
第五章
快速傅里叶变换
@ohohd

本章目录

- 直接计算DFT的问题及改进的途径
- 按**时间抽取**的基2-FFT算法
- 按**频率抽取**的基2-FFT算法
- **快速**傅里叶逆变换(IFFT)算法
- **Matlab**实现

5.1 引言

- DFT在实际应用中很重要: 可以计算信号的频谱、功率谱和线性卷积等。
- 直接按DFT变换进行计算，当序列长度*N*很大时，计算量非常大，所需时间会很长。
- FFT并不是一种与DFT不同的变换，而是DFT的一种快速计算的算法。

对于一个长度为 N 的离散信号来讲，我们对其取离散傅里叶变换有：

$$X(k) := \mathbf{DFT}\{x(n)\} = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}nk}, \quad k = 0, 1, \ldots, n-1 \quad (1)$$

其中 $\mathbf{DFT}\{\bullet\}$ 是 $\bullet$ 的**离散傅里叶变换**，其逆变换为：

$$x(n) = \mathbf{IDFT}\{X(k)\} = \frac{1}{N}\sum_{k=0}^{N-1} X(k) \cdot e^{j\frac{2\pi}{N}nk}, \quad n = 0, 1, \cdot, N-1 \quad (2)$$

其中 $\mathbf{IDFT}\{\bullet\}$ 是 $\bullet$ 的**离散逆傅里叶变换**。

从式 (1) 中我们可以发现，如果要求第 k 点的 $\mathbf{DFT}$ 值，我们需要做：

(i) :

$$X(k) = \sum_{n=0}^{N-1} \bullet \quad (3)$$

$N-1$ 次加法；和：

(ii) :

$$\underset{1}{x(0) \cdot e^{-j\frac{2\pi}{N}0k}} + \underset{2}{x(1) \cdot e^{-j\frac{2\pi}{N}1k}} + \ldots + \underset{N}{x(N-1) \cdot e^{-j\frac{2\pi}{N}(N-1)k}} \quad (4)$$

N 次乘法运算。这就意味着如果要计算 N 个 $X(k)$，$(k = 0, 1, \ldots, N-1)$ 的值那么总共需要计算 N^2 次的乘法和 $N \cdot (N-1)$ 次的加法。这样的计算量大小在 N 比较小的时候是体现不出来的，但是当 N 很大时这样的计算量是十分庞大的。所以，我们要设法改进算法，来减少运算量。

5.2 直接计算DFT的问题及改进的途径

- **DFT的运算量**

设复序列$x(n)$ 长度为N点，其DFT为

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

（1）计算一个$X(k)$ 值的运算量

复数乘法次数： N

复数加法次数： $N-1$

5.2.1 DFT的运算量

（2）计算全部N个$X(k)$值的运算量

复数乘法次数： N^2

复数加法次数： $N(N-1)$

（3）对应的实数运算量

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}=\sum_{n=0}^{N-1}[\mathrm{Re}\,x(n)+j\,\mathrm{Im}\,x(n)][\mathrm{Re}\,W_N^{nk}+j\,\mathrm{Im}\,W_N^{nk}]$$

$$=\sum_{n=0}^{N-1}\{[\mathrm{Re}\,x(n)\cdot\mathrm{Re}\,W_N^{nk}-\mathrm{Im}\,x(n)\cdot\mathrm{Im}\,W_N^{nk}]$$

$$+j[\mathrm{Re}\,x(n)\cdot\mathrm{Im}\,W_N^{nk}+\mathrm{Im}\,x(n)\cdot\mathrm{Re}\,W_N^{nk}]\}$$

一次复数乘法：4次实数乘法 ＋ 2次实数加法

一个$X(k)$：$4N$次实数乘法 ＋

$2N+2(N-1)=2(2N-1)$次实数加法

所以 整个N点DFT运算共需要：

实数乘法次数：$4N^2$

实数加法次数：$N\times 2(2N-1)=2N(2N-1)$

DFT运算量的结论

N点DFT的复数乘法次数举例

N	N2	N	N2
2	4	64	4049
4	16	128	16384
8	64	256	65 536
16	256	512	262 144
32	1028	1024	1 048 576

结论：当*N*很大时，其运算量很大，对实时性很强的信号处理来说，要求计算速度快，因此需要改进DFT的计算方法，以大大减少运算次数。

5.2.2 减少运算工作量的途径

主要原理是利用系数 W_N^{nk} 的以下特性对DFT进行分解：

（1）对称性

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

（2）周期性

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

（3）可约性

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

另外，

$$W_N^{N/2} = -1 \qquad W_N^{(k+N/2)} = -W_N^k$$

5.3 按时间抽取的基2-FFT算法

- 算法原理
- 按时间抽取基-2FFT算法与直接计算DFT运算量的比较
- 按时间抽取的FFT算法的特点
- 按时间抽取FFT算法的其它形式流程图

5.3.1 算法原理

设$N=2^L$，将$x(n)$按 n 的奇偶分为两组：

$$\begin{cases} x(2r)=x_1(r) \\ x(2r+1)=x_2(r) \end{cases} \qquad r=0,\ 1,\ \ldots,\ \frac{N}{2}-1$$

则

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

$$=\sum_{\substack{n=0\\ n\text{为偶数}}}^{N-1}x(n)W_N^{nk}+\sum_{\substack{n=0\\ n\text{为奇数}}}^{N-1}x(n)W_N^{nk}$$

$$=\sum_{\substack{n=0\\ n\text{为偶数}}}^{N-1}x(n)W_N^{nk}+\sum_{\substack{n=0\\ n\text{为奇数}}}^{N-1}x(n)W_N^{nk}$$

$$=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2rk}+\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_N^{(2r+1)k}$$

$$=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{\frac{N}{2}}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_{\frac{N}{2}}^{rk} \qquad =X_1(k)+W_N^kX_2(k)$$

式中，$X_1(k)$和$X_2(k)$分别是$x_1(n)$和$x_2(n)$的$N/2$的DFT。

另外，式中k的取值范围是：0，1，…，N/2－1 。

因此，$X(k)=X_1(k)+W_N^k X_2(k)$ 只能计算出X(k)的前一半值。

后一半X(k) 值， N/2 ， N/2 +1， …， N ？

利用

$$W_{N/2}^{r(N/2+k)} = W_{N/2}^{rk}$$

可得到

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

同理可得

$$X_2(\frac{N}{2}+k)=X_2(k)$$

考虑到

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

及前半部分$X(k)$

$$X(k)=X_1(k)+W_N^k X_2(k)$$

k=0， 1， …， N/2－1

因此可得后半部分$X(k)$

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

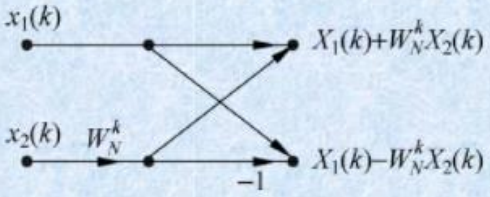
$$=X_1(k)-W_N^k X_2(k)$$

k=0， 1， …， N/2－1

蝶形运算

$$X(k)=X_1(k)+W_N^kX_2(k)$$
$$X(k)=X_1(k)-W_N^kX_2(k)$$

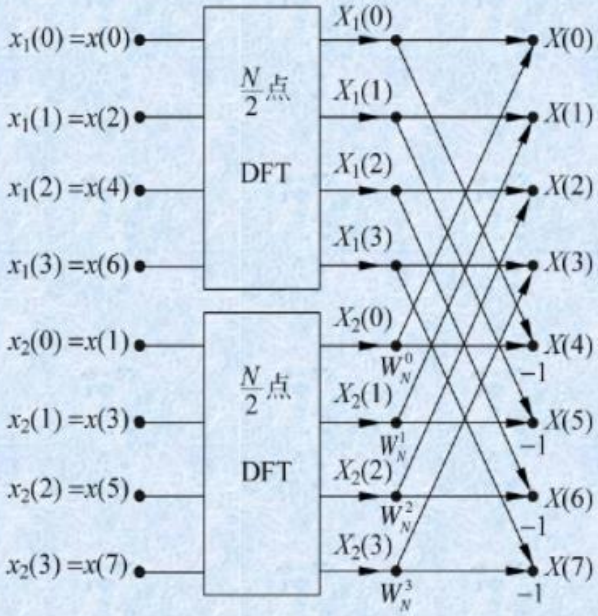
→ 蝶形运算式



蝶形运算信号流图符号

因此，只要求出2个N/2点的DFT，即X_1(k)和X_2(k)，再经过蝶形运算就可求出全部X(k)的值，运算量大大减少。

以8点为例第一次按奇偶分解



以N=8为例，分解为2个4点的DFT，然后做8/2=4次蝶形运算即可求出所有8点X(k)的值。

蝶形运算量比较

- *N*点DFT的运算量

  复数乘法次数： N^2

  复数加法次数： $N(N-1)$

- 分解一次后所需的运算量＝2个N/2的DFT＋N/2蝶形：

  复数乘法次数： $2*(N/2)^2+N/2=N^2/2+N/2$

  复数加法次数： $2*(N/2)(N/2-1)+2*N/2=N^2/2$

- 因此通过一次分解后，运算工作量减少了差不多一半。

进一步按奇偶分解

由于$N=2^L$，因而$N/2$仍是偶数，可以进一步把每个$N/2$点子序列再按其奇偶部分分解为两个$N/4$点的子序列。

以$N/2$点序列$x_1(r)$为例

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

则有

$$X_1(k)=\sum_{r=0}^{N/2-1}x_1(r)W_{N/2}^{rk}=\sum_{l=0}^{N/4-1}x_1(2l)W_{N/2}^{2lk}+\sum_{l=0}^{N/4-1}x_1(2l+1)W_{N/2}^{(2l+1)k}$$

$$=\sum_{l=0}^{N/4-1}x_3(l)W_{N/4}^{lk}+W_{N/2}^{k}\sum_{l=0}^{N/4-1}x_4(l)W_{N/4}^{lk}$$

$$=X_3(k)+W_{N/2}^{k}X_4(k)\qquad k=0,1,\ldots,\frac{N}{4}-1$$

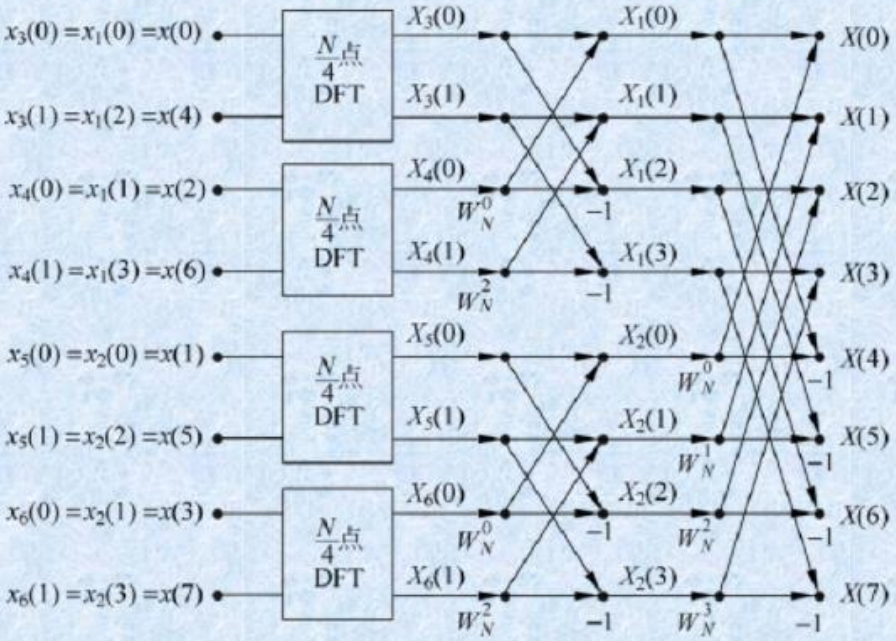
且

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

由此可见，一个$N/2$点DFT可分解成两个$N/4$点DFT。

同理，也可对$x_2(n)$进行同样的分解，求出$X_2(k)$。

以8点为例第二次按奇偶分解



算法原理

对此例N=8，最后剩下的是4个$N/4$= 2点的DFT，2点DFT也可以由蝶形运算来完成。以$X_3(k)$为例。

$$X_3(k)=\sum_{l=0}^{N/4-1}x_3(l)W_{N/4}^{lk}=\sum_{l=0}^{1}x_3(l)W_{N/4}^{lk}\qquad k=0,1$$

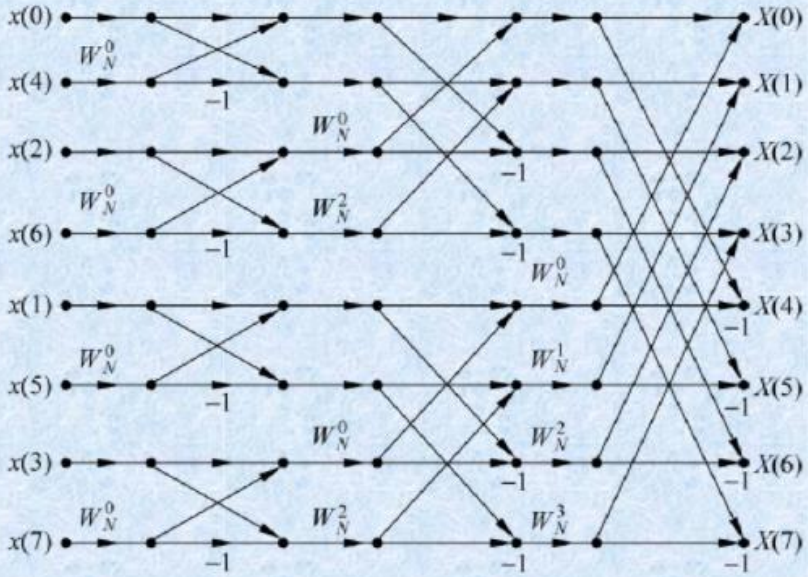
即

$$X_3(0)=x_3(0)+W_2^0x_3(1)=x(0)+W_2^0x(4)=x(0)+W_N^0x(4)$$

$$X_3(1)=x_3(0)+W_2^1x_3(1)=x(0)+W_2^1x(4)=x(0)-W_N^0x(4)$$

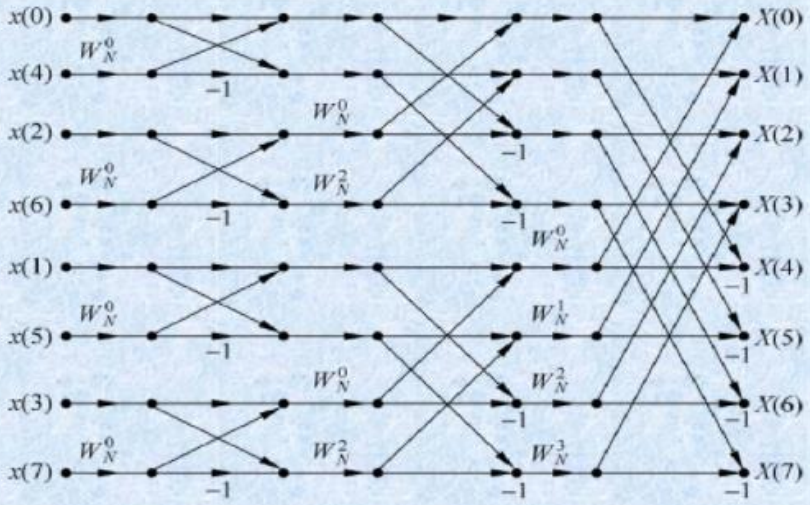
这说明，N=2^M的DFT可全部由蝶形运算来完成。

以8点为例第三次按奇偶分解



N=8按时间抽取法FFT信号流图

5.3.2 按时间抽取基2-FFT算法与直接计算DFT运算量的比较



由按时间抽取法FFT的信号流图可知，当$N=2^L$时，共有 L 级蝶形运算；每级都由 N/2 个蝶形运算组成，而每个蝶形有 1 次复乘、 2 次复加，因此每级运算都需 N/2 次复乘和 N 次复加。

这样 L 级运算总共需要：

复数乘法：$\frac{N}{2}\cdot L=\frac{N}{2}\log_2 N$

复数加法：$N\cdot L=N\log_2 N$

直接DFT算法运算量

复数乘法：N^2

复数加法：$N(N-1)$

直接计算DFT与FFT算法的计算量之比为M

$$M=\frac{N^2}{\frac{N}{2}\log_2 N}=\frac{2N}{\log_2 N}$$

FFT算法与直接DFT算法运算量的比较

N	N^2	$\frac{N}{2}\log_2 N$	计算量之比M	N	N^2	$\frac{N}{2}\log_2 N$	计算量之比M
2	4	1	4.0	128	16 384	448	36.6
4	16	4	4.0	256	65 536	1 024	64.0
8	64	12	5.4	512	262 144	2 304	113.8
16	256	32	8.0	1024	1 048 576	5 120	204.8
32	1028	80	12.8	2048	4 194 304	11 264	372.4
64	4049	192	21.4				

5.3.3 按时间抽取的FFT算法的特点

- 序列的逆序排列
- 同址运算（原位运算）
- 蝶形运算两节点间的距离
- W_N^r 的确定

序列的逆序排列

- 序列的逆序排列

由于 $x(n)$ 被反复地按奇、偶分组，所以流图输入端的排列不再是顺序的，但仍有规律可循：

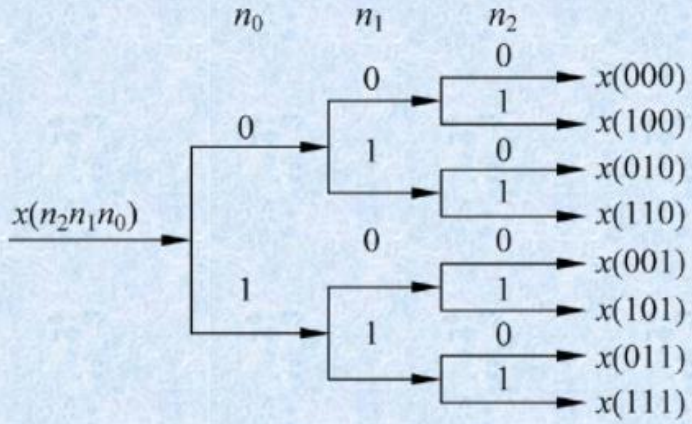
因为 $N=2^M$，对于任意 n（$0 \le n \le N-1$），可以用M个二进制码表示为：

$$n_{(DEC)} = (n_{M-1} n_{M-2} \cdots n_2 n_1 n_0)_{(BIN)}$$

$$n_{M-1}, n_{M-2}, \cdots, n_2, n_1, n_0 = \begin{cases} 0 \\ 1 \end{cases}$$

n 反复按奇、偶分解时，即按二进制码的“0”“1”分解。

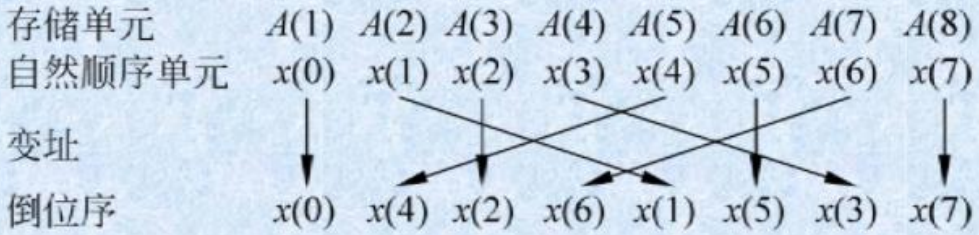
倒位序的树状图（N=8）



码位的倒位序(N=8)

自然顺序 n	二进制数	倒位序二进制数	倒位序顺序数 $\hat{n}$
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

倒位序的变址处理（N=8）

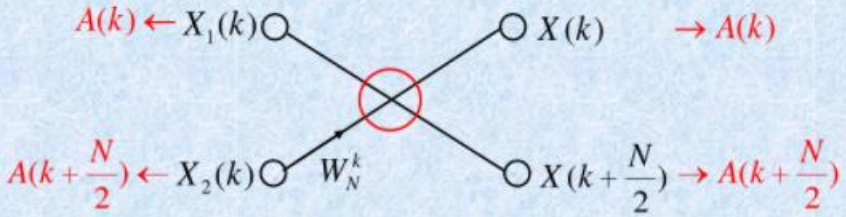


同址运算（原位运算）

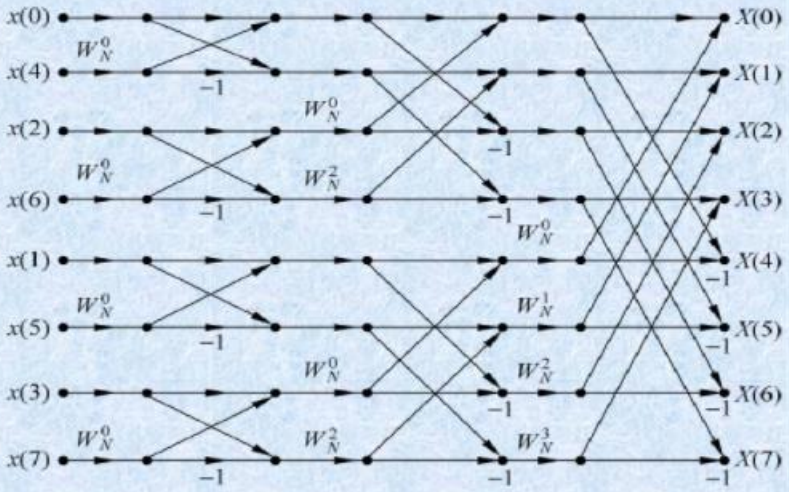
- 同址运算（原位运算）

某一列任何两个节点k 和j 的节点变量进行蝶形运算后，得到结果为下一列k、j两节点的节点变量，而和其他节点变量无关。这种原位运算结构可以节省存储单元，降低设备成本。

例　运算前　　运算后



观察原位运算规律



蝶形运算两节点间的距离

- 蝶形运算两节点间的距离

以N=8为例：

第一级蝶形，距离为： 1

第二级蝶形，距离为： 2

第三级蝶形，距离为： 4

规律：对于共L级的蝶形而言，其m级蝶形运算的节点间的距离为 2^{m-1}

W_N^r 的确定

- W_N^r 的确定

以N=8为例：

$m=1$时，$W_N^r=W_{N/4}^j=W_{2^m}^j=W_2^0, j=0$

$m=2$时，$W_N^r=W_{N/2}^j=W_{2^m}^j=W_4^j, j=0,1$

$m=3$时，$W_N^r=W_N^j=W_{2^m}^j=W_8^j, j=0,1,2,3$

$N=2^M$，第L级：

$$W_N^r=W_{2^L}^j, j=0,1,2,\cdots,2^{L-1}-1$$

$$\because 2^L=2^M\times 2^{L-M}=N\times 2^{L-M}$$

$$\therefore W_N^r=W_{N\cdot 2^{L-M}}^j=e^{-j\frac{2\pi}{N\cdot 2^{L-M}}\cdot j}=e^{-j\frac{2\pi}{N}j\cdot 2^{M-L}}=W_N^{j\cdot 2^{M-L}}$$

5.4 按频率抽取的基2-FFT算法

- **算法原理**

先把输入按n的顺序分成前后两半

再把输出$X(k)$按k的奇偶分组

设序列长度为$N=2^L$，L为整数

前半子序列$x(n)$　　$0 \le n \le \frac{N}{2}-1$

后半子序列$x(n+\frac{N}{2})$　　$0 \le n \le \frac{N}{2}-1$

5.4.1 算法原理

由DFT定义得

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

$$=\sum_{n=0}^{N/2-1}x(n)W_N^{nk}+\sum_{n=N/2}^{N-1}x(n)W_N^{nk}$$

$$=\sum_{n=0}^{N/2-1}x(n)W_N^{nk}+\sum_{n=0}^{N/2-1}x(n+\frac{N}{2})W_N^{(n+\frac{N}{2})k}$$

$$=\sum_{n=0}^{N/2-1}\left[x(n)+x(n+\frac{N}{2})W_N^{\frac{N}{2}k}\right]\cdot W_N^{nk} \qquad k=0,\ 1,\ \ldots,\ N$$

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

由于

$$W_N^{\frac{N}{2}}=e^{-j\frac{2\pi}{N}\cdot\frac{N}{2}}=e^{-j\pi}=-1$$

所以

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

则

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

k=0，1，…，N

然后按k的奇偶可将$X(k)$分为两部分

$$\begin{cases}k=2r\\k=2r+1\end{cases}\qquad r=0,1,\ \ldots,\frac{N}{2}-1$$

则式

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

可转化为

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

令

$$\begin{cases} x_1(n) = x(n) + x(n+\dfrac{N}{2}) \\ x_2(n) = \left[x(n) - x(n+\dfrac{N}{2}) \right] W_N^n \end{cases}$$

n=0，1，…，$\dfrac{N}{2}-1$

代入

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

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

可得

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

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

r=0，1，…，$\dfrac{N}{2}-1$

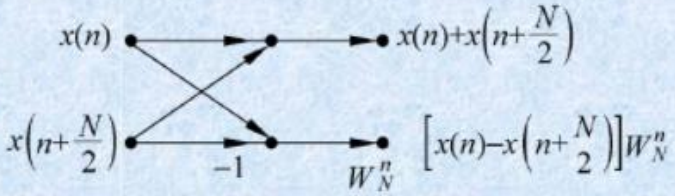
为2个$N/2$点的DFT，合起来正好是N点$X(k)$的值。

蝶形运算

将

$$\begin{cases} x_1(n) = x(n) + x(n+\dfrac{N}{2}) \\ x_2(n) = \left[x(n) - x(n+\dfrac{N}{2}) \right] W_N^n \end{cases}$$

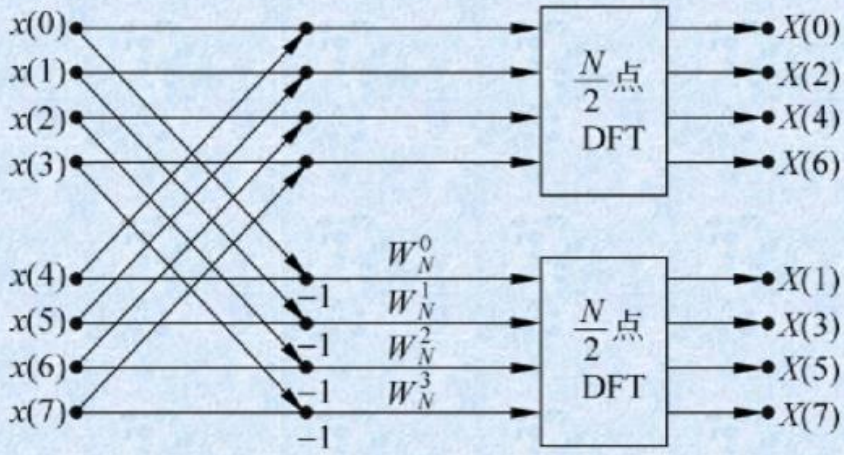
称为蝶形运算



与时间抽选基2FFT算法中的蝶形运算符号略有不同。

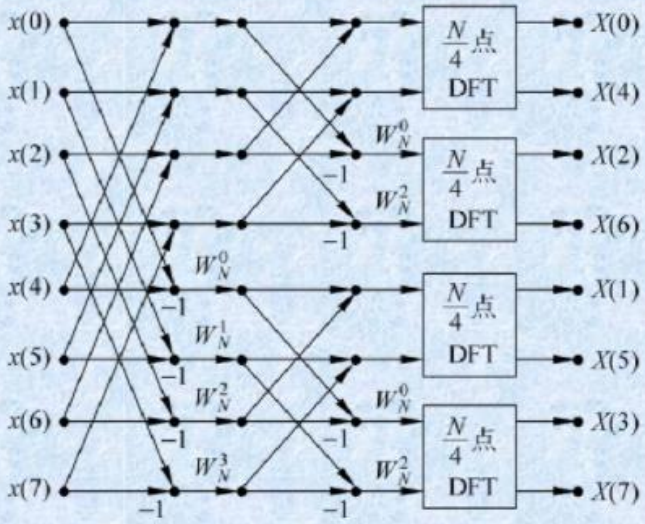
例 按频率抽取(N=8)

例 按频率抽取，将N点DFT分解为两个N/2点DFT的组合(N=8)



与时间抽取法的推导过程一样，由于 $N=2^L$，$N/2$仍然是一个偶数，因而可以将每个$N/2$点DFT的输出再分解为偶数组与奇数组，这就将$N/2$点DFT进一步分解为两个$N/4$点DFT。

N=8



5.4.2 频率抽取法与时间抽取法的异同

- 频率抽取法输入是自然顺序，输出是倒位序的；时间抽取法正好相反。
- 频率抽取法的基本蝶形与时间抽取法的基本蝶形有所不同。
- 频率抽取法运算量与时间抽取法相同。
- 频率抽取法与时间抽取法的基本蝶形是互为转置的。

5.5 快速傅里叶逆变换(IFFT)算法

IDFT公式

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

DFT公式

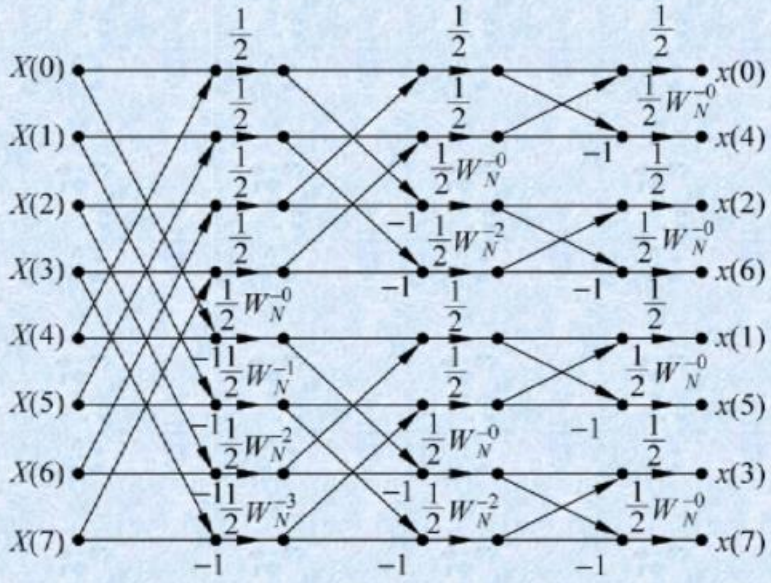
$$X(k)=DFT[x(n)]=\sum_{n=0}^{N-1}x(n)W_N^{nk}$$

比较可以看出，$W_N^{nk} \Longrightarrow W_N^{-nk}$

IDFT多出 $N=2^M \Longrightarrow \frac{1}{N}=\left(\frac{1}{2}\right)^M$

M个1/2可分解到M级蝶形运算中。

例 频率抽取IFFT流图(N=8)



快速傅里叶逆变换另一种算法

$$IDFT[X(k)] = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk}$$

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

$$\therefore \quad IFFT[X(k)] = \frac{1}{N}\left\{FFT[X^*(k)]\right\}^*$$

$$X(k) \xrightarrow{\text{求共轭}} X^*(k) \xrightarrow{\text{求}FFT} FFT[X^*(k)]$$

$$\xrightarrow{\text{求共轭}} \left\{FFT[X^*(k)]\right\}^* \xrightarrow{\text{除以}N} x(n)$$

5.8 Matlab实现

- 用FFT进行谱分析的Matlab实现
- 用CZT进行谱分析的Matlab实现
- 在Matlab中使用的线性调频z变换函数为czt，其调用格式为
  - >>X= czt(x, M, W, A)
  - 其中，x是待变换的时域信号x(n)，其长度为N，M是变换的长度，W确定变换的步长，A确定变换的起点。若M= N，A= 1，则CZT变成DFT。

5.8.1 用FFT进行谱分析的Matlab实现

例5.1 设模拟信号 $x(t)=2\sin(4\pi t)+5\cos(8\pi t)$，以 $t=0.01n$ (n=0: N-1) 进行取样，试用fft函数对其做频谱分析。N分别为：(1) N=45；(2) N=50；(3) N=55；(2) N=60。

程序清单如下

```
%计算N=45的FFT并绘出其幅频曲线
N=45;n=0:N-1;t=0.01*n;
q=n*2*pi/N;
x=2*sin(4*pi*t)+5*cos(8*pi*t);
y=fft(x,N);
figure(1)
subplot(2,2,1)
plot(q,abs(y))
title('FFT N=45')
```

例5.1程序清单

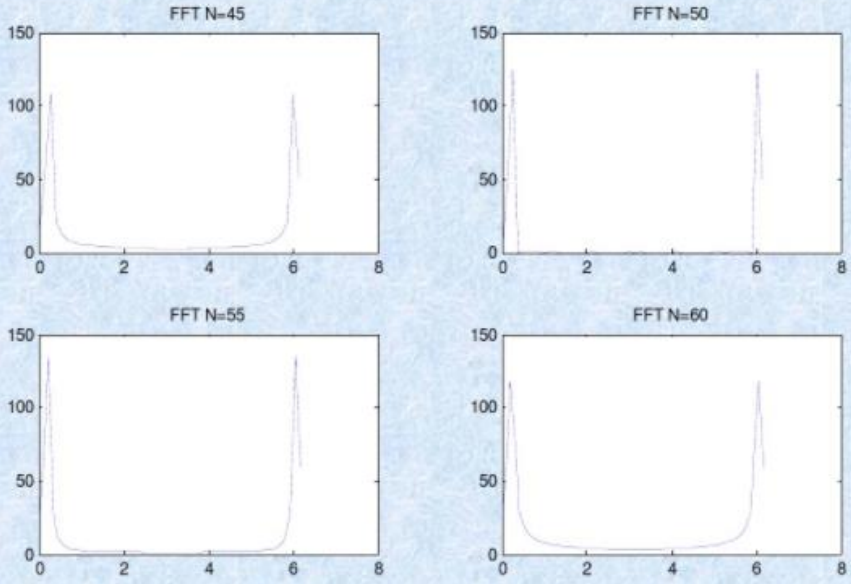
```
%计算N=50的FFT并绘出其幅频曲线
N=50;n=0:N-1;t=0.01*n;
q=n*2*pi/N;
x=2*sin(4*pi*t)+5*cos(8*pi*t);
y=fft(x,N);
figure(1)
subplot(2,2,2)
plot(q,abs(y))
title('FFT N=50')
```



```
%计算N=55的FFT并绘出其幅频曲线
N=55;n=0:N-1;t=0.01*n;
q=n*2*pi/N;
x=2*sin(4*pi*t)+5*cos(8*pi*t);
y=fft(x,N);
figure(1)
subplot(2,2,3)
plot(q,abs(y))
title('FFT N=55')
```

```
%计算N=60的FFT并绘出其幅频曲线
N=60;n=0:N-1;t=0.01*n;
q=n*2*pi/N;
x=2*sin(4*pi*t)+5*cos(8*pi*t);
y=fft(x,N);
figure(1)
subplot(2,2,4)
plot(q,abs(y))
title('FFT N=60')
```


例5.1程序运行结果



从图中可以看出，这几种情况下均有较好的精度。

s

例5.1程序运行结果分析

分析：由*t*=0.01*n*进行取样可得，采样频率fs=100Hz。而连续信号的最高模拟角频率为Ω＝8 π ，由Ω＝2 πf可得，最高频率为8 π /2 π=4Hz。因此，满足采样定理的要求。

采样序列为

$$x(n)=2\cos(4\pi Tn)+5\cos(8\pi Tn)$$

即

$$x(n)=2\cos\left(\frac{4\pi}{100}n\right)+5\cos\left(\frac{8\pi}{100}n\right)$$

为周期序列，周期*N*=50。

将程序中plot改为stem函数，则可以更清楚地看出频谱。

例5.1修改程序运行结果