

第五章 快速傅里叶变换FFT

本章主要内容

- **FFT算法的基本原理**
- **基2 FFT算法的理论推导**
- **按时间抽取的FFT算法**
- **按频率抽取的FFT算法**
- **IDFT的快速运算方法**
- **FFT在线性卷积、相关中的应用**

5.1 FFT算法的基本原理

■ 矩阵方程 - DFT的计算量分析

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

上式表示 N 个方程的计算。若 $N=4$ ，上式可以写为如下形式

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

式中 $W = e^{-j(2\pi/N)}$ 。为方便起见，这里用 W^{nk} 替代 W_N^{nk} 。

可进一步表示成矩阵形式

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

把前面的矩阵形式重写如下

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1A)$$

或更紧凑地表示成

$$X(k) = W^{nk} x(n)$$

考察式(5.1A)，可以看到，由于 W 是复数， $x(n)$ 也可能是复数，因此

- 1、每计算一个 $X(k)$ ，需要 N 次复数乘法， $N-1$ 次复数加法；
- 2、完成矩阵运算，即整个DFT运算，就需要 N^2 次复数乘法和 $N(N-1)$ 次复数加法。
- 3、每个复数乘法包含4次实数乘法和2次实数加法，每个复数加法包含2次实数加法

例如：

当 $N=8$ ，复乘次数=64，复加次数=56；

当 $N=1024$ ，复乘次数 $(1024)^2=1048576$ ，复加次数 $(1024 \times (1024-1))=1047552$

DFT的复数相乘与复数相加的运算复杂度都为 $O(N^2)$ ，计算量大，难以做到实时处理

■ 直观上讨论如何减少计算DFT所需的乘法和加法次数

把矩阵(5.1A) 重写如下

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1A)$$

直观推导（第一步）： 利用关系 $W^{nk} = W^{((nk))_N}$ ，因此，如 $N = 4, n = 2, k = 3$ 时，
则 $W^6 = W^2$

这是因为

$$\begin{aligned} W^{nk} &= W^6 = e^{(-j\frac{2\pi}{4})(6)} = e^{-j3\pi} \\ &= W^{((nk))_N} = W^2 = e^{(-j\frac{2\pi}{4})(2)} = e^{-j\pi} \end{aligned}$$

于是，矩阵(5.1A)可改写为

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1B)$$

直观推导 (第二步)

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1C)$$

将式(5.1C)中的矩阵 $[W^{nk}]$
分解因子成如下形式

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1D)$$

式(5.1D)与式(5.1C)比较, 可以看到第一行和第二行相互交换了, 式(5.1D)改写了列矢量 $X(k)$ 。交换后的矢量用 $\overrightarrow{X(k)}$ 表示, 即:

$$\overrightarrow{X(k)} = \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \quad (5.1E)$$

这个因子分解是FFT算法之所以有效的关键 (注意其变换结果 $\overrightarrow{X(k)}$ 是乱序的)

重写式 (5.1D)

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1D)$$

下面进一步讨论计算上述这个方程的乘法次数

首先设 (中间计算步骤)

$$\begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1F)$$

即, 列矢量 $x_1(n)$ 等于式(5.1D)右边两个矩阵的乘积, 元素 $x_1(0)$ 的计算要用一次复数乘法和一次复数加法来确定, 即

$$x_1(0) = x(0) + W^0 x(2)$$

为了推得普遍的结果, 这里 W^0 不化为1

(5.1G)

$x_1(1)$ 的计算也是用一次复数乘法和一次复数加法来确定, 但计算 $x_1(2)$ 只要一次复数加法, 这是因为, $W^0 = -W^2$, 因此

$$x_1(2) = x(0) + W^2 x(2) = x(0) - W^0 x(2) \quad (5.1H)$$

上式中的复数乘法 $W^0 x(2)$ 已在计算 $x_1(0)$ 时计算过了。

同理，计算 $x_1(3)$ 也只要一次复数加法，不需要乘法。这样中间矢量 $x_1(n)$ 只需要四次复数加法，不需作乘法。

下面继续完成式(5.1D)的计算（计算列矢量 $x_2(0)$ ）

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (5.1D)$$

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} x_2(0) \\ x_2(1) \\ x_2(2) \\ x_2(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} \quad (5.1I)$$

$x_2(0)$ 项可用一次复数乘法和一次复数加法来确定，即

$$x_2(0) = x_1(0) + W^0 x_1(1) \quad (5.1J)$$

$x_2(1)$ 的计算，只要一次复数加法，因为， $W^0 = -W^2$ 。同样， $x_2(2)$ 只要一次复数乘法和一次复数加法，而 $x_2(3)$ 的计算只要一次复数加法。

因此，用式(5.1D)计算 $\vec{X}(k)$ ，总共需4次复数乘法和8次复数加法，而直接计算DFT需要16次复数乘法和12次复数加法。

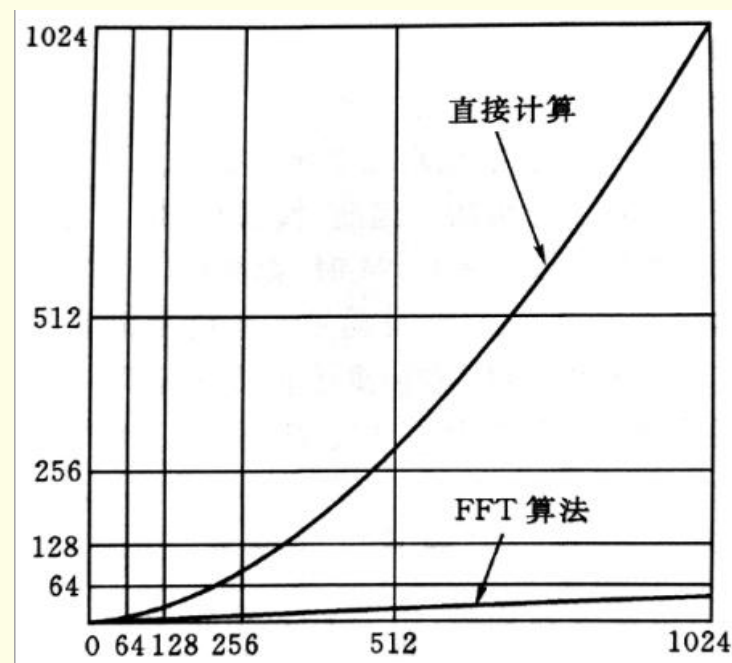
在前面计算式(5.1D)时, 矩阵分解因子过程中, 由于把零引入了被分解的矩阵, 从而减少了乘法次数。

- 对于 $N = 2^M$ 的FFT算法, 简单地讲, 就是将 $N \times N$ 的矩阵分解为 M 个 $N \times N$ 矩阵 (每一个矩阵的具有复数加法和复数加法次数最少的特性)
- 将大 N 的DFT分解为若干小点数的DFT的组合, 使整个DFT的计算过程变成一系列迭代运算过程
- 引伸上面例子的结果, 可以看到,
 $N=2^M$ 的FFT算法只需要 $\frac{1}{2} N \cdot M$ 次复数乘法和 $N \cdot M$ 次复数加法, 而直接计算DFT需要 N^2 次复数乘法和 $N(N-1)$ 次复数加法。

若只考虑乘法次数, 直接计算DFT与FFT的计算时间之比的近似关系

$$\frac{N^2}{N \cdot M/2} = \frac{2N}{M}$$

对于 $N=1024=2^{10}$, 可以使计算量减少到200比1
 计算复杂度从 $O(N^2)$ 下降为 $O(N \log(N))$



N 点DFT和FFT所需乘法次数的比较

在前面计算式(5.1D)时, 矩阵分解因子过程中, 引进了一个差异。在计算式(5.1D)时, 我们得到的是 $\overrightarrow{X(k)}$, 而不是 $X(k)$, 即

$$\overrightarrow{X(k)} = \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \quad \text{代替了} \quad X(k) = \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} \quad (5.1K)$$

这种重新排列, 是矩阵分解过程固有的。可直接对 $\overrightarrow{X(k)}$ 重新排序得到 $X(k)$ 。

对结果重新排序

用相应的二进制数代替自变量 k ，重写 $X(k)$ ，即

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \quad \text{变成} \quad \begin{bmatrix} X(00) \\ X(10) \\ X(01) \\ X(11) \end{bmatrix} \quad (5.1L)$$

将上面矩阵的二进制数**位序反转**，那么

$$\overrightarrow{X(k)} = \begin{bmatrix} X(00) \\ X(10) \\ X(01) \\ X(11) \end{bmatrix} \quad \text{反转} \quad \begin{bmatrix} X(00) \\ X(01) \\ X(10) \\ X(11) \end{bmatrix} = \mathbf{X}(k) \quad (5.1M)$$

对于 $N > 4$ 的FFT算法按式(5.1D)的方式描述矩阵因子分解过程比较麻烦，后面将用图解的方式讨论式(5.1D)的计算过程，用这种形式导出计算机的流程图。

■ 减少DFT计算量的依据

DFT的公式

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

改进DFT计算效率的多数方法利用了 W_N^{nk} 的周期性和对称性，即

W_N^{nk} 的周期性

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

W_N^{nk} 的共轭对称性

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

且由于 $W_N^{N/2} = e^{-j\pi} = -1$ ，有

$$W_N^{(k+N/2)} = -W_N^k$$

同时利用 W_N^{nk} 的对称性和周期性可以大大减少DFT的计算量

举例 对实数序列 $x(n)$, 利用权重的对称性, 则可将其DFT公式中含有 n 和 $N-n$ 的项组合在一起

$$\underline{x(n)W_N^{nk} + x(N-n)W_N^{(N-n)k}}$$

$$= x(n)W_N^{nk} + x(N-n)(W_N^{nk})^*$$

$$= \underline{[x(n) + x(N-n)]\text{Re}(W_N^{nk}) + j[x(n) - x(N-n)]\text{Im}(W_N^{nk})}$$

原式需4次实数乘法

变形以后需2次实数乘法

对其它项也做类似的组合处理, 计算DFT的乘法次数就减少一半

5.2 按时间抽取的FFT算法

■ 算法原理 (基2-FFT)

先将 $x(n)$ 按 n 的奇偶分为两组, 设 $N=2^M$, 不足时补零, n 用变量 $2r$ 表示, 即

$$n \text{ 为偶数: } x(2r) = x_1(r), \quad r = 0, 1, \dots, \frac{N}{2} - 1$$

$$n \text{ 为奇数: } x(2r + 1) = x_2(r), \quad r = 0, 1, \dots, \frac{N}{2} - 1$$

因此有

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

由于 $W_N^2 = e^{-j\frac{2\pi}{N}2} = e^{-j2\pi/(\frac{N}{2})} = W_{N/2}$, 上式可表示为:

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x_1(r) W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x_2(r) W_{\frac{N}{2}}^{rk} = X_1(k) + W_N^k X_2(k)$$

式中 $k = 0, \dots, \frac{N}{2} - 1$

其中

$$\left\{ \begin{array}{l} X_1(k) = \sum_{r=0}^{\frac{N}{2}-1} x_1(r) W_{\frac{N}{2}}^{rk} = \text{DFT}(x_1(r)) \\ X_2(k) = \sum_{r=0}^{\frac{N}{2}-1} x_2(r) W_{\frac{N}{2}}^{rk} = \text{DFT}(x_2(r)) \end{array} \right.$$

■ 结论

- 1、 $X_1(k)$, $X_2(k)$ 均为 $N/2$ 点的DFT;
- 2、 $X(k) = X_1(k) + W_N^k X_2(k)$ 只能确定出 $X(k)$ 的 $k=0, 1, \dots, \frac{N}{2} - 1$ 的值, 即 $X(k)$ 前一半的结果。

■ $X(k)$ 后一半的计算

根据 W_N^{nk} 的周期性有 $W_{N/2}^{r(k+N/2)} = W_{N/2}^{rk}$, 所以有:

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

同理有 $X_2\left(\frac{N}{2} + k\right) = X_2(k)$, $k = 0, \dots, \frac{N}{2} - 1$

这就是说, $X_1(k)$ 、 $X_2(k)$ 的后一半, 分别等于其前一半的值。

又由于 $W_N^{(N/2+k)} = W_N^{N/2} W_N^k = -W_N^k$, 且 $X_1(k), X_2(k)$ 以 $N/2$ 为周期, 所以有:

$$X(k + \frac{N}{2}) = X_1(k + \frac{N}{2}) + W_N^{k+\frac{N}{2}} X_2(k + \frac{N}{2}) = X_1(k) - W_N^k X_2(k)$$

$$k = 0, 1, \dots, \frac{N}{2} - 1$$

可见, $X(k)$ 的后一半, 也完全由 $X_1(k), X_2(k)$ 所确定。

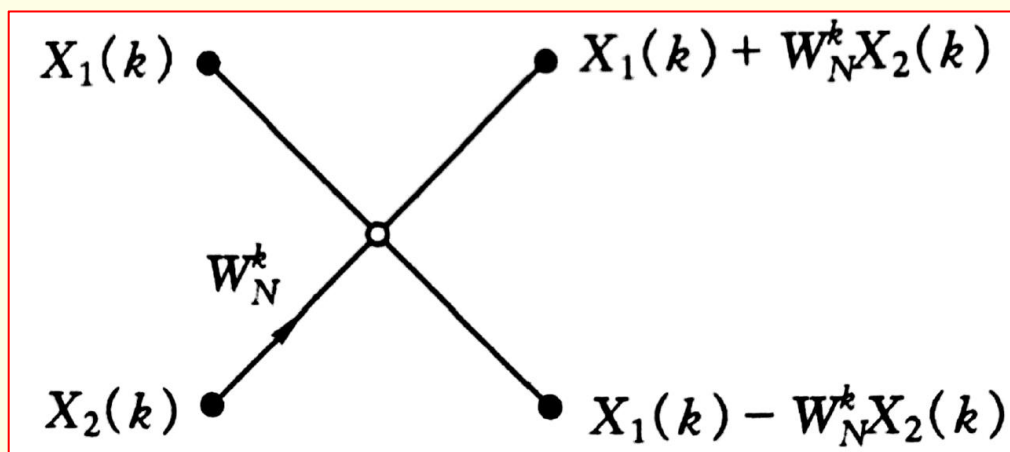
所以, N 点的 DFT 可由两个 $N/2$ 点的 DFT 来计算。

■ 蝶形运算

由 $X_1(k)$ 、 $X_2(k)$ 来表示 $X(k)$ 的运算如下

$$\begin{cases} X(k) = X_1(k) + W_N^k X_2(k) \\ X(k + \frac{N}{2}) = X_1(k) - W_N^k X_2(k) \end{cases} \quad (k = 0, 1, \dots, \frac{N}{2} - 1)$$

实现上式运算的流图称作**蝶形运算**



■ 举例

$N=8$ 序列的DFT, 可以分解为两个 $N/2=4$ 点的DFT, 具体方法如下:

1、 n 为偶数时的序列 $x_1(n)$ 为

$$x_1(0) = x(0), x_1(1) = x(2), x_1(2) = x(4), x_1(3) = x(6)$$

进行 $N/2=4$ 点的DFT得 $X_1(k)$

$$X_1(k) = \sum_{r=0}^3 x_1(r) W_4^{rk} = \sum_{r=0}^3 x(2r) W_4^{rk}, k = 0, 1, 2, 3$$

2、 n 为奇数时的序列 $x_2(n)$ 为

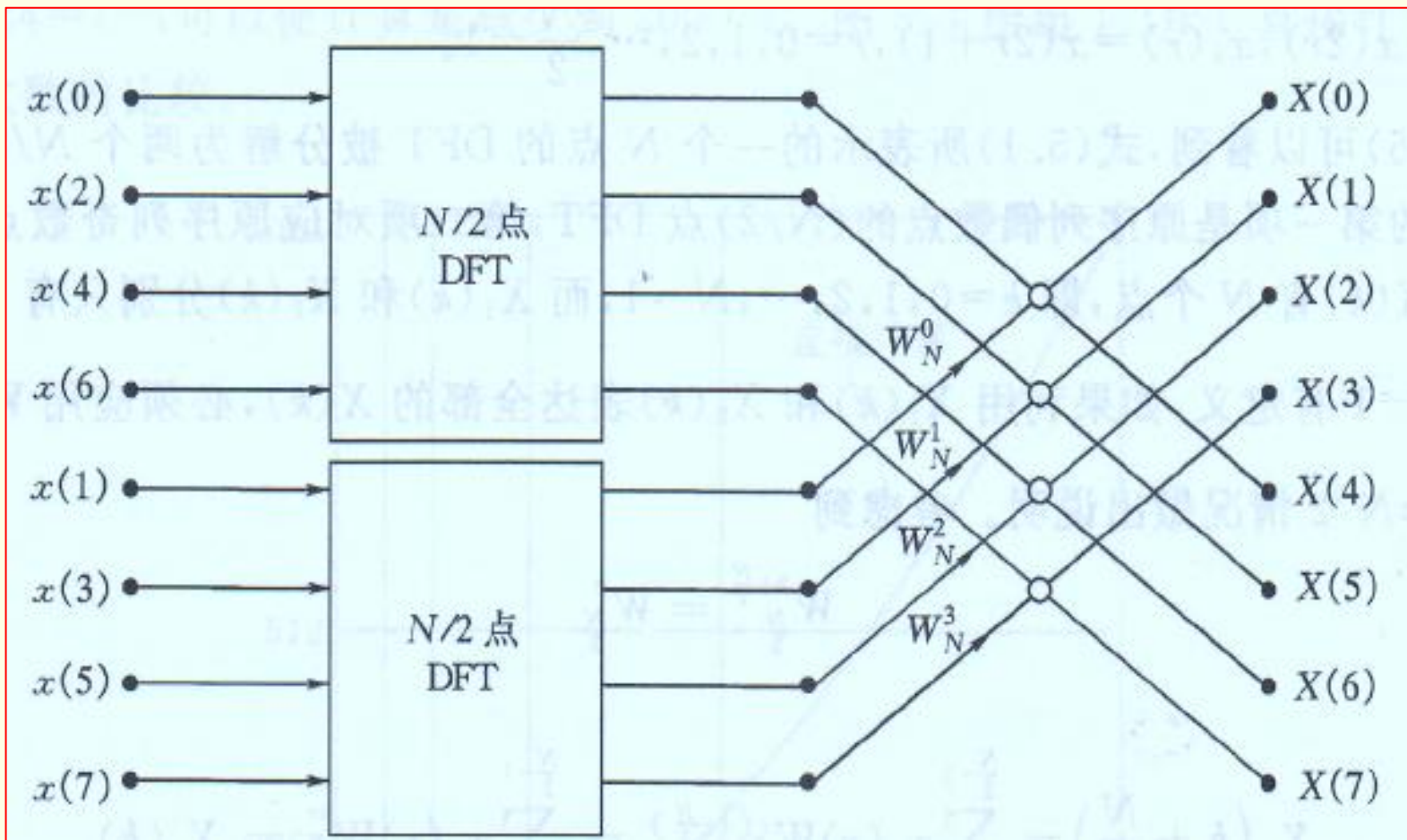
$$x_2(0) = x(1), x_2(1) = x(3), x_2(2) = x(5), x_2(3) = x(7)$$

进行 $N/2=4$ 点的DFT得 $X_2(k)$

$$X_2(k) = \sum_{r=0}^3 x_2(r) W_4^{rk} = \sum_{r=0}^3 x(2r+1) W_4^{rk}, k = 0, 1, 2, 3$$

根据蝶形运算, 即可由4点 $X_1(k)$ 和4点 $X_2(k)$ 来计算全部 $N=8$ 点 $X(k)$

3、对 $X_1(k)$ 和 $X_2(k)$ 进行蝶形运算，前半部为 $X(0)$ 到 $X(3)$ ，后半部分为 $X(4)$ 到 $X(7)$ ，整个过程如下图所示：



■ 迭代奇偶分组

按照奇偶分组的基本思想，不妨对每个 $N/2$ 的序列进一步奇偶分组，从而进一步减少运算量。假设序列总长 $N=2^L$ ，就可以这样分解 L 层。当不满足时，可以补零操作。

1、对前面的偶序列 $x_1(r)$, $r = 0, \dots, \frac{N}{2} - 1$ 进一步奇偶数分解，即

$$x_1(2l) = x_3(l), \quad l = 0, \dots, \frac{N}{4} - 1$$

$$x_1(2l+1) = x_4(l), \quad l = 0, \dots, \frac{N}{4} - 1$$

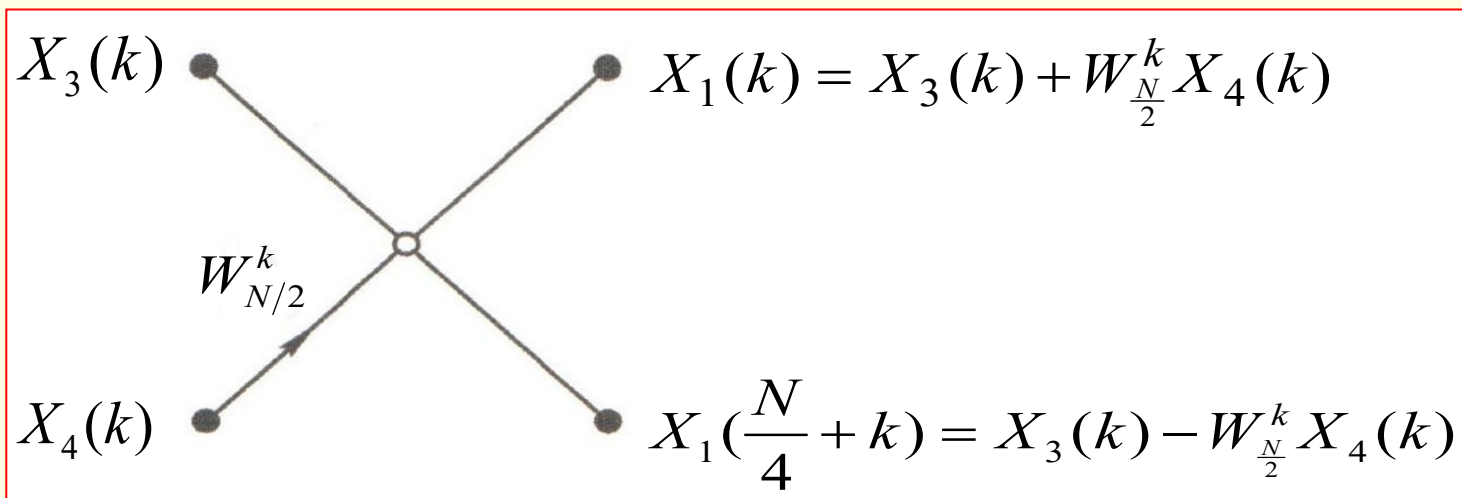
分别进行 $N/4$ 点的DFT，得到：

$$\begin{cases} X_3(k) = \sum_{l=0}^{\frac{N}{4}-1} x_3(l) W_{N/4}^{lk}, & k = 0, \dots, \frac{N}{4} - 1 \text{ (偶数序列中再取偶)} \\ X_4(k) = \sum_{l=0}^{\frac{N}{4}-1} x_4(l) W_{N/4}^{lk}, & k = 0, \dots, \frac{N}{4} - 1 \text{ (偶数序列中再取奇)} \end{cases}$$

利用蝶形运算，可用 $N/4$ 点的 $X_3(k)$ 和 $X_4(k)$ 计算 $N/2$ 点的 $X_1(k)$ ，即

$$\begin{cases} X_1(k) = X_3(k) + W_{\frac{N}{2}}^k X_4(k) \\ X_1(\frac{N}{4} + k) = X_3(k) - W_{\frac{N}{2}}^k X_4(k) \end{cases} \quad k = 0, 1, \dots, \frac{N}{4} - 1$$

其蝶形运算流图如下



2、同样对前面的奇序列 $x_2(r)$, $r = 0, \dots, \frac{N}{2} - 1$ 进一步奇偶数分解为 $N/4$ 的序列 $x_5(l)$ 和 $x_6(l)$, 并分别进行 $N/4$ 点的DFT如下

$$\left\{ \begin{array}{l} X_5(k) = \sum_{l=0}^{\frac{N}{4}-1} x_2(2l) W_{N/4}^{lk} = \sum_{l=0}^{\frac{N}{4}-1} x_5(l) W_{N/4}^{lk} \quad (\text{奇数序列中再取偶序号子序列}) \\ X_6(k) = \sum_{l=0}^{\frac{N}{4}-1} x_2(2l+1) W_{N/4}^{lk} = \sum_{l=0}^{\frac{N}{4}-1} x_6(l) W_{N/4}^{lk} \quad (\text{奇数序列中再取奇序号子序列}) \end{array} \right.$$

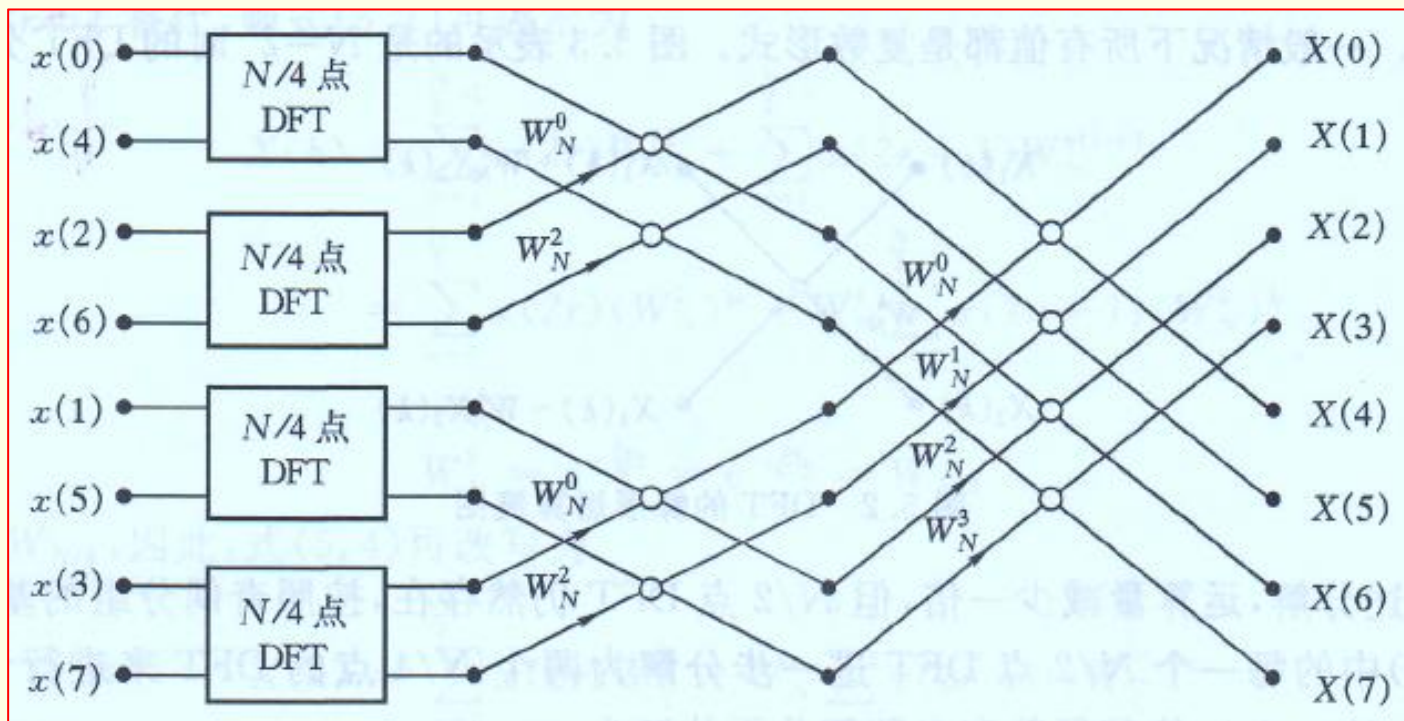
由 $X_5(k)$ 、 $X_6(k)$ 进行蝶形运算计算 $N/2$ 点的 $X_2(k)$, 得到

$$\left\{ \begin{array}{l} X_2(k) = X_5(k) + W_{N/2}^k X_6(k) ; k = 0, 1, \dots, \frac{N}{4} - 1 \\ X_2(\frac{N}{4} + k) = X_5(k) - W_{N/2}^k X_6(k) ; k = 0, 1, \dots, \frac{N}{4} - 1 \end{array} \right.$$

所以, N 点的DFT分解为四个 $N/4$ 点的DFT来计算。

3、两级蝶形计算流图如下

(为减少参数个数, 设 $W_{N/2}^0 = W_N^0$, $W_{N/2}^1 = W_N^2$)



经过上述两级蝶形运算, 用四个 $N/4$ 点 DFT 计算 N 点序列的 DFT, 其运算量可再减少约一半, 即为 N 点 DFT 计算量的 $1/4$ 。

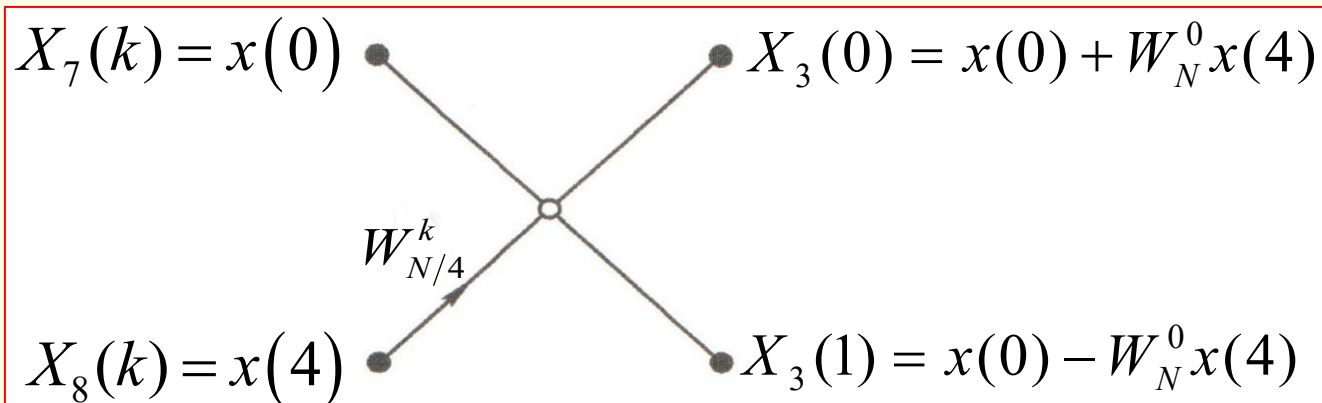
4、最后一级——2点DFT的蝶形运算

对于 $N=2^3=8$ 时DFT，经过2级迭代，即为 $N/4$ 点即为两点DFT。

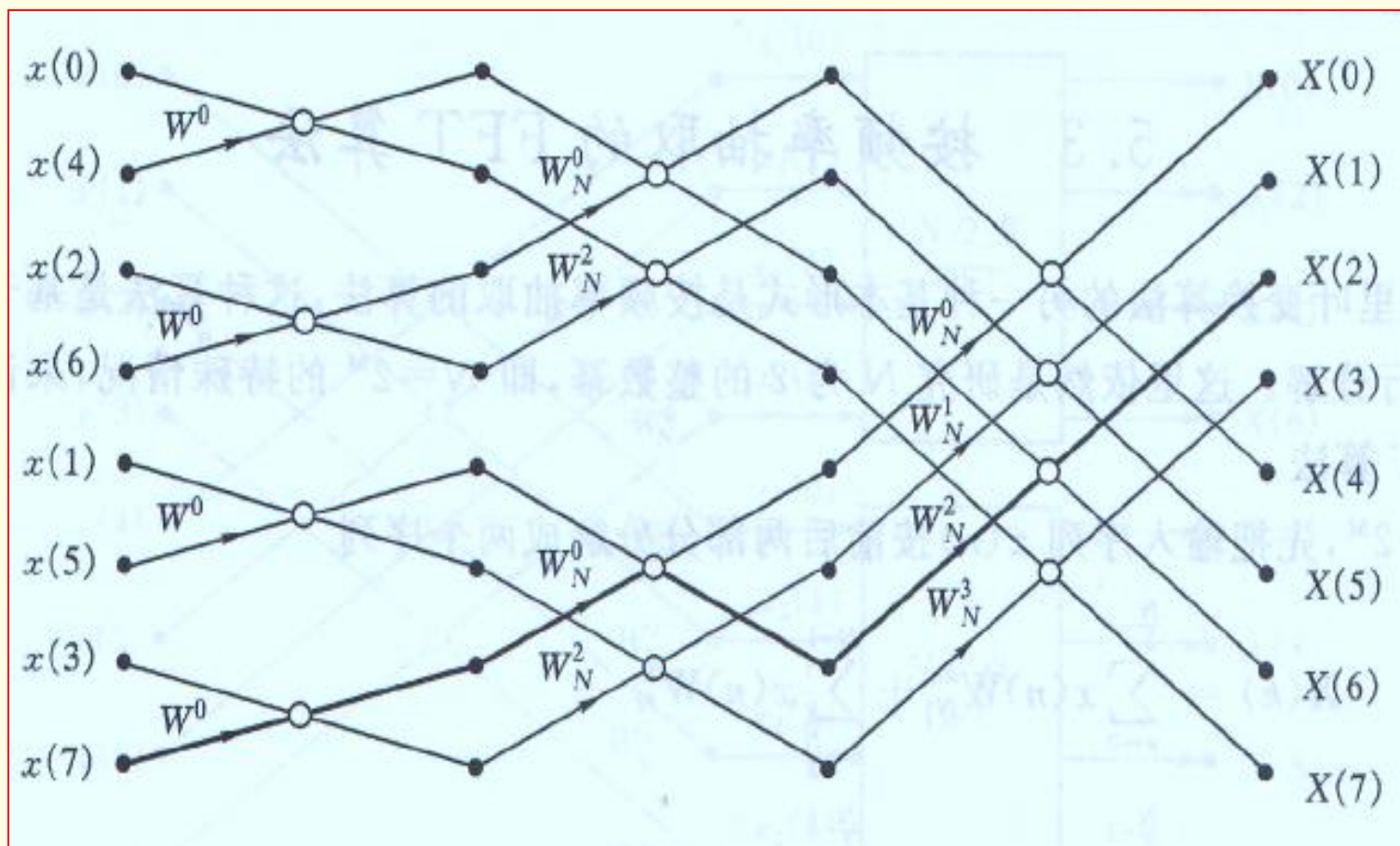
以 $X_3(k) = \sum_{l=0}^1 x_3(l)W_2^{lk}$ ， $k=0,1$ 为例，即

$$\begin{cases} X_3(0) = x_7(0) + W_{N/4}^0 x_8(0) = x(0) + W_N^0 x(4) \\ X_3(1) = x_7(0) - W_{N/4}^0 x_8(0) = x(0) - W_N^0 x(4) \end{cases}$$

2点DFT仍可以用蝶形运算分解为单点DFT，进一步减少运算量，即



5、最终，得到 $N=2^M=8$ 点DFT分解为 $M=3$ 级迭代的蝶形运算流程图如下：



这种FFT算法, 是在时间上对输入序列的次序是属于偶数还是属于奇数来进行分解的, 所以称作按时间抽取的算法(DIT)。

二. 运算量

由上述分析可知, $N=8$ 需三级蝶形运算 $N=2^3=8$, 由此可知, $N=2^L$ 共需 L 级蝶形运算, 而且每级都由 $N/2$ 个蝶形运算组成, 每个蝶形运算有一次复乘, 两次复加。

因此, N点的FFT的运算量为

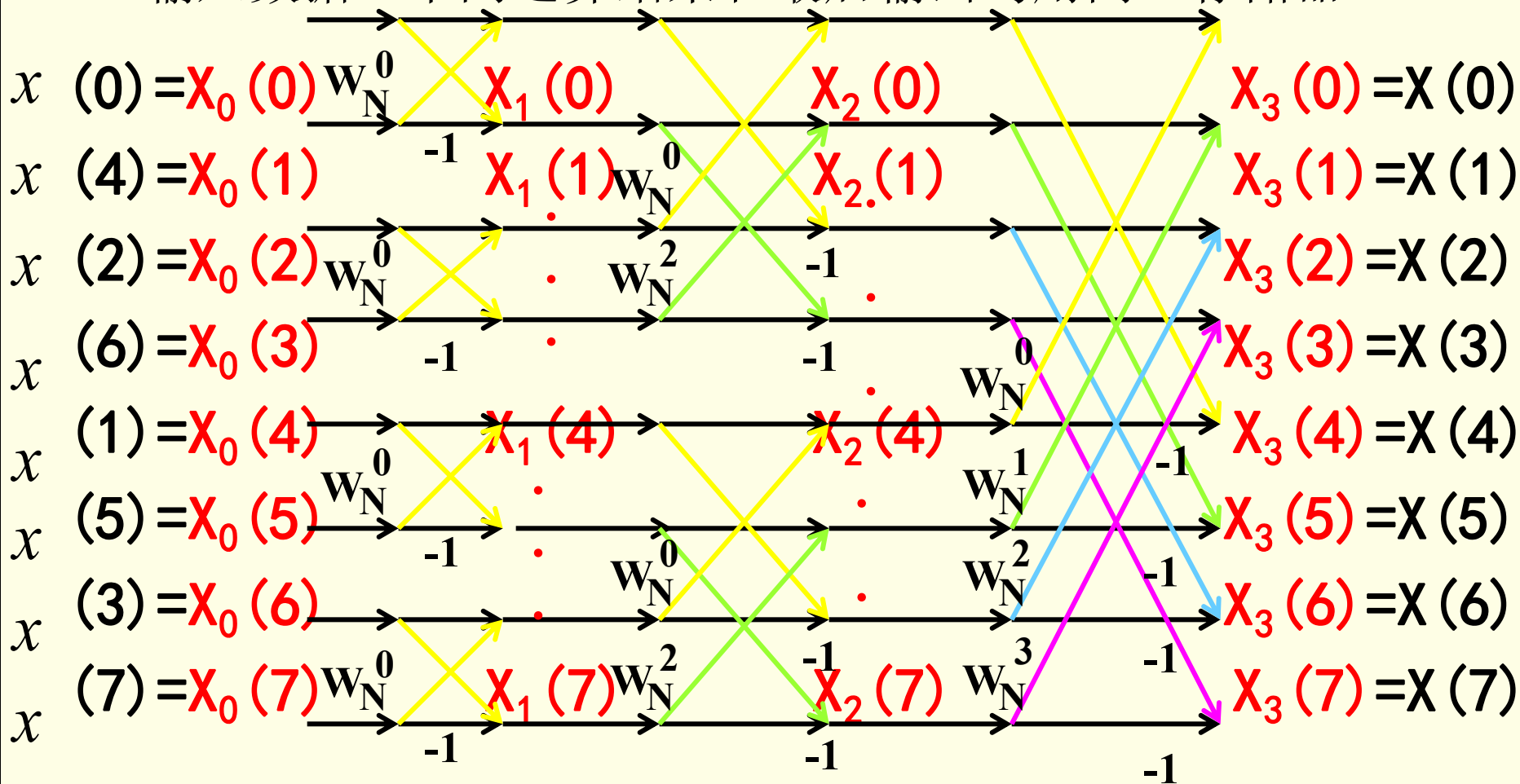
$$\text{复乘: } m_F = (N/2) L = (N/2) \log_2 N$$

$$\text{复加: } a_F = N L = N \log_2 N$$

三. DIT的FFT算法的特点

1. 原位运算

输入数据、中间运算结果和最后输出均用同一存储器。



由运算流程图可知, 一共有**N个输入/出行**, 一共有 **$\log_2 N=L$ 列 (级) 蝶形运算 (基本迭代运算)**.

设用 $m(m=1, 2, \dots, L)$ 表示第 m 列; 用 k, j 表示蝶形输入数据所在的 (上/下) 行数 ($0, 1, 2, \dots, N-1$); 这时任何一个蝶形运算可用下面通用式表示, 即

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

$$\begin{aligned} x(0) &= X_0(0), & x(1) &= X_0(4), \\ x(4) &= X_0(1), & x(5) &= X_0(5), \\ x(2) &= X_0(2), & x(3) &= X_0(6), \\ x(6) &= X_0(3), & x(7) &= X_0(7). \end{aligned}$$

所以, 当 $m=1$ 时, 则有 (前两个蝶形)

$$\begin{cases} X_1(0) = X_0(0) + X_0(1)W_N^0 \\ X_1(1) = X_0(0) - X_0(1)W_N^0 \end{cases}$$

$$\begin{cases} X_1(2) = X_0(2) + X_0(3)W_N^0 \\ X_1(3) = X_0(2) - X_0(3)W_N^0 \end{cases}$$

当 $m=2$ 时, 则有 (前两个蝶形)

$$\begin{cases} X_2(0) = X_1(0) + X_1(2)W_N^0 \\ X_2(2) = X_1(0) - X_1(2)W_N^0 \\ X_2(1) = X_1(1) + X_1(3)W_N^2 \\ X_2(3) = X_1(1) - X_1(3)W_N^2 \end{cases}$$

当 $m=3$ 时, 则有 (前两个蝶形)

$$\begin{cases} X_3(0) = X_2(0) + X_2(4)W_N^0 \\ X_3(4) = X_2(0) - X_2(4)W_N^0 \\ X_3(1) = X_2(1) + X_2(5)W_N^1 \\ X_3(5) = X_2(1) - X_2(5)W_N^1 \end{cases}$$

可见, 在某列进行蝶形运算的任意两个节点(行) k 和 j 的节点变量 $X_{m-1}(k), X_{m-1}(j)$ 就完全可以确定蝶形运算的结果 $X_m(k), X_m(j)$, 与其它行(节点)无关。

这样, 蝶形运算的两个输出值仍可放回蝶形运算的两个输入所在的存储器中, 即实现所谓原位运算。每一组(列)有 $N/2$ 个蝶形运算, 所以只需 N 个存储单元, 可以节省存储单元。

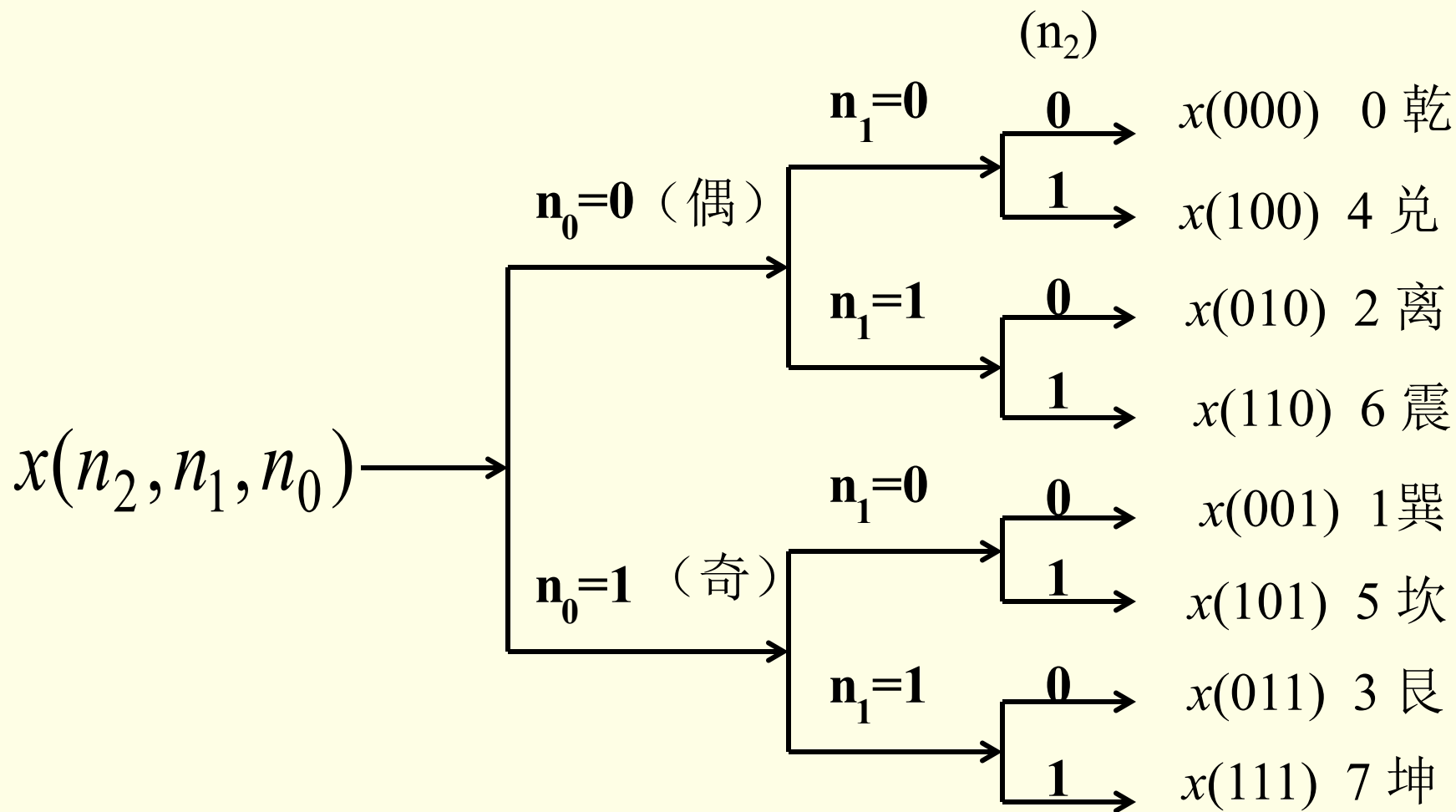
2 倒位序规律

由图可知, 输出 $X(k)$ 按正常顺序排列在存储单元, 而输入是按顺序:

$x(0), x(4), x(2), x(6)$; $x(1), x(5), x(3), x(7)$;

这种顺序称作倒位序, 即二进制数倒位。

这是由奇偶分组造成的, 以 $N=8$ 为例
说明如下:



3. 倒位序实现

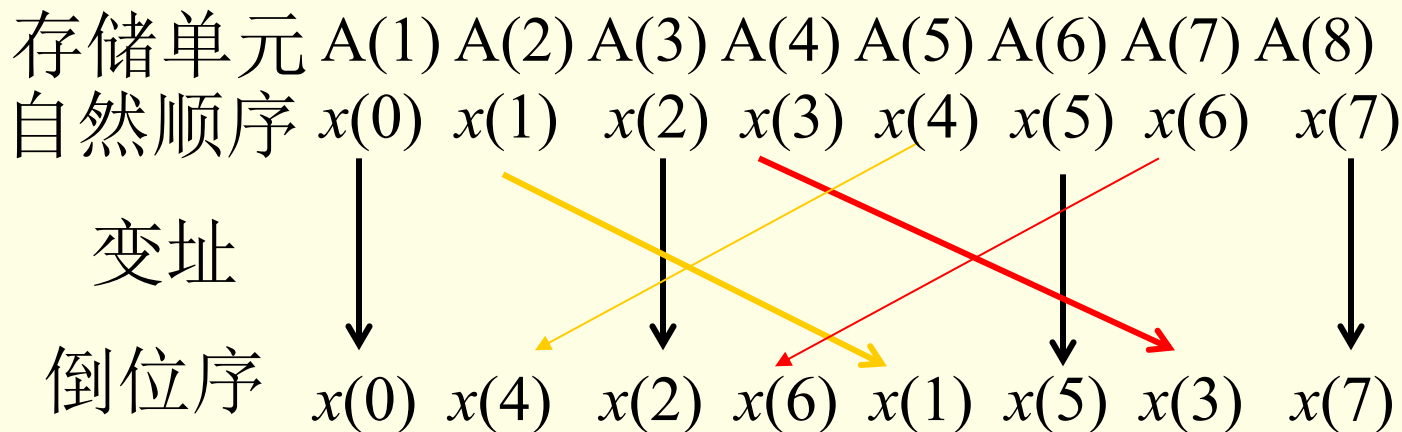
输入序列先按自然顺序存入存储单元, 然后经变址运算来实现倒位序排列

设输入序列的序号为 n , 二进制为

$(n_2 \ n_1 \ n_0)_2$, 倒位序顺序用 \hat{n} 表示, 其倒位序二进制为 $(n_0 \ n_1 \ n_2)_2$, 例如 ,
N=8时如下表:

自然顺序 n	二进制 $n_2 \ n_1 \ n_0$	倒位序二进制 $n_0 \ n_1 \ n_2$	倒位顺序 n
0	0 0 0	0 0 0	0
1	0 0 1	1 0 0	4
2	0 1 0	0 1 0	2
3	0 1 1	1 1 0	6
4	1 0 0	0 0 1	1
5	1 0 1	1 0 1	5
6	1 1 0	0 1 1	3
7	1 1 1	1 1 1	7

变址处理方法



4. 蝶形运算两节点的距离: 2^{m-1}

其中, m 表示第 m 列, 且 $m = 1, \dots, L$

例如 $N=8=2^3$, 第一级(列)距离为 $2^{1-1}=1$,

第二级(列)距离为 $2^{2-1}=2$,

第三级(列)距离为 $2^{3-1}=4$ 。

5. W_N^r 的确定 (仅给出方法)

考虑蝶形运算两节点的距离为 2^{m-1} , 蝶形运算可表为

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

由于 N 为已知, 所以将 r 的值确定即可。

为此, 令 $k = (n_2 n_1 n_0)_2$, 再将 $k = (n_2 n_1 n_0)_2$ 左移 $(L-m)$ 位, 右边位置补零, 就可得到 $(r)_2$ 的值, 即 $(r)_2 = (k)_2 2^{L-m}$ 。

例如 $N=8=2^3$

(1) $k=2$, $m=3$ 的r值, $\because k=2=(010)_2$
左移 $L-m=3-3=0$, $\therefore r=(010)_2=2$;

(2) $k=3$, $m=3$ 的r值; $k=3=(011)_2$, 左
移0位, $r=3$;

(3) $k=5$, $m=2$ 的值; $k=5=(101)_2$ 左移
 $L-m=1$ 位 $r=(010)_2=2$ 。

6. 存储单元

存输入序列 $x(n)$, $n=0, 1, \dots, N-1$,
计 N 个单元;

存放系数 W_N^r , $r=0, 1, \dots, (N/2) - 1$,
需 $N/2$ 个存储单元;

共计 $(N+N/2)$ 个存储单元。

5.3 按频率抽取的FFT算法

■ 算法原理 (基2-FFT)

1、把 $N=2^M$ 的输入序列 $x(n)$ 按前后两部分分解为2个 $N/2$ 长的短序列, $x(n)$ 的DFT可以计算为

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{n=N/2}^{N-1} x(n) W_N^{nk} \quad \text{前后分组} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x(n + \frac{N}{2}) W_N^{(n+\frac{N}{2})k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x(n + \frac{N}{2}) W_N^{\frac{N}{2}k} \right] W_N^{nk} \end{aligned}$$

由于 $W_N^{\frac{N}{2}k} = \left(W_N^{\frac{N}{2}}\right)^k = (-1)^k$

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

2、将 N 点DFT输出序列 $X(k)$ 按 k 的奇偶分组分为两个 $N/2$ 点DFT

当 k 为偶数, 即 $k=2r$ 时, 有 $(-1)^k = 1$

当 k 为奇数, 即 $k=2r+1$ 时, 有 $(-1)^k = -1$

这样, $X(k)$ 可分为两部分:

k 为偶数时

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

k 为奇数时

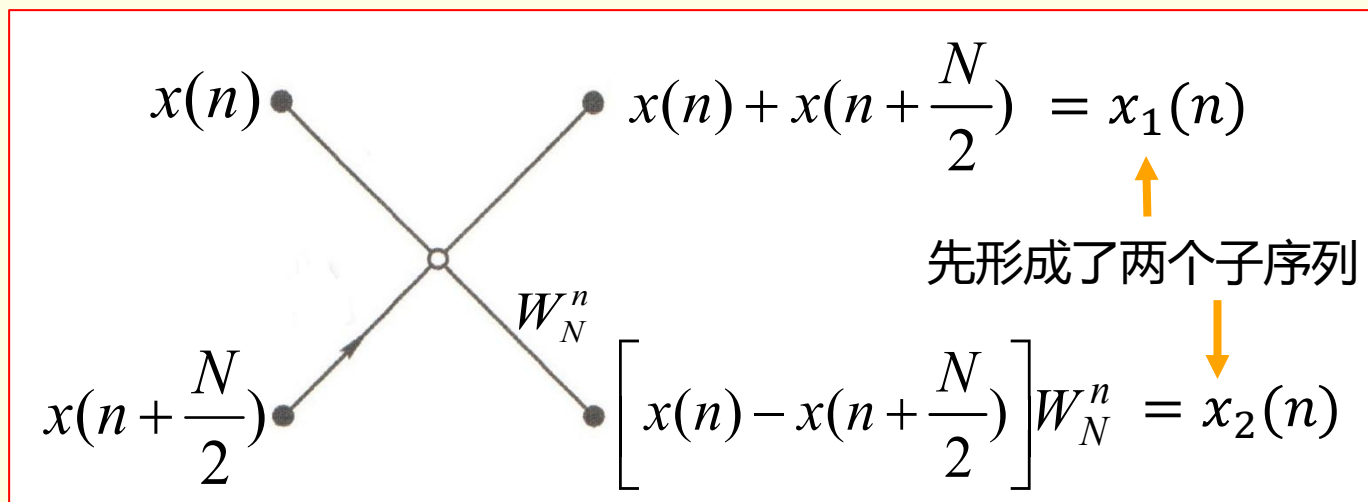
$$\begin{aligned} X(2r+1) &= \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) - x\left(n + \frac{N}{2}\right) \right] W_N^{n(2r+1)} \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ \left[x(n) - x\left(n + \frac{N}{2}\right) \right] W_N^n \right\} W_{\frac{N}{2}}^{nr}, \quad 0 \leq r \leq \frac{N}{2} - 1 \end{aligned}$$

上面两式均满足 $N/2$ 点 DFT 的定义式

3、蝶形运算

$$\begin{cases} X(2r) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n + \frac{N}{2}\right) \right] W_{\frac{N}{2}}^{nr} \\ X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ \left[x(n) - x\left(n + \frac{N}{2}\right) \right] W_N^n \right\} W_{\frac{N}{2}}^{nr} \end{cases} \quad 0 \leq n \leq \frac{N}{2}-1$$

对 $x(n)$ 和 $x(n + \frac{N}{2})$ 进行如下蝶形运算



4、 $N=8$ 时的计算流图（先形成子序列，得到两个4点的DFT运算）

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

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

$$x_1(n) = x(n) + x\left(n + \frac{N}{2}\right)$$

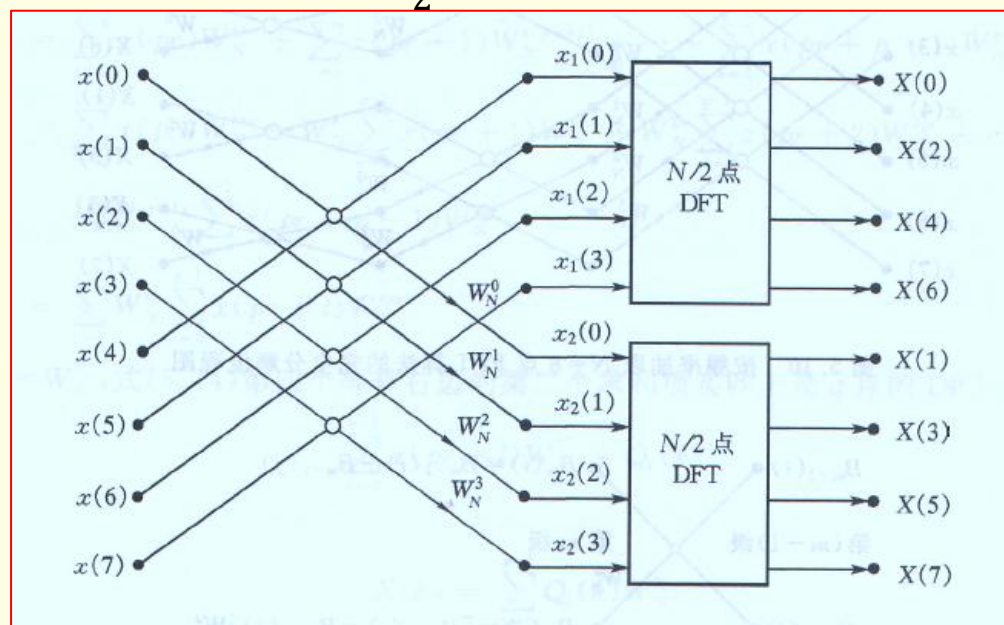
$$x_2(n) = [x(n) - x\left(n + \frac{N}{2}\right)] W_N^n$$

$$\text{子序列 } 0 \leq n \leq \frac{N}{2} - 1$$

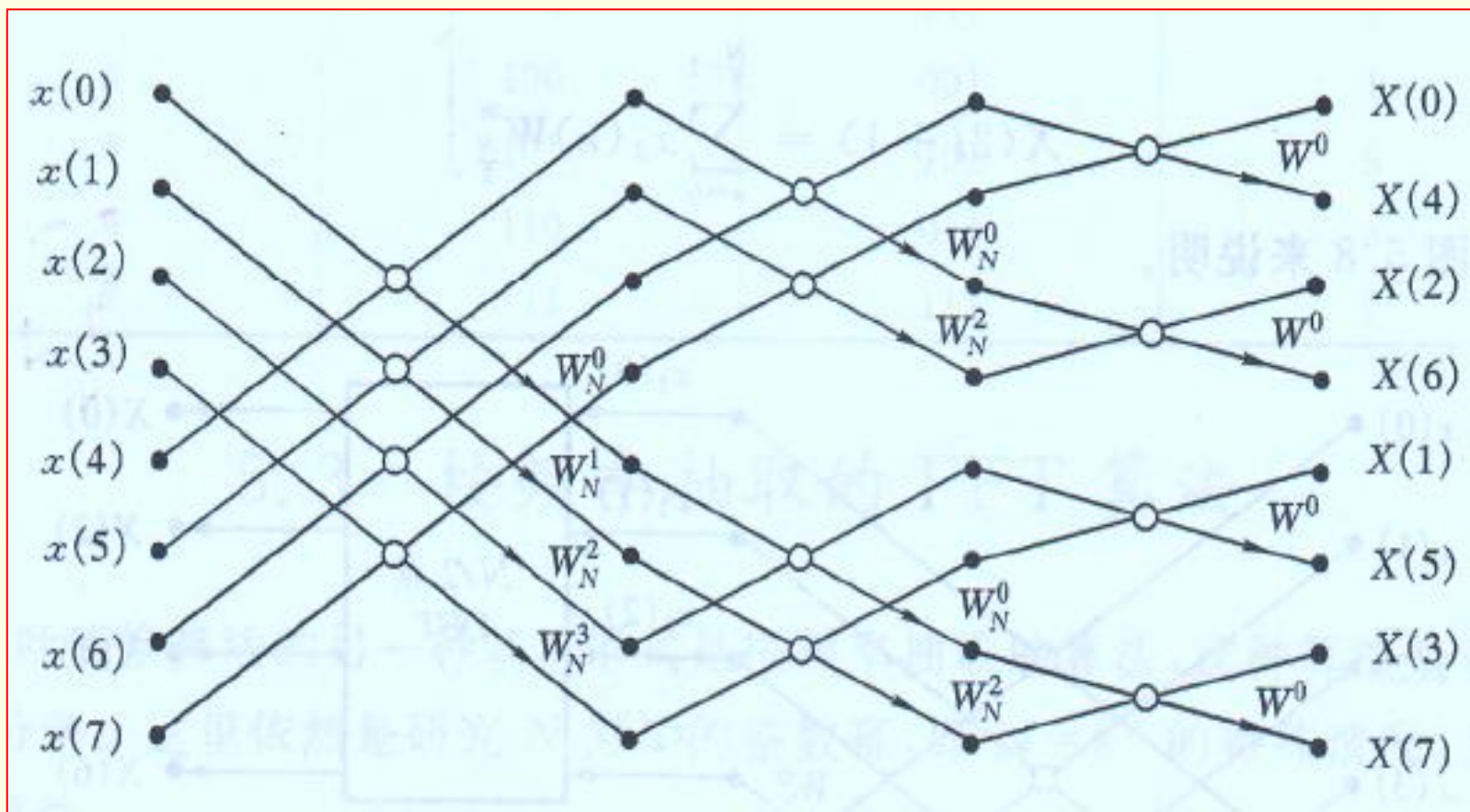
因为 $W_N^2 = W_{\frac{N}{2}}$ ，于是，由上面两组公式得到两个 $\frac{N}{2}$ 点的DFT运算，即

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} x_1(n) W_{\frac{N}{2}}^{rn}$$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} x_2(n) W_{\frac{N}{2}}^{rn}$$



- 5、按照时域抽取FFT的思路，再将 $N/2$ 点DFT按 k 的奇偶进一步分解为两个 $N/4$ 点的DFT，如此迭代进行下去，直至分解为单点DFT。得到 $N=8$ 时按频域抽取FFT的完整计算流图：



■ 权函数 W_N^p 的确定

设迭代蝶形运算的次数为 M ($1 \leq l \leq M$)次, 计算序列的长度为 $N=2^M$, l 表示计算阵列的第 l 列。

权函数 W_N^p 的计算主要是确定 p 值, p 值计算的一种方法如下:

- 1、把 p 值表示成 M 位的二进制数 $(p)_2$, 其中 M 应满足: $M=\log_2 N$, N 为处理点数;
- 2、将 $(p)_2$ 右移 $(M-l)$ 位, 并把左边的空位补零, 结果依然为 M 位;
- 3、将移位补零的 M 位二进制数进行比特倒置;
- 4、倒置后的二进制数转换成十进制数即得到 p 值。

■ 一节点的权函数是 W_N^p , 其对偶节点的权函数必然为 $W_N^{p+N/2}$,

而且 $W_N^p = -W_N^{p+N/2}$, 所以对偶节点可按下式计算

$$\begin{cases} x_l(k) = x_{l-1}(k) + W_N^p x_{l-1}(k + N/2^l) \\ x_l(k + N/2^l) = x_{l-1}(k) - W_N^p x_{l-1}(k + N/2^l) \end{cases}$$

■ 两种FFT的主要异同

1、不同点

倒位序不同：按时域抽取FFT的输入为倒位序，输出为自然顺序；按频域抽取FFT的输入为自然顺序，输出为倒位序。

蝶形运算形式不同。

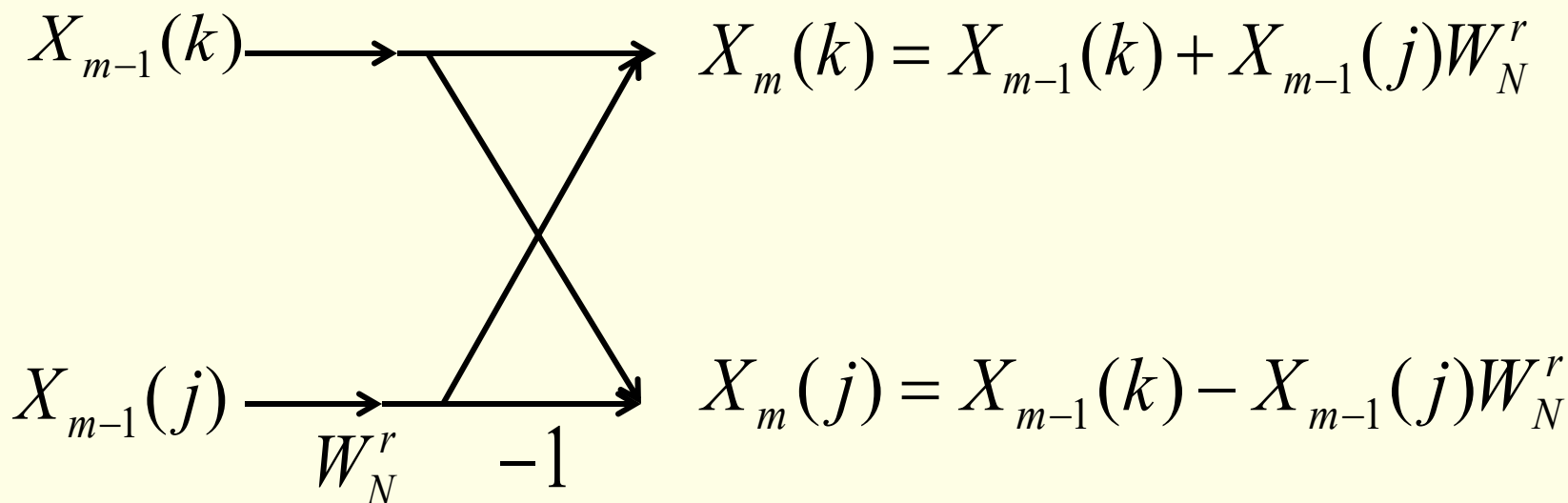
2、相同点

运算量相同，均为 $(N/2)\log_2 N$ 次复乘， $N\log_2 N$ 次复加；

两种FFT形式上具有左右（输入、输出）**对称性**。

(2) 蝶形运算不同

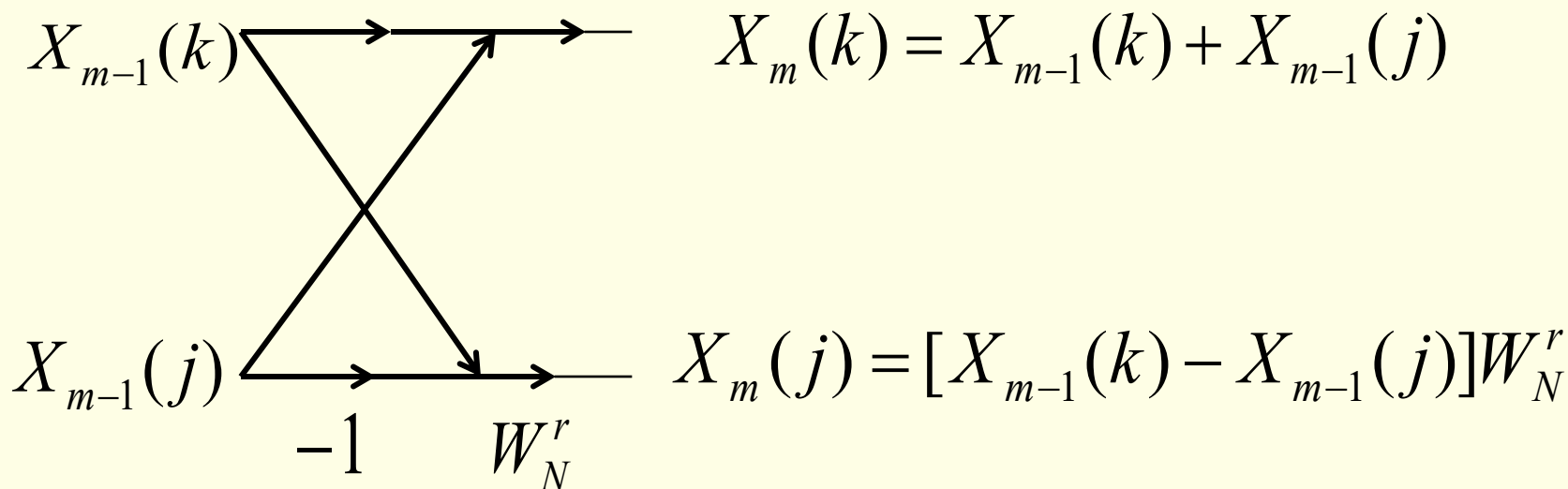
a. (DIT)



用矩阵表示

$$\begin{bmatrix} X_m(k) \\ X_m(j) \end{bmatrix} = \begin{bmatrix} 1 & W_N^r \\ 1 & -W_N^r \end{bmatrix} \begin{bmatrix} X_{m-1}(k) \\ X_{m-1}(j) \end{bmatrix}$$

b. (DIF)



用矩阵表示

$$\begin{bmatrix} X_m(k) \\ X_m(j) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ W_N^r & -W_N^r \end{bmatrix} \begin{bmatrix} X_{m-1}(k) \\ X_{m-1}(j) \end{bmatrix}$$

(3) 两种蝶形运算的关系

互为转置（矩阵）；

a. (DIT)

$$\begin{bmatrix} 1 & W_N^r \\ 1 & -W_N^r \end{bmatrix}$$

b. (DIF)

$$\begin{bmatrix} 1 & 1 \\ W_N^r & -W_N^r \end{bmatrix}$$

将流图的所有支路方向都反向, 交换输入和输出, 即可得到另一种蝶形。

5.4 IDFT的快速运算方法

■ 算法原理

$$\begin{aligned} X(k) &= \text{DFT}[x(n)] = \sum_{n=0}^{N-1} x(n)W_N^{nk} \\ x(n) &= \text{IDFT}[X(k)] = \frac{1}{N} \sum_{k=0}^{N-1} X(k)W_N^{-nk} \end{aligned}$$

比较两式可知，只要DFT的每个系数 W_N^{nk} 换成 W_N^{-nk} ，最后再乘以常数 $1/N$ 就可以得到IDFT的快速算法——IFFT。

可以将常数 $1/N$ 分配到每一级蝶形运算中， $1/N=(1/2)^L$ ，即每级蝶形运算均乘以 $1/2$ 。

■ 不改FFT程序直接实现IFFT

由于 $[W_N^{-nk}]^* = W_N^{nk}$, $[A \cdot B]^* = A^* \cdot B^*$

对IDFT取共轭, 得到

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

因此有

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

- 算法步骤:
- 1、先对 $X(k)$ 取共轭, 即将 $X(k)$ 的虚部乘-1;
 - 2、直接利用FFT程序计算其DFT;
 - 3、对计算结果再取一次共轭;
 - 4、最后再乘以常数 $1/N$, 即得 $x(n)$ 。

从而, FFT、IFFT可共用同一个子程序。

5.5 实数序列的FFT算法

通常考虑时间的实函数，而频率函数通常是复的，因此要设计一个即能减少计算DFT，又能计算IDFT的程序，就要假设一个复序列的DFT：

$$X(k) = \sum_{n=0}^{N-1} [x_r(n) + jx_i(n)] e^{-j2\pi nk/N}$$

因为，根据复数共轭的反变换公式可以得到：

$$x(n) = \frac{1}{N} \left[\sum_{k=0}^{N-1} [\operatorname{Re}(X(k)) + j\operatorname{Im}(X(k))]^* e^{-j2\pi nk/N} \right]^*$$

由于上面两式中，包含着共同的因子 $e^{-j2\pi nk/N}$ ，因此，用同一个程序就可以计算DFT和它的反变换。

■ 利用复时间函数的虚部, 使实函数的FFT计算有更高的效率

□ 同时计算两个实序列的FFT

我们希望用下列复序列的形式：

$$v(n) = x(n) + jy(n)$$

同时计算两个实序列 $x(n)$ 和 $y(n)$ 的DFT。也就是说， $v(n)$ 是由两个实函数组成，其中一个实函数作为虚部。

按DFT的线性性质， $v(n)$ 的DFT由下式给出：

$$\begin{aligned} V(k) &= \sum_{n=0}^{N-1} [x(n) + jy(n)] e^{-j\frac{2\pi}{N}kn} \\ &= \sum_{n=0}^{N-1} \left[\underset{\color{red}{V_1}}{x(n)\cos\left(\frac{2\pi}{N}kn\right)} + \underset{\color{red}{V_2}}{y(n)\sin\left(\frac{2\pi}{N}kn\right)} \right] \\ &\quad + j \sum_{n=0}^{N-1} \left[\underset{\color{red}{V_3}}{y(n)\cos\left(\frac{2\pi}{N}kn\right)} - \underset{\color{red}{V_4}}{x(n)\sin\left(\frac{2\pi}{N}kn\right)} \right] \\ & (= V_1(k) + V_2(k) + j[V_3(k) - V_4(k)] = V_R(k) + jV_I(k)) \end{aligned}$$

把前面公式的求和式改写为

$$\begin{aligned} V(k) &= V_1(k) + V_2(k) + j[V_3(k) - V_4(k)] \\ &= V_R(k) + jV_I(k) \end{aligned}$$

同理有

$$\begin{aligned} V(N-k) &= V_1(k) - V_2(k) + j[V_3(k) + V_4(k)] \\ &= V_R(N-k) + jV_I(N-k) \end{aligned}$$

由于

$$\begin{cases} X(k) = \sum_{n=0}^{N-1} x(n) \left[\cos\left(\frac{2\pi}{N}kn\right) - j \sin\left(\frac{2\pi}{N}kn\right) \right] \\ Y(k) = \sum_{n=0}^{N-1} y(n) \left[\cos\left(\frac{2\pi}{N}kn\right) - j \sin\left(\frac{2\pi}{N}kn\right) \right] \end{cases}$$

可得

$$\begin{cases} X(k) = V_1(k) - jV_4(k) \\ Y(k) = V_3(k) - jV_2(k) \end{cases}$$

最后可解得

$$\begin{cases} X(k) = \frac{1}{2}[V_R(k) + V_R(N-k)] + j\frac{1}{2}[V_I(k) - V_I(N-k)] \\ Y(k) = \frac{1}{2}[V_I(k) + V_I(N-k)] + j\frac{1}{2}[V_R(k) - V_R(N-k)] \end{cases}$$

而

$$v(n) = x(n) + jy(n) \xrightarrow{\text{FFT}} V_R(k) + jV_I(k)$$

将 $X(k)$ 和 $Y(k)$ 重新表示如下

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \left[\cos\left(j \frac{2\pi}{N} kn\right) - j \sin\left(j \frac{2\pi}{N} kn\right) \right] \\ &= \left(\frac{V_R(k)}{2} + \frac{V_R(N-k)}{2} \right) + j \left(\frac{V_I(k)}{2} - \frac{V_I(N-k)}{2} \right) \\ Y(k) &= \sum_{n=0}^{N-1} y(n) \left[\cos\left(j \frac{2\pi}{N} kn\right) - j \sin\left(j \frac{2\pi}{N} kn\right) \right] \\ &= \left(\frac{V_I(k)}{2} + \frac{V_I(N-k)}{2} \right) - j \left(\frac{V_R(k)}{2} - \frac{V_R(N-k)}{2} \right) \end{aligned}$$

因此，假若一个复时间序列的DFT的实部和虚部按照上面两式分解，那么，两个实时间序列的DFT就能同时完成。显而易见，这种方法使效率提高了1倍，只是需要将最后的计算结果分离一下。

□ 同时计算两个实序列的DFT的过程

1. 函数 $x(n)$ 和 $y(n)$ 是实序列, $n = 0, 1, \dots, N - 1$ 。
2. 构成复序列: $v(n) = x(n) + jy(n)$, $n = 0, 1, \dots, N - 1$
3. 计算

$$V(k) = \sum_{n=0}^{N-1} v(n)e^{-j2\pi nk/N} = V_R(k) + jV_I(k), \quad k = 0, 1, \dots, N - 1$$

式中 $V_R(k)$ 和 $jV_I(k)$ 分别为 $V(k)$ 的实部和虚部。

4. 计算

$$\begin{cases} X(k) = \frac{1}{2}[V_R(k) + V_R(N-k)] + j\frac{1}{2}[V_I(k) - V_I(N-k)] \\ Y(k) = \frac{1}{2}[V_I(k) + V_I(N-k)] + j\frac{1}{2}[V_R(k) - V_R(N-k)] \end{cases} \quad k = 0, 1, \dots, N - 1$$

这里 $X(k)$ 和 $Y(k)$ 分别为 $x(n)$ 和 $y(n)$ 的DFT。

□ 用 N 点变换计算 $2N$ 个样本的DFT

考虑一个用 $2N$ 个样本点描述的序列 $x(n)$ ，其 $2N$ 点的DFT变换为

$$X(k) = \sum_{n=0}^{2N-1} x(n)W_{2N}^{kn}, \quad k = 0, 1, 2, \dots, 2N-1$$

我们希望用 N 点的DFT来计算序列 $x(n)$ 的DFT。

也就是说，我们希望把 $2N$ 点的序列 $x(n)$ 分解为两个 N 点的序列。但是，序列 $x(n)$ 不能简单地分成两半，而要按 n 的奇偶分为两组，即

$$\begin{cases} x(2r) = u(r) \\ x(2r+1) = v(r) \end{cases}, \quad r = 0, 1, 2, \dots, N-1$$

式中序列 $u(r)$ 等于 $x(r)$ 的偶数号样本点，而 $v(r)$ 等于奇数号样本点（注意， $u(r)$ 和 $v(r)$ 并不是由 $x(n)$ 分解所得的偶函数和奇函数）。

把长度为 $2N$ 的实序列 $x(n)$ 分解为两个 N 点实序列（奇偶两组） 计算：

$$\begin{aligned}
 X(k) &= \sum_{n=0}^{2N-1} x(\textcolor{red}{n})e^{-j2\pi k\textcolor{red}{n}/2N} = \sum_{r=0}^{N-1} x(\textcolor{red}{2r})e^{-j2\pi k(\textcolor{red}{2r})/2N} + \sum_{r=0}^{N-1} x(\textcolor{red}{2r+1})e^{-j2\pi k(\textcolor{red}{2r+1})/2N} \\
 &= \sum_{r=0}^{N-1} x(\textcolor{red}{2r})e^{-j2\pi k(\textcolor{red}{r})/N} + e^{-j\pi k/N} \sum_{r=0}^{N-1} x(\textcolor{red}{2r+1})e^{-j2\pi k(\textcolor{red}{r})/N} \\
 &= \sum_{r=0}^{N-1} u(\textcolor{red}{r})e^{-j2\pi k(\textcolor{red}{r})/N} + e^{-j\pi k/N} \sum_{r=0}^{N-1} v(\textcolor{red}{r})e^{-j2\pi k(\textcolor{red}{r})/N} \\
 &= U(k) + e^{-j\pi k/N} V(k), \quad k = 0, 1, 2, \dots, N-1
 \end{aligned}$$

其中 $U(k)$ 、 $V(k)$ 是两个长度为 N 的实序列 $u(r)$ 、 $v(r)$ 的DFT，可以通过前面讨论的“同时计算两个实序列的DFT的方法” 实现。

最后得到

$$\begin{aligned}
 X(k) &= U(k) + e^{-j\pi k/N} V(k), \quad k = 0, 1, 2, \dots, N-1 \\
 X(N+k) &= U(k) - e^{-j\pi k/N} V(k), \quad k = 0, 1, 2, \dots, N-1
 \end{aligned}$$

5.6 利用FFT实现线性卷积、相关的快速计算

■ 利用FFT实现线性卷积的快速计算

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

其中, $x(n)$ 长度为 L , $h(n)$ 长度为 M , $L \geq M$ 。

步骤:

- 1、分别对 $x(n)$ 、 $h(n)$ 补零点至长度至少为 $N=M+L-1$ 点
- 2、用FFT求 $H(k)=\text{FFT}[h(n)]$
- 3、用FFT求 $X(k)=\text{FFT}[x(n)]$
- 4、求 $Y(k)=X(k)H(k)$
- 5、用IFFT求 $y(n)=\text{IFFT}[Y(k)]$

■ 利用FFT实现线性相关的快速计算

$$\begin{aligned} r_{xy}(m) &= \sum_{n=0}^{L-1} x(n+m)y^*(n) = \sum_{n=0}^{L-1} x(n)y^*(n-m) \\ &= x(m) * y^*(-m) \quad (\text{把线性相关转换到卷积表示}) \end{aligned}$$

其中, $x(n)$ 长度为 L , $y(n)$ 长度为 M , $L \geq M$ 。

步骤:

- 1、将 $x(n)$ 、 $y(n)$ 补零点至长度至少为 $N=M+L-1$ 点
- 2、用FFT求 $X(k)=\text{FFT}[x(n)]$
- 3、用FFT求 $Y(k)=\text{FFT}[y(n)]$
- 4、求 $R_{xy}(k)=X(k)Y^*(k)$
- 5、用IFFT求 $r_{xy}(n)=\text{IFFT}[R_{xy}(k)]$

5.7 Chirp- z 变换

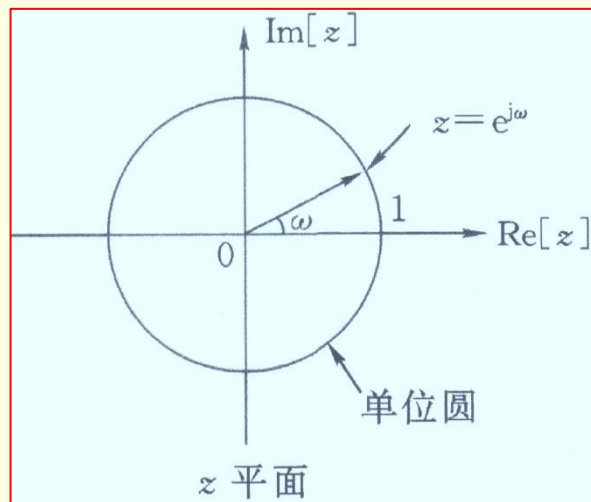
■ 回顾：DFT与标准 z 变换的关系

z 变换
$$Z[x(n)] = X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

DFT
$$X(k) = X(z)|_{z=e^{j\frac{2\pi}{N}k}=W_N^{-k}} = X\left(e^{j\frac{2\pi}{N}k}\right) = \sum_{n=0}^{N-1} x(n)W_N^{kn}$$

在 z 变换中, $z = re^{j\omega}$, 取单位圆 $r=1$, 对单位圆进行等间隔采样 $\omega = \frac{2\pi}{N}k$, 并取结果的主值区间 $k=[0, N-1]$, 得 $z = e^{j\frac{2\pi}{N}k} = W_N^{-k}$, 即得DFT。

在实际中, 有时对一个时间序列的某个频率分段感兴趣, 就会要求这个分段的采样频率与其它分段不一样。



■ Chirp-z变换的定义

$$Z[x(n)] = X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

在 z 变换中, 使 z 沿一段螺线作等角采样, 即采样点:

$$\mathbf{z}_k = A\mathbf{W}^{-k}, \quad k = 0, \dots, M-1$$

其中: $A = A_0 e^{j\theta_0}$ $W = W_0 e^{-j\phi_0}$

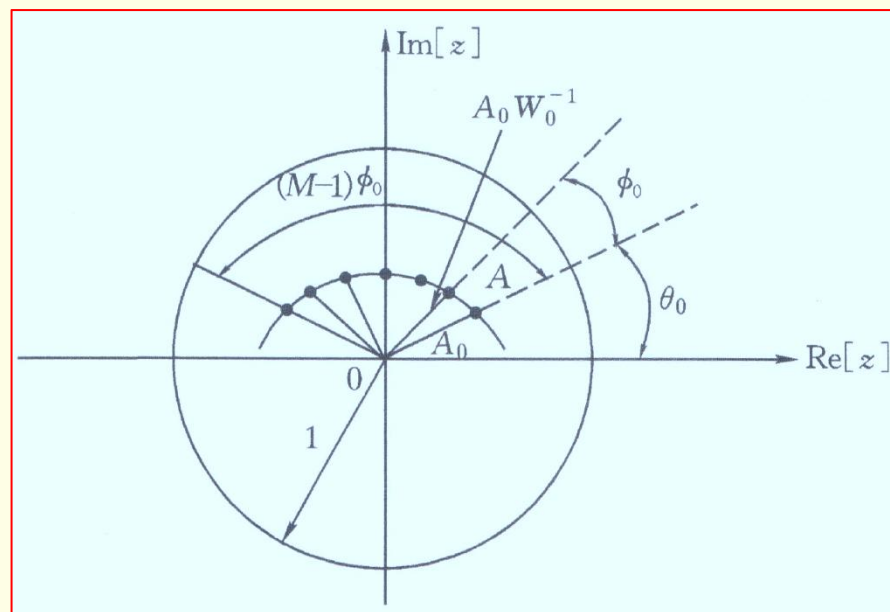
参数含义:

A_0 ——起始采样点的矢量半径

 θ_0 ——起始采样点的相角 ϕ_0 ——两相邻采样点间的角度差 W_0 ——螺线的伸展率，

$W_0 > 1$ 时, 螺线内缩;

$W_0 < 1$ 时, 螺线外伸。



对Chirp- z 变换 $z_k = AW^{-k}$, $k = 0, \dots, M - 1$, 可以看出当取 $A = 1$, $W = e^{-j\frac{2\pi}{N}}$ 时有 $z_k = e^{j\frac{2\pi}{N}k}$, $k = 0, \dots, M - 1$, Chirp- z 变换即退化成为DFT。

■ Chirp- z 变换的优势

- 1、可以只计算单位圆上感兴趣的一小段频谱的采样，而非整个单位圆，这特别适用于高分辨率窄带信号；
- 2、可以计算远离单位圆的任意点处的频谱，特别适用于语音及雷达信号。

Chirp- z 变换的快速算法

CZT的定义

$$X(z_k) = \sum_{n=0}^{N-1} x(n) z_k^{-n} = \sum_{n=0}^{N-1} x(n) A^{-n} W^{nk},$$

$$k = 0, \dots, M-1$$

Bluestein等式

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

(为了提高计算速度)

得到

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

令

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

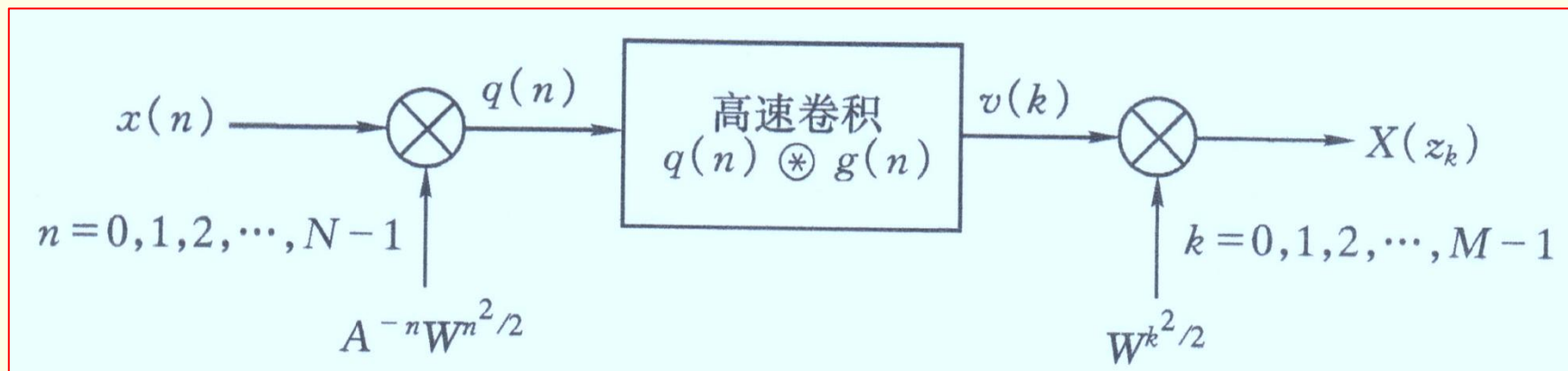
$$h(n) = W^{-\frac{n^2}{2}}$$

得到

$$X(z_k) = W^{\frac{k^2}{2}} \sum_{n=0}^{N-1} g(n) h(k-n) = W^{\frac{k^2}{2}} [g(k) * h(k)]$$

用FFT求解线性卷积即可实现CZT的快速计算

■ 基于卷积计算CZT变换的运算过程



卷积的长度 $L \geq N + M - 1$, 若取 $L = N + M - 1$, 则有

$$q(n) = \begin{cases} x(n)A^{-n}W^{n^2/2}, & n = 0,1,2,\dots,N-1 \\ 0, & n = N, N+1,\dots,L-1 \end{cases}$$

$$g(n) = \begin{cases} W^{-n^2/2}, & 0 \leq n \leq M-1 \\ W^{-(L-n)^2/2}, & L-N+1 \leq n < L \\ \text{任意值}, & \text{其他 } n \end{cases}$$

本章小结

- FFT算法的基本原理
- 基2 FFT算法的理论推导
- 按时间抽取的FFT算法
- 按频率抽取的FFT算法
- IDFT的快速运算方法
- FFT在线性卷积、相关中的应用