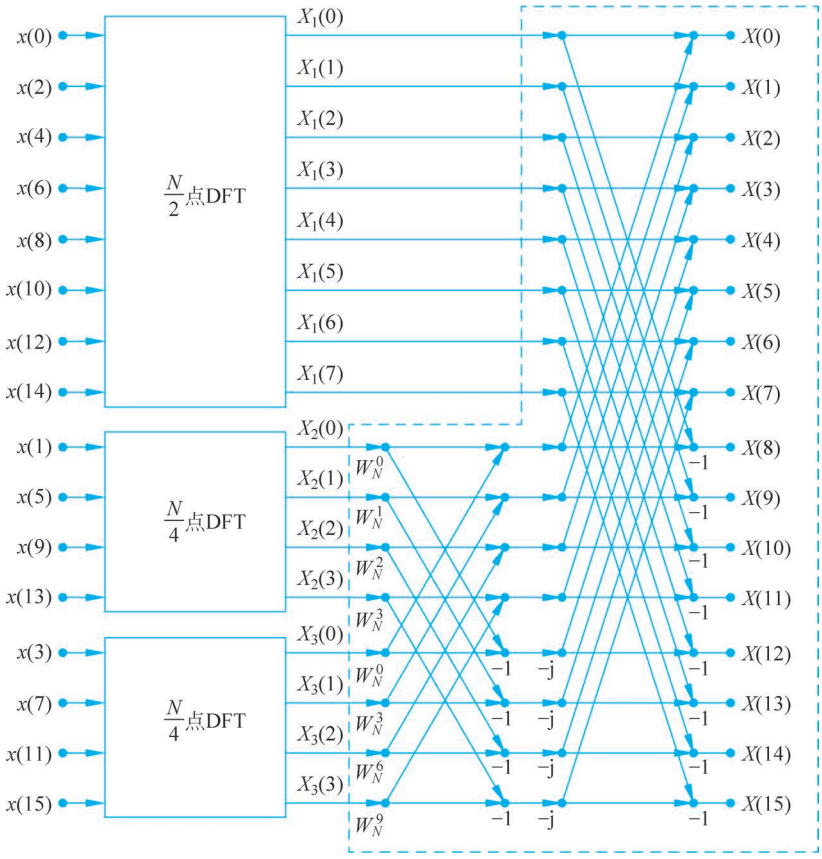


图 5-25　DIT 分裂基($N=16$)FFT 算法第一级分解过程

从图 5-25 可以看出，当 $N=16$ 时，第一级分解得到了图中虚线框内的 4 个分裂基、虚线框左方的 8 点 DFT(左上)和两个 4 点 DFT(左下)。

在第一级分解的基础上，根据式(5-50)继续进行下一级分解，即对虚线框左上的 8 点 DFT 以及左下的两个 4 点 DFT 继续进行分解，直至分解到分裂基的最小单元。对虚线框左上方的 8 点 DFT，根据式(5-50)或图 5-23 可知，对 $X_1(k)$ 进行第二级分解包含 2 个分裂基以及一个 4 点 DFT 和两个 2 点 DFT，而 $X_2(k)$ 和 $X_3(k)$ 均为 4 点 DFT，每一个 4 点 DFT 正好包含一个分裂基单元和一个 2 点 DFT。

由此可得出当 $N=16$ 时，完整分裂基运算流程图共包含 9 个分裂基，如图 5-26 所示。由于图 5-26 分裂基 FFT 算法是根据时域序号 n 的奇偶对序列进行分解的，因此，其输入和输出顺序与 DIT 基-2 FFT 算法一样，输入为正常顺序，输出为倒位序。与基-2 快速算法类似，分裂基快速算法也包括按频率抽取的分裂基快速算法。

5.6.2　分裂基算法的运算量

对于基-2 和基-4 算法，无论是基于时间抽取还是基于频率抽取，其快速算法的运算量均与算法的基本蝶形单元数量有关。分裂基算法的运算量也类似，对于 $N=2^L$ 点的分裂基

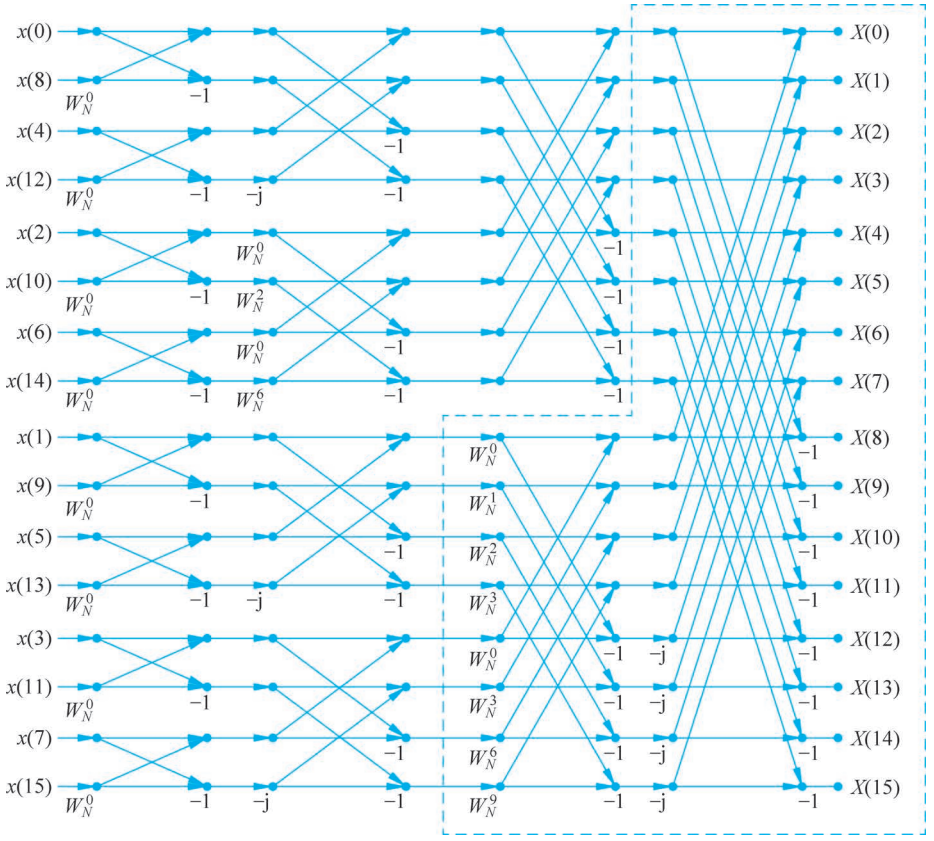


图 5-26 DIT 分裂基(N=16)FFT 算法完整蝶形运算图

算法的运算量,其复数乘法次数与基本分裂基单元的数量密切相关,每一个分裂基蝶形结构包含两次复数乘法运算,而复数加法的运算量与蝶形图节点总数相关。若序列的点数相同,则基-2、基-4 和分裂基算法的节点数相同,故复数加法运算量与基-2 算法相同,不同 FFT 算法的运算量的差别主要体现在复数乘法运算量上,因此,估计分裂基算法总运量实际上就是计算基本分裂基单元的数量。

根据第一次分解后的蝶形运算流图(见图 5-25)可知,当 $N=2^L=16$ 时,第一级分解得到了 $2^{L-2}=2^2$ 个分裂基,同时还有 $2^{L-1}=8$ 点的 DFT 和两个 $2^{L-2}=4$ 点的 DFT。

设 $N=2^L$ 时,分裂基的数量为 s_L,因此,可得如下关于分裂基数量 S_L 的递推方程:

$$s_L=2^{L-2}+s_{L-1}+2s_{L-2}$$

由于 4 点构成一个基本分裂基单元,由此可得完整的递推关系表达式为

$$s_L=2^{L-2}+s_{L-1}+2s_{L-2},\quad L\geqslant 2 \tag{5-51}$$

初始条件为

$$s_0=s_1=0,\quad s_2=1$$

因此,当 $N=2^L$ 时,采用分裂基算法的复数乘法次数为

$$m_{\mathrm{T}}=2s_L$$

即

$$m_{\mathrm{T}}=2^{L-1}+2s_{L-1}+4s_{L-2},\quad L\geqslant 2 \tag{5-52}$$

若初始条件为

$$s_0 = s_1 = 0, \quad s_2 = 1$$

根据式(5-51)及式(5-52)，可得 4 点以上分裂基算法复数乘法运算量如表 5-5 所示，由于支路系数为 W_N^0、$W_N^{N/2}$ 时并不需要进行乘法运算，因此，实际上乘法运算量比表 5-5 略少。

表 5-5　分裂基 FFT 算法复数乘法次数

N	4	8	16	32	64	128	256	512	1024
L	2	3	4	5	6	7	8	9	10
S_L	1	3	9	23	57	135	313	711	1593
m_F	2	4	18	46	114	270	626	1422	3186

表 5-5 可以查询 1024 点以下的分裂基算法的复数乘法运算量，对于 $N>1024$ 点的序列 $x(n)$，可以由式(5-52)递推求得复数乘法次数。从表 5-5 可以发现，分裂基算法的复数乘法次数比 $\frac{1}{3}N\log_2 N$ 更少。以 $N=128$ 为例，$\frac{1}{3}N\log_2 N=299$，而 $m_T=270<299$；基-2 快速算法的乘法次数为 $\frac{N}{2}N\log_2 N=448$，基-4 算法的复数乘法次数为 $\frac{3}{8}N\log_2 N=336$，均比分裂基算法的运算量大一些。

5.7　线性调频 z 变换

FFT 算法可以快速计算出序列 $x(n)$的离散傅里叶变换，即有限长序列 $x(n)$的 z 变换 $X(z)$在 z 平面单位圆上 N 个等间隔抽样值。尽管 z 变换和 DFT 应用非常广泛，对推动离散频域分析起了非常重要的作用，但 z 变换在面对如下问题时仍存在明显的不足。

(1) 工程应用中有时可能仅对信号的某一频段感兴趣，即只需要计算单位圆上某一局部频段的频谱值，例如，为提高窄带信号的频率分辨率，应在窄带范围内进行高密度抽样，而在窄带范围之外可以不抽样。由于 DFT 是基于单位圆上的等间隔抽样，故为了满足窄带部分需求应大幅增加频域的抽样点数，但这样做不仅增大了计算量；进而影响实时性，而且浪费资源。

(2) 对非单位圆上进行抽样，例如语音信号处理需要了解 z 变换极点处的复频率，而对于稳定系统，极点位置可能与单位圆有一定的距离，因此，仅在单位圆上进行抽样难以了解极点处复频率，因而客观上需要在单位圆内部的某一曲线上进行抽样。

线性调频 z 变换(Chirp-z 变换，CZT)的提出正是为了满足上述需求，它是一种可沿 z 平面任意螺线进行抽样的频谱分析方法，并可利用 FFT 原理进行快速计算。

5.7.1　基本原理

已知序列 $x(n)$是点数为 N 的有限长序列，则 $x(n)$的 z 变换为

$$X(n)=\sum_{n=0}^{N-1}x(n)z^{-n}$$

为满足 z 变量在 z 平面的非单位圆路径上进行抽样，可以选择 z 平面的一条螺旋线进行等间隔(角度)的抽样。设抽样点 z_k 为

$$z_k = AW^{-k}, \quad k=0,1,\cdots,M-1 \tag{5-53}$$

M 为所分析的复频域的抽样点数，M 可以大于或等于 N，A 和 W 都是任意复数，为满

足抽样路径的灵活多样性，令

$$\begin{cases} A = A_0 e^{j\theta_0} \\ W = W_0 e^{-j\phi_0} \end{cases} \tag{5-54}$$

式中，A_0 和 W_0 为正实数，将式(5-54)代入式(5-53)得

$$z_k = A_0 e^{j\theta_0} W_0^{-k} e^{jk\phi_0} = A_0 W_0^{-k} e^{j(\theta_0 + k\phi_0)} \tag{5-55}$$

把 $k=0,1,2,\cdots,M-1$ 代入式(5-55)，可得

$$\begin{aligned} z_0 &= A_0 e^{j\theta_0} \\ z_1 &= A_0 W_0^{-1} e^{j(\theta_0+\phi_0)} \\ &\vdots \\ z_k &= A_0 W_0^{-k} e^{j(\theta_0+k\phi_0)} \\ &\vdots \\ z_{M-1} &= A_0 W_0^{-(M-1)} e^{j[\theta_0+(M-1)\phi_0]} \end{aligned}$$

抽样值 z_k 在如图 5-27 所示的 z 平面的螺旋线上进行。

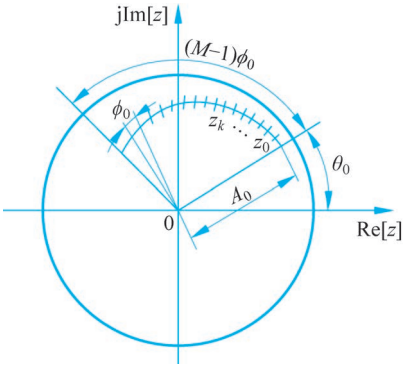


图 5-27 CZT 在 z 平面螺旋线上的抽样点

根据上述分析及式(5-53)可知：

(1) A_0 表示螺旋线的第一个抽样点(起始抽样点)到坐标原点的距离，即抽样点 z_0 的向量半径长度，由于稳定系统的极点在单位圆内，因此 $A_0 \leqslant 1$。

(2) θ_0 表示第一个抽样点 z_0 的相位角，可以为任意正负值。

(3) ϕ_0 表示两相邻抽样点之间的相角差，若 $\phi_0 > 0$ 时，表示抽样沿逆时针方向进行；$\phi_0 < 0$，表示抽样沿顺时针方向进行。

(4) W_0 表示螺旋线的伸展率，$W_0 > 1$ 时，螺旋线随着 k 的增加向圆心收缩；$W_0 < 1$ 时，螺旋线随 k 的增加远离圆心向外伸展；$W_0 = 1$ 表示以 A_0 为半径的一段圆弧，若 $A_0 = 1$ 则该圆弧是单位圆的一段。

若 $\phi_0 = \dfrac{2\pi}{N}$，$M=N$，$A = A_0 e^{j\theta_0} = 1$，$W_0 = 1$，则可得出 $W = e^{-j\frac{2\pi}{N}}$，螺旋线演变为单位圆，式(5-53)演变为 N 点 DFT 运算，即 N 个抽样点等间隔均匀分布在单位圆上。这时若取 ϕ_0 为非零任意值，则 DFT 的结果是单位圆上某一段圆弧的频谱。

将式(5-53)代入 z 变换表达式，可得

$$X(z_k) = \sum_{n=0}^{N-1} x(n) z_{z_k}^{-n} = \sum_{n=0}^{N-1} x(n) A^{-n} W^{nk}, \quad 0 \leqslant k \leqslant M-1 \tag{5-56}$$

5.7.2 CZT 快速算法

1. 快速算法原理

若对式(5-56)直接进行计算，则计算过程和方法均与直接计算 DFT 类似，可获得频域

M 个抽样点的离散频谱。

式(5-56)的运算量也可以准确估计，共包含 NM 次复数乘法和 $(N-1)M$ 次复数加法。显然，若 N 与 M 大致相当，则乘法次数(NM)与 DFT 的 N^2 为同阶量级，若采样点数很大，即 N 与 M 均较大，则运算量很大，将影响到 CZT 的运算速度，因此，应考虑采用快速运算。

布鲁斯坦(Bluestein)提出了实现 CZT 快速算法的基本思路，可以将该运算转换为卷积运算形式，从而可以根据 FFT 算法原理实现 CZT 快速算法，提高运算速度。

布鲁斯坦快速算法原理利用了如下代数恒等式：

$$nk=\frac{1}{2}[n^2+k^2-(k-n)^2]$$

将该恒等式代入式(5-56)，可得

$$X(z_k)=\sum_{n=0}^{N-1}x(n)A^{-n}W^{\frac{n^2}{2}}W^{-\frac{(k-n)^2}{2}}W^{\frac{k^2}{2}}=W^{\frac{k^2}{2}}\sum_{n=0}^{N-1}[x(n)A^{-n}W^{\frac{n^2}{2}}]W^{-\frac{(k-n)^2}{2}} \tag{5-57}$$

为利用快速卷积对式(5-57)进行计算，设

$$g(n)=x(n)A^{-n}W^{\frac{n^2}{2}},\quad n=0,1,\cdots,N-1 \tag{5-58}$$

$$h(n)=W^{-\frac{n^2}{2}} \tag{5-59}$$

则式(5-57)可得到以下类似卷积形式的表达式：

$$X(z_k)=W^{\frac{k^2}{2}}\sum_{n=0}^{N-1}g(n)h(k-n),\quad k=0,1,\cdots,M-1 \tag{5-60}$$

该式表明，抽样点 z_k 处的 z 变换可通过求 $g(n)$ 与 $h(n)$ 的卷积结果再乘以系数 $W^{\frac{k^2}{2}}$ 实现。因此，式(5-60)可表示为

$$X(z_k)=W^{\frac{k^2}{2}}[g(k)*h(k)],\quad k=0,1,\cdots,M-1 \tag{5-61}$$

由此可得，计算 $X(z_k)$ 的流程如图 5-28 所示。

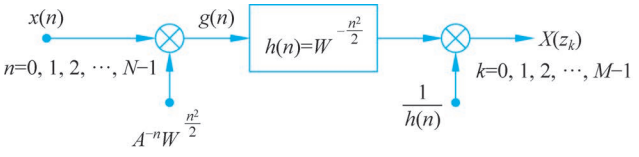


图 5-28　Chirp-z 变换运算流程

序列 $h(n)=W^{-\frac{n^2}{2}}$ 可以理解为频率随离散时间(n)呈线性增长的复指数序列，该信号形式是雷达系统中的一种重要信号类型，称为线性调频信号(Chirp signal)，因此，该变换**称为线性调频 z 变换**。

2. 快速算法实现步骤

根据式(5-61)计算卷积，先分析序列 $g(n)$ 和 $h(n)$ 的长度和区间范围，由于 $x(n)$ 是 N 点序列，$n=0,1,2,\cdots,N-1$，因此 $g(n)$ 也是 N 点序列。而 $h(n)=W^{-\frac{n^2}{2}}$ 为无限长偶对称序列，但由于 z 平面上的抽样点只有 M 个，根据式(5-60)，若将 $h(n)$ 视为单位抽样响应，则

$$k=0,1,2,\cdots,M-1$$

$h(n)$是非因果线性系统，$h(n)$在$n=-(N-1)\sim(M-1)$范围内取值，如图 5-29 所示，$h(n)$为 $L=N+M-1$ 点有限长序列。

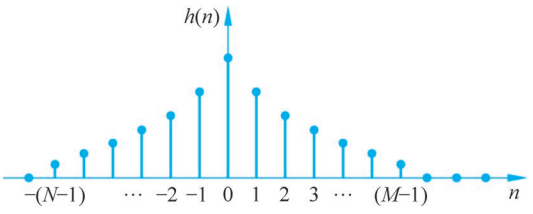


图 5-29　序列 $h(n)$的区间范围

根据序列自变量的取值范围，$g(n)*h(n)$卷积序列的长度为 $2N+M-2$，根据圆周卷积代替线性卷积的条件，圆周卷积长度应不小于 $2N+M-2$。

由于仅需计算 $X(z_k)$的前 M 个值，因此，为了降低运算量，可将圆周卷积的点数 L 取为

$$L \geqslant N+M-1$$

L 应为 2 的整数幂，取满足条件的最小 L 值即可。根据运算要求，对序列 $h(n)$应补零值点，使序列长度等于 L，即从 $n=M$ 开始补 $L-(N+M-1)$个零点，当 $n=L-N$ 时，对序列 $h(n)$以 L 为周期进行周期延拓，对 $g(n)$同样在末端补零至 L 点。$h(n)$ 和 $g(n)$均为 L 点序列之后，就可以进行 CZT 快速运算了，一般包括如下五个步骤。

(1) 计算 FFT[$g(n)$]。

根据线性调频计算原理、圆周卷积代替线性卷积的条件以及 FFT 计算对点数的要求，选择满足 $L\geqslant N+M-1$ 和 L 取 2 的整数幂的最小整数，然后对序列 $g(n)=x(n)A^{-n}W^{\frac{n^2}{2}}$ 的末端按如下方式补零值点，使序列 $g(n)$成为 L 点有限长序列：

$$g(n)=\begin{cases} A^{-n}W^{\frac{n^2}{2}}x(n), & 0\leqslant n\leqslant N-1 \\ 0, & N\leqslant n\leqslant L-1 \end{cases} \tag{5-62}$$

并计算序列 $g(n)$的 L 点 FFT 运算，得 L 点 $G(k)$，即

$$G(k)=\text{FFT}[g(n)]$$

(2) 计算 FFT[$h(n)$]。

对序列 $h(n)$补零点成为 L 点有限长序列，在 $n=0\sim(M-1)$范围内取 $h(n)=W^{-\frac{n^2}{2}}$，在 $n=M\sim(L-N)$范围内 $h(n)$可取任意值，或者直接取零值，在 $n=(L-N+1)\sim(L-1)$范围内，$h(n)=W^{-\frac{n^2}{2}}$并进行周期延拓，则延拓序列为 $W^{-\frac{m^2}{2}}$，即

$$h(n)=\begin{cases} W^{-\frac{n^2}{2}}, & 0\leqslant n\leqslant M-1 \\ 0(\text{或任意值}), & M\leqslant n\leqslant L-N \\ W^{-\frac{(L-n)^2}{2}}, & L-N+1\leqslant n\leqslant L-1 \end{cases} \tag{5-63}$$

根据序列 $h(n)$的表达式可知，$h(n)$是序列 $W^{-\frac{m^2}{2}}$ 以 L 为周期进行周期延拓的主值序列，如图 5-30 所示，对序列 $h(n)$进行 L 点 FFT 运算，即

$$H(k)=\mathrm{FFT}[h(n)]$$

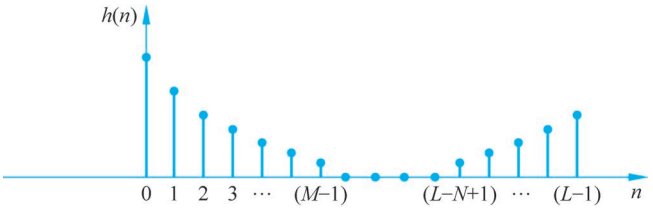


图 5-30　序列 $h(n)$ 延拓后的主值序列

(3) 计算频域乘积。

计算频域序列 $H(k)$ 和 $G(k)$ 的乘积，即计算 $Y(k)=H(k)G(k)$，得到 L 点频域序列 $Y(k)$。

(4) 计算圆周卷积。

计算 $Y(k)$ 的 L 点傅里叶逆变换，即对 $Y(k)$ 进行 IFFT 计算，得到 $h(n)$ 与 $g(n)$ 的圆周卷积序列为

$$y(n)=h(n)\otimes g(n)=\frac{1}{L}\sum_{k=0}^{L-1}H(k)G(k)\mathrm{e}^{\mathrm{j}\frac{2\pi}{L}kn}$$

圆周卷积序列 $y(n)$ 的前 M 个值等于原 $h(n)$ 与 $g(n)$ 的线性卷积，当 $n\geqslant M$ 时，序列 $y(n)$ 的值无意义，可以不予计算。

(5) 计算 $X(z_k)$。

$$X(z_k)=W^{\frac{k^2}{2}}y(k),\quad 0\leqslant k\leqslant M-1 \tag{5-64}$$

3. MATLAB 实现 CZT 快速算法

MATLAB 提供了实现 CZT 快速算法函数 czt，调用格式如下：

```
y = czt(x,M,W,A)
```

该函数的功能是计算长度为 M 点的序列 $x(n)$ 的 Chirp-z 变换 $X(z_k)$，A 为频率抽样的起始点位置，W 为采样轮廓线上各抽样点之间的比率，y 用于存储 CZT 的变换值 $X(z_k)$。

5.7.3　CZT 算法运算量

当 N 和 M 很大时，采用 CZT 快速算法计算 $X(z_k)$ 比直接求 $X(z_k)$ 能有效节省运算量，CZT 快速算法所需的运算量主要包括如下几项。

(1) 求 $g(n)$ 的系数。补零点时获得 L 点序列 $g(n)=(A^{-n}W^{\frac{n^2}{2}})x(n)$，由于仅有 N 点非零值，复数乘法次数为 N 次，而 $g(n)$ 的系数 $A^{-n}W^{\frac{n^2}{2}}$ 可用如下递推法求解：

$$g(n)=(A^{-n}W^{\frac{n^2}{2}})x(n)=C_n x(n)$$

式中

$$C_n=A^{-n}W^{\frac{n^2}{2}}=C_{n-1}D_{n-1}$$

$$D_n=W^n W^{\frac{1}{2}}A^{-1}=W^n D_0=WD_{n-1}$$

初始条件为

$$C_0=1,\quad D_0=W^{\frac{1}{2}}A^{-1}=\frac{W_0^{1/2}}{A_0}\mathrm{e}^{-\mathrm{j}\left(\frac{\phi_0}{2}+\theta_0\right)}$$

因此,根据式(5-85)可求出 N 个系数 C_n,共包括 $2N$ 次复数乘法。

(2) 求 $h(n)$的系数。补零点得到 L 点序列 $h(n)$,而 $h(n)$由 $W^{-\frac{n^2}{2}}$ 在 $-(N-1)\leqslant n\leqslant M-1$ 段内的序列值构成,由于 $W^{-\frac{n^2}{2}}$ 具有偶对称性,故若 $N>M$,则只需求出 $0\leqslant n\leqslant N-1$ 范围内的 N 点序列值即可。和步骤(1)类似,$W^{-\frac{n^2}{2}}$ 的值同样采用递推法求解,共需 $2N$ 次复数乘法。

(3) 计算 $G(k)H(k)$。需两次 L 点 FFT 运算,复数乘法运算量为 $2\times\frac{1}{2}L\log_2 L$。

(4) 计算圆周卷积 $y(n)$。需一次 L 点 IFFT 运算,复数乘法运算量为$\frac{1}{2}L\log_2 L$。

(5) 计算 $Y(k)=G(k)H(k)$。需 L 次复数乘法。

(6) 计算 $X(z_k)$。$X(z_k)=W^{\frac{k^2}{2}}y(k)(0\leqslant k\leqslant M-1)$,需 M 次复数乘法。

因此,计算 CZT 全部值所需的复数乘法次数为

$$m_{\mathrm{T}}=\frac{3}{2}L\log_2 L+5N+L+M \tag{5-65}$$

而直接计算 $X(z_k)$的复数乘法次数为 NM 次,当 N、M 均为较大数时,采用基于 FFT 的 CZT 快速算法比直接计算的运算量要小很多,且 N、M 越大,优势越明显,例如若 $N=M=64$,CZT 快速算法与直接算法的乘法运算量相差无几,若 $N=M=128$,则快速运算开始显示出优势。

CZT 的算法非常灵活,时域输入序列 $x(n)$的点数 N 可以和频域序列的点数 M 不相等,且 N 和 M 均可以是质数或其他任意正整数。抽样间隔 ϕ_0 可以根据需要任意选取,即频率分辨率可以根据需要进行调整。计算 z 变换的螺旋线也可以根据需要进行选择。第一个采样点的选取也无特殊限制,可以根据应用需要自由选择,即可以从任意复频率开始对输入数据进行分析,可以方便地对窄带高频信号进行高密度抽样分析。相比 DFT 而言,CZT 是进行频谱细化分析的一种方法,对于某些特定应用,FFT 不能精确反映信号的局部频谱特性,但采用 CZT 算法可以获得更精确的频谱特性。

5.8 线性卷积与线性相关的 FFT 算法

有限长序列傅里叶变换(DFT)的快速算法是很多其他快速算法的基础,库利-图基发表的傅里叶变换快速算法的论文,极大地推动了傅里叶变换走向工程应用。实际上,FFT 不仅是一种快速算法,而且是一种思考问题的方法,它不限于 DFT 快速运算的实现,将 FFT 的思维方法应用于其他算法的快速实现也是 FFT 思想的核心。

5.8.1 线性卷积的 FFT 算法

线性卷积运算是信号处理的基本运算,应用非常广泛。例如,对于 FIR 滤波器,其输出

等于滤波器的单位冲激响应 $h(n)$ 与输入信号 $x(n)$ 的线性卷积。

设输入序列 $x(n)$ 为 N 点，单位冲激响应序列 $h(n)$ 为 M 点，则输出 $y(n)$ 为

$$y(n)=x(n)*h(n)=\sum_{m=0}^{M-1}h(m)x(n-m)$$

$y(n)$ 为 $L=N+M-1$ 点有限长序列。

1. 线性卷积的运算量

根据线性卷积的计算公式，序列 $x(n)$ 的每一个值都必须和序列 $h(n)$ 的全部值进行乘法运算，因此，线性卷积的乘法的总运算量为 NM 次，以 M_{Con} 表示乘法次数，则有

$$M_{\text{Con}}=NM \tag{5-66}$$

对于线性相位 FIR 滤波器而言，由于满足

$$h(n)=\pm h(M-1-n)$$

所以线性相位数字滤波器的网络结构通常是 $h(n)$ 与 $h(M-1-n)$ 共用放大器，因此，乘法运算次数可以减少一半，即

$$M_{\text{Con}}=\frac{NM}{2} \tag{5-67}$$

由上可知，线性卷积算法复数乘法的运算量为 $M_{\text{Con}}=NM$，虽然在线性相位数字滤波器中运算量为 $M_{\text{Con}}=\dfrac{NM}{2}$，当两序列长度接近时，如 N 与 M 接近或者 $N=M$ 时，则运算量近似为 N^2。由于计算 N 点序列 $x(n)$ 的 DFT 运算量为 N^2，即线性卷积和 DFT 运算量相等，都非常大，因此，对于线性卷积，采用快速算法具有很好的效果。

2. 线性卷积的快速算法

线性卷积快速算法的基本思路是以 FFT 算法为基础进行快速运算，即用圆周卷积来代替该线性卷积，然后用 FFT 算法进行计算，如图 5-31 所示。

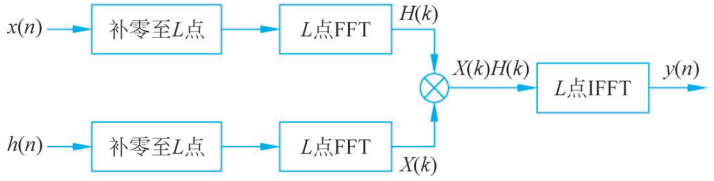


图 5-31　线性卷积快速算法结构

当序列 $x(n)$ 和 $h(n)$ 末端补零值点至 $L\geqslant N+M-1$ 时，则圆周卷积可以代替线性卷积，因此，需要在序列 $x(n)$ 和 $h(n)$ 末端补零点，使 $L\geqslant N+M-1$ 并取 L 为 2 的整数幂，此时可以采用 FFT 算法，即

$$x(n)=\begin{cases}x(n), & 0\leqslant n\leqslant N-1\\ 0, & N\leqslant n\leqslant L-1\end{cases}$$

$$h(n)=\begin{cases}h(n), & 0\leqslant n\leqslant M-1\\ 0, & M\leqslant n\leqslant L-1\end{cases}$$

补零后计算圆周卷积为

$$y(n)=x(n)\otimes h(n)$$

由于 $L\geqslant N+M-1$ 时，圆周卷积与线性卷积结果一致。因此，可得线性卷积快速运算

的基本步骤如下。

(1) 计算 $H(k)=\mathrm{FFT}[h(n)]$。

(2) 计算 $X(k)=\mathrm{FFT}[x(n)]$。

(3) 计算 $Y(k)=X(k)H(k)$。

(4) 计算 $y(n)=\mathrm{IFFT}[Y(k)]$。

步骤(1)、(2)、(4)均采用 FFT 进行计算,步骤(3)为乘法运算,因此,线性卷积快速算法的乘法运算量包括三次 FFT 运算和 N 次乘法运算,总运算量如下:

$$m_{\mathrm{T}}=\frac{3}{2}L\log_2 L+L=L\left(1+\frac{3}{2}\log_2 L\right) \tag{5-68}$$

虽然运算量包含 3 个 L 点的 FFT 运算,但当 N 和 M 较大时(即 L 比较大),依然具有很好的快速运算效果。现在以线性相位 FIR 数字滤波器为例,分析直接计算线性卷积和用 FFT 计算线性卷积这两种方法的乘法次数。

令 k 为 m_{Con} 与 m_{T} 的比值,则

$$k=\frac{m_{\mathrm{Con}}}{m_{\mathrm{T}}}=\frac{NM}{2L\left(1+\frac{3}{2}\log_2 L\right)} \tag{5-69}$$

若 $k>1$ 时,说明采用线性卷积快速算法能节省运算量。$L=N+M-1$,工程应用中,一般 $x(n)$和 $h(n)$的点数较大,即 N 和 M 较大,因而 $L\approx N+M$,根据代数知识,M 和 N 越接近时,乘积 NM 越大,分两种情况讨论。

1) $x(n)$与 $h(n)$点数接近

若 $M=N$,对提升比值 k 有利,这时 $L=2N-1\approx 2N$,则

$$k=\frac{NM}{2L\left(1+\frac{3}{2}\log_2 L\right)}\approx\frac{N^2}{4L\left(1+\frac{3}{2}\log_2(2N)\right)}=\frac{N}{4(2.5+1.5\log_2 N)}=\frac{N}{10+6\log_2 N}$$

根据序列长度不同,运算量比值如表 5-6 所示

表 5-6 线性卷积直接计算与快速算法运算量比值($N=M$)

$N=M$	8	32	64	128	256	512	1024	2048	4096
k	0.286	0.80	1.39	2.46	4.41	8	14.62	26.95	49.95

根据表 5-6,若 $N=M=8$、32 时,快速算法的运算量反而大于直接卷积,此时不宜采用快速算法。若 $N=M=64$,快速算法比直接卷积效果略好,因此 $N=M=64$ 是节省运算量的临界值。若 $N=512$,快速算法的运算量仅为直接卷积的 1/8;若 $N=4096$,快速算法的运算量仅有直接卷积的 2%。

【例 5-2】 FFT 计算快速卷积示例。

已知序列 $x(n)$和 $h(n)$如下:

$$x(n)=0.6^n,\quad 0\leqslant n\leqslant 15,h(n)=R_{15}(n)$$

试计算 $y(n)=x(n)*h(n)$。

解: 用 FFT 计算线性卷积,应注意圆周卷积代替线性卷积的条件为

$$L\geqslant N_1+N_2-1$$

$$N_1=\mathrm{length}[x(n)],\quad N_2=\mathrm{length}[h(n)]$$

计算线性卷积的 MATLAB 程序如下：

```
clc;clear all;close all
n = [0:1:15];xn = 0.6.^n;n1 = length(n);
m = [0:1:15];n2 = length(m);hn = ones(1,n2);
L = n1 + n2 - 1;
Xk = fft(xn,L);Hk = fft(hn,L);Yk = Xk. * Hk;
yn = ifft(Yk,L)
figure(1);stem(abs(yn))
xlabel('n');ylabel('y(n) = x(n) * h(n)');
title('FFT 实现快速卷积运算');
axis([0,32,0,1.1 * max(yn)]);
```

程序计算结果如下：

```
yn  = Columns 1 through 12
    1.0000    1.6000    1.9600    2.1760    2.3056    2.3834    2.4300    2.4580
    2.4748    2.4849    2.4909    2.4946
  Columns 13 through 24
    2.4967    2.4980    2.4988    2.4993    1.4993    0.8993    0.5393    0.3233
    0.1937    0.1159    0.0693    0.0413
  Columns 25 through 31
    0.0245    0.0144    0.0084    0.0047    0.0026    0.0013    0.0005
```

FFT 快速卷积结果的波形如图 5-32 所示。

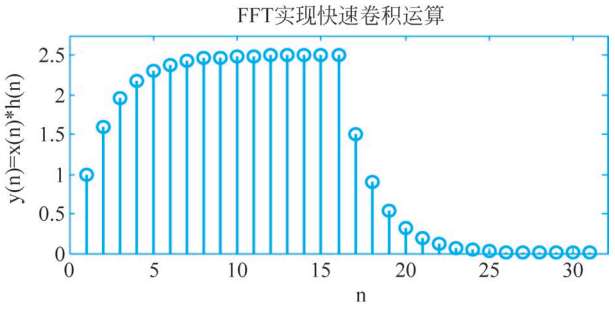


图 5-32　FFT 计算线性卷积的结果

2）$x(n)$与 $h(n)$点数相差很大

若序列 $x(n)$的点数很多，即

$$N \gg M$$

则有

$$L = N + M - 1 \approx N$$

可得

$$k = \frac{NM}{2L\left(1 + \frac{3}{2}\log_2 L\right)} \approx \frac{NM}{2N\left(1 + \frac{3}{2}\log_2 N\right)} = \frac{M}{2 + 3\log_2 N} \tag{5-70}$$

当 L 与 M 相差很大时，k 值将快速下降，线性卷积快速算法的优越性就难以体现，这时，可以采用分段卷积方法进行卷积快速运算。

应用中，有时会遇到一个短序列与一个长序列的卷积计算问题，显然，这时不宜直接应用上述卷积快速算法，否则浪费资源。可以考虑将长序列分解为与短序列点数相当或相等的若干段，分别计算每一段的卷积结果，然后用合适的方法将各段的计算结果合并，得到最

终计算结果。由于分段后点数相当,故对每一段的卷积运算均可采用快速算法。对 $x(n)$分段进行卷积运算的方法包括重叠相加法和重叠保留法两种。

(1) 重叠相加法。

重叠相加法(overlap-add)是指在计算分段卷积时需要对各输出段的重叠部分进行相加,因而称为**重叠相加法**。

设 $x(n)$为长序列,$h(n)$为短序列,点数为 M。将长序列 $x(n)$分解为互不重叠的若干段,每段为 N 点,N 与 M 相等或相差不大,令 $x_i(n)$表示 $x(n)$的第 i 段:

$$x_i(n)=\begin{cases}x(n), & iN\leqslant n\leqslant (i+1)N-1\\ 0, & \text{其他}\end{cases},\quad i=0,1,\cdots \tag{5-71}$$

则输入序列 $x(n)$可表示为

$$x(n)=\sum_i x_i(n) \tag{5-72}$$

$$y_i(n)=\sum_i x_i(n)*h(n)$$

因此,$x(n)$与 $h(n)$的线性卷积等于各 $x_i(n)$与 $h(n)$的线性卷积之和,即

$$y(n)=x(n)*h(n)=\sum_i x_i(n)*h(n) \tag{5-73}$$

求和的每一卷积项都可采用线性卷积的快速算法进行计算。由于每一段的卷积结果 $y_i(n)=\sum_i x_i(n)*h(n)$ 的长度为 $L\geqslant N+M-1$,L 向上取值并符合 FFT 算法要求。

应用快速算法计算每一段的卷积前,应先对 $x_i(n)$及 $h(n)$末端补零点,补到 L 点,以满足圆周卷积代替线性卷积 $y_i(n)$的条件,方便进行快速卷积运算:

$$y_i(n)=x_i(n)\otimes h(n)$$

因为 $x_i(n)$为 L 点,而 $y_i(n)$为 $N=L+M-1$ 点,故相邻两段卷积计算的结果将有$(M-1)$点产生重叠,即上一段卷积结果的后$(M-1)$点和下一段卷积运算结果的前$(M-1)$个点发生重叠。根据式(5-73),应将重叠部分相加,再与无重叠部分合并为 $y(n)$的输出结果。

因此,采用 FFT 实现重叠相加法的步骤如下:

① 用 FFT 计算 $H(k)=\mathrm{DFT}[h(n)]$;

② 用 FFT 计算 $X_i(k)=\mathrm{DFT}[x_i(n)]$;

③ 计算乘积运算 $Y_i(k)=X_i(k)H(k)$;

④ 用 IFFT 计算 $y_i(n)=\mathrm{IDFT}[Y_i(k)]$;

⑤ 将各段卷积结果 $y_i(n)$相加,注意重叠区间的长度,$y(n)=\sum_i y_i(n)$。

【例 5-3】 重叠相加法计算线性卷积示例。

已知序列 $x(n)$和 $h(n)$如下:

$$x(n)=3n-1,\quad 0\leqslant n\leqslant 15,\quad h(n)=[1,0,-1]$$

试采用重叠相加法计算 $y(n)=x(n)\otimes h(n)$。

解: 对于两序列长度相差很大的线性卷积计算,可以采用重叠相加法。

MATLAB 提供了基于 FFT 的重叠相加法计算线性卷积的函数 fftfilt.m,该函数计算圆周卷积时采用快速傅里叶变换 FFT。函数调用格式如下:

```
y = fftfilt(h,x,L)
```

x 和 h 分别表示序列 $x(n)$和 $h(n)$，L 表示分段长度，该参数可用默认值，调用格式如下：

```
y = fftfilt(h,x)
```

分段长度 L 为默认值时，系统自动对长序列选择一个合适的分段长度。

计算线性卷积的 MATLAB 程序如下：

```
clc;clear all;close all
n = 0:15;
x = 3 * n - 1;h = [1,0, - 1];
y = fftfilt(h,x)
```

本题 $x(n)$和 $h(n)$的序列长度相差较大，为了加深对重叠相加法的理解，程序设置了分段长度为 3 和 4 两种情况，计算结果相同。

程序计算的卷积结果如下：

```
y = Columns 1 through 11
 - 1.0000    2.0000    6.0000    6.0000    6.0000    6.0000    6.0000    6.0000
6.0000    6.0000    6.0000
Columns 12 through 16
6.0000    6.0000    6.0000    6.0000    6.0000
```

(2) 重叠保留法。

重叠保留法(overlap-save)与重叠相加法的相同之处是都需要先对序列 $x(n)$进行分段，分为长度为 N 点的若干短序列。不同之处是，对分段之后的短序列的处理方式不同，重叠保留法的计算步骤如下。

① 分段。重叠保留法对短序列不进行补零操作，而是在 $x_i(n)$的开始部分补上前一段 $x_{i-1}(n)$保留下来的$(M-1)$个输入序列的值，组成长度为 $L=N+M-1$ 点的第 i 段序列 $x_i(n)$。若 $L=N+M-1$ 不满足 2 的整数幂，则可在每段序列末端补零值点以达到要求。

② 首段单独处理。分段的第一段 $x_0(n)$由于没有上一段，应特殊处理，这时可以直接将第一段 $x_0(n)$的前$(M-1)$个值赋为零。

③ 计算各段线性卷积。采用 FFT 计算 $h(n)$和 $x_i(n)$的线性卷积，这时每段圆周卷积计算结果的前$(M-1)$点的值存在误差，不等于线性卷积的值，应舍去，保留后面的 $N-M+1$ 点的值。

④ 修正后连接成卷积序列 $y(n)$。完成每一段卷积的计算并修正到正确值，最后将修正后正确的卷积结果 $y_i(n)$连接成一个输出序列 $y(n)$。

由于该方法在计算时，每一组(每一段)均由前一段保留下来的$(M-1)$个点和新的$(N-M+1)$个点所组成，故称为**重叠保留法**。

【例 5-4】 重叠保留法计算线性卷积示例。

已知序列 $x(n)$和 $h(n)$如下：

$$x(n)=6n-1,\quad 0\leqslant n\leqslant 15,\quad h(n)=[1,0,-1]$$

试采用重叠保留法计算 $y(n)=x(n)*h(n)$。

解： 重叠保留法计算线性卷积的 MATLAB 程序如下：

```
clc;close all;clear all;
n = 0:12;x = 6 * n - 1;h = [1,0, - 1];
L_x = length(x);L_h = length(h);
N = 6;N = 2^(ceil(log10(N)/log10(2)));
M = L_h - 1;L = N - M;
h_fft = fft(h,N);
k = ceil(L_x/L);
x = [zeros(1,M),x,zeros(1,N - 1)];
y = zeros(k,N);
for i = 0:k - 1
   xk = fft(x(i * L + 1:i * L + N));
   y(i + 1,:) = real(ifft(xk. * h_fft));
end
y = y(:,L_h:N)'; % 舍弃前面 M 个值
yn = (y(:))'
```

计算的卷积结果如下：

```
yn = - 1.0000    5.0000   12.0000   12.0000   12.0000   12.0000   12.0000   12.0000
12.0000    12.0000    12.0000    12.0000   12.0000    - 65.0000   - 71.0000
```

5.8.2 线性相关的 FFT 算法

线性相关包括自相关与互相关，它们在信息与通信、统计和信号处理等众多领域得到了广泛应用。互相关函数和自相关函数都是信号分析与处理的重要概念，互相关函数表示两个不同时间序列之间的相关程度，它描述了信号 $x(n)$ 与 $y(n)$ 在任意两个不同时刻取值之间的相关程度，两个不同信号既可以是确定性信号，也可以是随机信号。一般情况下，信号相关的概念通常指两个信号的互相关。

应用 FFT 计算相关，即利用圆周相关来计算线性相关，常称之为快速相关计算。与线性卷积的快速算法类似，也应采用末端补零点的方法避免混叠失真。

设序列 $x(n)$ 长度为 N 点，$y(n)$ 长度为 M 点，线性相关定义如下：

$$r_{xy}(n)=\sum_{m=0}^{M-1}x(n+m)y^{*}(m) \tag{5-74}$$

用圆周相关代替线性相关应满足 $L\geqslant N+M-1$，且 L 为 2 的整数幂，因此，需对序列末端按如下方式补零：

$$x(n)=\begin{cases}x(n), & 0\leqslant n\leqslant N-1\\ 0, & N\leqslant n\leqslant L-1\end{cases}$$

$$y(n)=\begin{cases}y(n), & 0\leqslant n\leqslant M-1\\ 0, & M\leqslant n\leqslant L-1\end{cases}$$

快速相关计算的步骤如下：

(1) 求 L 点 FFT，$X(k)=\mathrm{DFT}[x(n)]$；

(2) 求 L 点 FFT，$Y(k)=\mathrm{DFT}[y(n)]$；

(3) 求乘积，$R_{xy}(k)=X(k)Y^{*}(k)$；

(4) 求 L 点 IFFT，$r_{xy}(n)=\mathrm{IFFT}[R_{xy}(k)]$。

由计算步骤可知，采用 FFT 算法计算线性相关的运算量与采用 FFT 计算线性卷积的运算量是相等的，而且可以应用已有的 FFT 算法程序进行计算。

【例 5-5】　FFT 计算线性相关示例。

已知序列 $x(n)$和 $y(n)$如下：

$$x(n)=[3,5,7,0,-1,3,5],\quad y(n)=R_{15}(n)$$

试计算 $x(n)$与 $y(n)$的线性相关。

解：用 FFT 计算线性相关，应先对序列 $x(n)$和 $y(n)$的末端补零值点，直至长度满足条件：

$$L\geqslant N_1+N_2-1$$

$$N_1=\text{length}[x(n)],\quad N_2=\text{length}[y(n)]$$

计算线性卷积的 MATLAB 程序如下：

```
clc;clear all;close all
xn = [3,5,7,0, - 1,3,5];n1 = length(xn);
noise = randn(1,n1);
yn = [5,7,0, - 1,3,5,3];
n1 = length(xn);n2 = length(yn);
L = n1 + n2 - 1;
Xk = fft(xn,L);Yk = fft(yn,L);
Rk = conj(Xk). * Yk;Rxy = ifft(Rk,L)
figure(1);n = length(Rxy);stem(abs(Rxy))
xlabel('n');ylabel('序列相关');
title('FFT 实现快速相关计算');
axis([0,16,0,1.1 * max(Rxy)]);
```

线性相关的计算结果如下：

```
Rxy = 77.0000   18.0000   13.0000   47.0000   55.0000   30.0000    9.0000   25.0000
50.0000    16.0000   - 12.0000    47.0000    109.0000
```

线性相关的波形如图 5-33 所示。

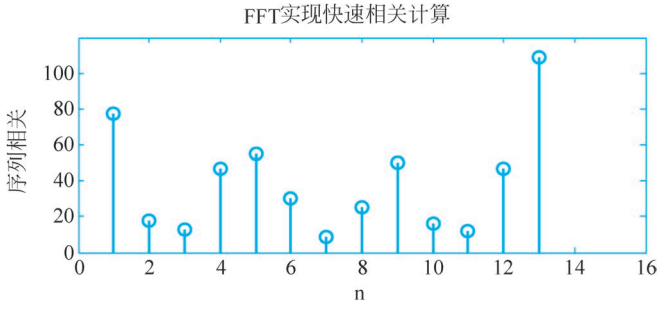


图 5-33　FFT 计算线性相关的结果

习题

1. 已知 $X(k)$和 $Y(k)$是两个 N 点实序列 $x(n)$和 $y(n)$的 DFT，即

$$X(k)=\text{DFT}[x(n)]=\sum_{n=0}^{N-1}x(n)z^{-n}$$

$$Y(k)=\text{DFT}[y(n)]=\sum_{n=0}^{N-1}y(n)z^{-n}$$

为了提高计算效率,试设计用一个 N 点 IDFT/IFFT 计算出 $x(n)$和 $y(n)$的算法。

2. 试根据 DIT 算法原理,绘出当 $N=8$ 时,DIT 基-2 快速算法完整的蝶形运算图,其中输入为自然顺序,输出为倒位序,并标出正确的支路系数。

3. 若计算机运算一次复数乘法的平均时间为 2μs,计算一次复数加法的平均时间为 0.2μs,用该计算机求 1024 点 DFT,试计算完成下列运算所需要的时间:

(1) 使用 DFT 计算需要多少时间?

(2) 使用 FFT 计算需要多少时间?

4. 试根据 DIT 算法原理,绘出当 $N=16$ 时,DIT 基-2 快速算法完整的蝶形运算图,其中输入为倒位序,输出为自然顺序,并标出正确的支路系数。

5. 若有限长序列 $x(n)$的长度 $N=64$,试写出 DIT 基-2 FFT 算法(输入倒位序、输出自然顺序)完整蝶形图中第三级蝶形运算中的全部支路系数。

6. 试根据 DIF 算法原理,绘出当 $N=16$ 时,DIF 基-2 快速算法完整的蝶形运算图,其中输入为自然顺序,输出为倒位序,并标出正确的支路系数。

7. 已知 $x(n)$为 8 点有限长序列,其定义如下:

$$x(n)=\begin{cases}1, & 0\leqslant n\leqslant 7\\ 0, & \text{其他 } n\end{cases}$$

若 z 平面螺线路径参数如下:

$$A_0=0.8,\quad W_0=1.2,\quad \theta_0=\frac{\pi}{3},\quad \phi_0=\frac{\pi}{10}$$

试完成如下计算:

(1) 采用 CZT 算法求其前 10 点的复频谱 $X(z_k)$;

(2) 根据 CZT 实现原理,绘出 z_k 的路径图。

8. CZT 算法可用于计算 N 点有限长序列 $h(n)$在 z 平面实轴上各 z_k 点的 z 变换 $H(z)$,使

(a) $z_k=ak$,$k=0,1,\cdots,N-1$,a 为实数,$a\neq 0$

(b) $z_k=a^k$,$k=0,1,\cdots,N-1$,a 为实数,$a\neq\pm 1$

(c) (a)和(b)都可以

(d) (a)和(b)都不可以

上述说法中哪一说法是正确的?并简要分析原因。

9. 对信号 $x(t)=\mathrm{e}^{-0.1t}$,$t\geqslant 0$ 进行频谱分析。

(1) 根据傅里叶变换求出其频谱;

(2) 若以 $T=0.75$s 的抽样间隔对 $x(t)$采样,求离散信号频谱的重复周期 Ω_s。

10. 已知序列 $h(n)$ 为 $N=8$ 点的有限长序列,$H(\mathrm{e}^{\mathrm{j}\omega})=\mathrm{DTFT}[h(n)]$。如果要计算 $H(\mathrm{e}^{\mathrm{j}\omega})$在频率 $\omega_k=\frac{2\pi}{64}k^2$($k=0,1,\cdots,7$)共 8 个频率抽样点处的值,要求采用绿色计算,即不能采用先算出超过 8 个抽样点,再舍弃一些点的计算方法,试分析采用 CZT 算法是否可行?并简要说明理由。

11. 已知 FIR 滤波器的单位抽样响应 $h(n)$为 $M=60$ 的有限长序列,若用该滤波器对一串很长的数据进行滤波,并采用基于重叠保留法的 FFT 实现这一滤波功能,离散傅里叶

变换为 128 点。为了实现该滤波功能：(1)各输入段必须重叠多少个抽样点？(2)数据如何分段？(3)从每一段卷积结果中取出多少个点才能使这些值连接在一起得到正确的卷积序列(滤波器的输出)？(4)若对于长数据已分段为每段 120 点，应如何处理？

12. 已知序列 $x(n)$和 $h(n)$如下：

$$x(n)=\{2,8,2,1,8,2,1,2,1,6\},\quad h(n)=\{-1,0,1\}$$

试用基于重叠相加法的 FFT 算法计算序列 $x(n)$和 $h(n)$的线性卷积。

13. 已知序列 $x(n)$和 $h(n)$如下：

$$x(n)=\{2,1,2,1,8,2,1,6,2\},\quad h(n)=x(n-3)$$

试用 FFT 计算序列 $x(n)$和 $h(n)$的互相关序列。