



---

---

---

---

---

---

---

---

通过第二章的学习，我们知道有限长序列的一个重要特点是其频域也可以离散化，即可以进行DFT变换。DFT变换在数字信号处理中占有很重要的地位，例如FIR滤波器的设计，以及信号的频谱分析都离不开DFT运算。

尽管如此，但在很长一段时间里，由于DFT运算的冗长和繁杂，DFT并未得到真正的运用。直到1965年，快速傅里叶变换（FFT）算法的出现使得DFT的运算大大简化，DFT才真正在实际中得到广泛应用。

本章主要内容：

3.1 DFT运算的特点	3.2 按时间抽取
3.3 按频率抽取	3.4 IDFT的运算方法
3.5 任意基数的算法	3.6 Chirp-z变换
3.7 线性卷积的FFT算法	3.8 FFT的流水线工作原理

本章主要内容

9/21/2022

2

---

---

---

---

---

---

---

---

3.1 DFT运算的特点

我们先来计算一下，对有限长序列 $x(n)$ 进行一次DFT运算的工作量：

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

一般， $x(n)$ 和 $W_N^{nk}$ 都是复数。因此，每计算一个 $X(k)$ 值，必须进行 $N$ 次复数相乘和 $N-1$ 次复数相加。 $X(k)$ 共有 $N$ 个点，因此要完成全部DFT运算需要进行 $N^2$ 次复数相乘和 $N(N-1)$ 次复数相加。每一个复数相乘包括4个实数相乘和2个实数相加。

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

这样，每运算一个 $X(k)$ 值需要进行 $4N$ 次实数相乘和 $2N+2(N-1)=2(2N-1)$ 次实数相加。因此整个DFT运算需要 $4N^2$ 次实数相乘和 $2N(2N-1)$ 次实数相加。可见，DFT的运算量与 $N^2$ 成正比。

3.1 DFT运算的特点

9/21/2022

3

---

---

---

---


---

---

---

---

1

 西安交通大学

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

显然，IDFT与DFT具有相同的运算量。

考察上面的DFT和IDFT运算，可以发现：

系数  $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$  是一个周期函数，利用它的周期性和对称性可使运算得以改善。例如  $W_N^{n(N-k)} = W_N^{k(N-n)} = W_N^{-nk}$ ，又如  $W_N^{N/2} = -1$ ，因此  $W_N^{(k+N/2)} = -W_N^k$ 。

- ① 利用这些周期性和对称性，使DFT运算中有些项可以合并。
- ② 利用  $W_N^{nk}$  的周期性和对称性，使DFT运算尽量分解为更少点数的DFT运算。

快速傅里叶变换算法正是基于这样的基本思想发展起来的。

**快速傅里叶变换算法可分为两大类：时间抽取法和频率抽取法**

3.1 DFT运算的特点 9/21/2022 4

---

---

---

---

---


---

---

---

---

---

 西安交通大学

### 3.2 按时间抽取

为了将一个N点的DFT运算分解，假定  $N = 2^M$ ，其中M为正整数。

首先，将序列  $x(n)$  分解为两组，偶数项为一组，奇数项为一组。

$$\begin{cases} x(2r) = x_1(r) \\ x(2r+1) = x_2(r) \end{cases} \quad r = 0, 1, \dots, N/2-1$$

将DFT运算也相应分为两组：

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

由于  $W_N^{2n} = e^{-j\frac{2\pi}{N}2n} = e^{-j\frac{2\pi}{N/2}n} = W_{N/2}^n$  所以

3.2 按时间抽取 9/21/2022 5

---

---

---

---

---


---

---

---

---

---

 西安交通大学

$$\begin{aligned} X(k) &= \sum_{r=0}^{N/2-1} x_1(r) W_N^{2rk} + W_N^k \sum_{r=0}^{N/2-1} x_2(r) W_N^{2rk} \\ &= \sum_{r=0}^{N/2-1} x_1(r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x_2(r) W_{N/2}^{rk} \\ &= X_1(k) + W_N^k X_2(k), \quad k = 0, 1, \dots, N-1 \quad (3-2) \end{aligned}$$

于是，一个N点的DFT被分解为两个N/2点的DFT了，这两个N/2点的DFT再按照式（3-2）合并成一个N点DFT。

$X_1(k)$ 、 $X_2(k)$  只有N/2个点，而  $X(k)$  却有N个点，要用  $X_1(k)$ 、 $X_2(k)$  表达全部  $X(k)$  值，还必须利用W系数的周期特性。

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

3.2 按时间抽取 9/21/2022 6

---

---

---

---

---

---

---

---

---

---

即  $X_1(N/2+k)=X_1(k)$  同样  $X_2(N/2+k)=X_2(k)$

又考虑到  $W_N^{N/2+k}=W_N^{N/2} \cdot W_N^k = -W_N^k$

将上述三个式子代入式(3-2), 就可将X(k)表达为前后两部分:

$$X(k)=X_1(k)+W_N^k X_2(k), \quad k=0,1,\cdots,N/2-1 \quad (3-3)$$

$$X(N/2+k)=X_1(N/2+k)+W_N^{N/2+k} X_2(N/2+k) \\ =X_1(k)-W_N^k X_2(k), \quad k=0,1,\cdots,N/2-1 \quad (3-4)$$

可以用下图的“蝶形结”来表示

蝶形运算流程图符号

3.2 按时间抽取

9/21/2022

7

---

---

---

---

---

---

---

---

通过上述分解后, 每个N/2点DFT只需要 $(N/2)^2=N^2/4$ 次复数相乘。两个N/2点的DFT需要 $2(N/2)^2=N^2/2$ 次复数相乘, 再加上将两个N/2点的DFT合成N点DFT时, 蝶形结前的N/2次复乘, 一共需要 $N^2/2+N/2=N(N+1)/2 \approx N^2/2$ 次复乘。可见, 分解后运算量大约节省了一倍。

既然这样的分解是有效的, 由于 $N=2^M$ , N/2仍然是偶数, 因此可以对两个N/2点的DFT再分别作进一步分解。如右图所示:

3.2 按时间抽取

9/21/2022

8

---

---

---

---

---

---

---

---

一个8点的DFT的完整的按时间抽取运算的流程图如下图所示:

由于每一步分解都是按输入序列在时域上的次序是属于偶数还是奇数来抽取的, 所以称为“按时间抽取法”或“时间抽取法”。

对于任何一个2的整数幂 $N=2^M$ 总是可以通过M次的分解最后完全成为2点的DFT运算的。这样的M次分解就构成从x(n)到X(k)的M级运算过程。

而每级运算都由N/2个蝶形运算构成。每一级蝶形运算都需要N/2次复乘和N次复加, 这样M级总运算量为:

$$\text{复乘数: } m_r = \frac{N}{2} \cdot M = \frac{N}{2} \log_2 N \quad \text{复加数: } a_r = N \cdot M = N \log_2 N$$

结论: 用时间抽取法所需的运算量与 $N \log_2 N$ 成正比。

3.2 按时间抽取

9/21/2022

9

---

---

---

---

---

---

---

---

3

3.1 节中，我们已经知道直接DFT运算的运算量与  $N^2$  成正比。

**FFT算法与直接算法的运算量比较**

M	N	$N^2$	$N \log_2 N$	$N^2 / (N \log_2 N)$
1	2	4	2	2.0
2	4	16	8	2.0
3	8	64	24	2.7
4	16	256	64	4.0
5	32	1024	160	6.4
6	64	4096	384	10.7
7	128	16384	896	18.3
8	256	65536	2048	32.0
9	512	262144	4608	56.9
10	1024	1048576	10240	102.4
11	2048	4194304	22528	186.2
12	4096	16777216	49152	341.3

**原位运算和变址**

这种时间抽取法的特点：


- 每一级运算都有 $N/2$ 个蝶形运算，如图的蝶形运算结构图所示。它由一次加权运算和一次加减运算构成；
- 原位运算**，整个变换运算可以分为 $M$ 级，每一级运算均可在原位进行，这种原位运算的结构可以节省存储单元。但这种原位运算的输入却不能按自然顺序存放。

**蝶形运算流程图**

西安交通大学  
XIAN JIAO TONG DA XUE

变址：输入序列的排序是原序列二进制的码位倒置。

自然顺序	二进制	码位倒置	倒置顺序	自然顺序	二进制	码位倒置	倒置顺序
0	000	000	0	4	100	001	1
1	001	100	4	5	101	101	5
2	010	010	2	6	110	011	3
3	011	110	6	7	111	111	7


  
 西安交通大学

```

void inverse(double a[], int len, int number_of_bits){
    /* a: 输入的数据数组; len: a的长度
    number_of_bits: 进行两位倒置的位数. len = 2^number_of_bits;
    */
    unsigned in, out, /*一进一出两个缓存*/last_bit, i, j;
    for(i = 0; i < len; i++){
        out = i;
        in = 0;
        for(j = 1; j <= number_of_bits; j++){
            last_bit = out & 1; /*取出出缓冲的最后一位*/
            out >>= 1; /*出缓冲右移一位*/
            in = in << 1 + last_bit; /*入缓冲左移一位, 并移入出缓冲的最后一位*/
        }
        if(i < in){
            double temp;
            temp = a[i];
            a[i] = a[in];
            a[in] = temp; /*交换a[i]和a[in]的值*/
        }
    }
    return;
}
  
```

3.2 按时间抽取
9/21/2022
13

---

---

---

---

---


---

---

---

---

---


  
 西安交通大学

### 3.3 按频率抽取

仍然假定  $N = 2^M$ 。频率抽取法是将输入序列接前后对半分，这样可将N点DFT写成前后两部分。

$$\begin{aligned}
 X(k) &= \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2}^{N-1} x(n)W_N^{nk} \\
 &= \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=0}^{N/2-1} x(n+N/2)W_N^{(n+N/2)k} \\
 &= \sum_{n=0}^{N/2-1} [x(n) + W_N^{(N/2)k} x(n+N/2)]W_N^{nk}
 \end{aligned}$$

由于  $W_N^{N/2} = -1$ ,  $W_N^{(N/2)k} = (-1)^k$ ,  $k$  为偶数时  $(-1)^k = 1$ ,  $k$  为奇数时  $(-1)^k = -1$

因此,  $X(k)$  可进一步分解为偶数组和奇数组。

3.3 按频率抽取
9/21/2022
14

---

---

---

---

---


---

---

---

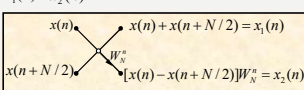
---

---


  
 西安交通大学

$$\begin{aligned}
 X(k) &= \sum_{n=0}^{N/2-1} [x(n) + (-1)^k x(n+N/2)]W_N^{nk} \\
 X(2r) &= \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)]W_N^{2nr} \\
 &= \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)]W_{N/2}^{nr} = \sum_{n=0}^{N/2-1} x_1(n)W_{N/2}^{nr} \quad (3-5) \\
 X(2r+1) &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_N^{(2r+1)n} \\
 &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_{N/2}^{nr} = \sum_{n=0}^{N/2-1} x_2(n)W_{N/2}^{nr} \quad (3-6)
 \end{aligned}$$

式(3-5)、(3-6)表示的正  $x_1(n)$ 、 $x_2(n)$  可用下图所示的蝶形运算来表示:



频率抽取法的蝶形运算

是两个N/2点的DFT运算, 这样同样将一个N点的DFT分解为两个N/2点的DFT了。

3.3 按频率抽取
9/21/2022
15

---

---

---

---

---

---

---

---

---

---

与时间抽取法的推演过程一样，由于  $N=2^M$ ， $N/2$  仍然是偶数，因此可以将  $N/2$  点的 DFT 的输出再分解为偶数组与奇数组，这样就将  $N/2$  点 DFT 进一步分解为两个  $N/4$  点的 DFT 了。

按频率抽取将  $N$  点 DFT 分解为 2 个  $N/2$  点 DFT      按频率抽取将  $N$  点 DFT 分解为 4 个  $N/4$  点 DFT

3.3 按频率抽取      9/21/2022      16

---

---

---

---

---

---

---

---

---

---

一个 8 点的 DFT 的完整的按频率抽取的 FFT 结构如下图所示：

**N=8 的频率抽取法 FFT 流程图**

由于每一步分解都是按输出  $X(k)$  在频域的顺序上是属于偶数还是奇数分组的，所以称为“频率抽取法”。

对于任何一个 2 的整数次幂  $N=2^M$ ，总是可以通过  $M$  次的分解最后完全成为 2 点的 DFT 运算的。

与时域抽取法相同，频率抽取法也共有  $M$  级运算。每级运算需要  $N/2$  个蝶形运算，因此其运算量与时域抽取法相同。

**频域抽取法与时域抽取法是两种等价的 FFT 运算**

3.3 按频率抽取      9/21/2022      17

---

---

---

---

---

---

---

---

---

---

### 3.4 IDFT 的运算方法

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

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

FFT 算法同样可以用于 IDFT 运算，简称为 IFFT 即快速傅里叶反变换。只要把 DFT 运算中的每一个系数  $W_N^{nk}$  改为  $W_N^{-nk}$  并且最后再乘以常数  $1/N$ ，那么前面所讨论的时间抽取和频率抽取的 FFT 算法都可以直接拿来运算 IDFT，只是在命名上要颠倒一下，如时间抽取的 FFT 运算用于 IDFT 时，应称为频率抽取 IFFT。

另外，在 IFFT 运算中，经常将常数  $1/N$  分解为  $(1/2)^M$ ，并且在  $M$  级运算中，每级运算都分别乘一个  $1/2$  因子，这样就得到 IFFT 的两种基本蝶形运算结构。

3.4 IDFT 的运算方法      9/21/2022      18

---

---

---

---

---

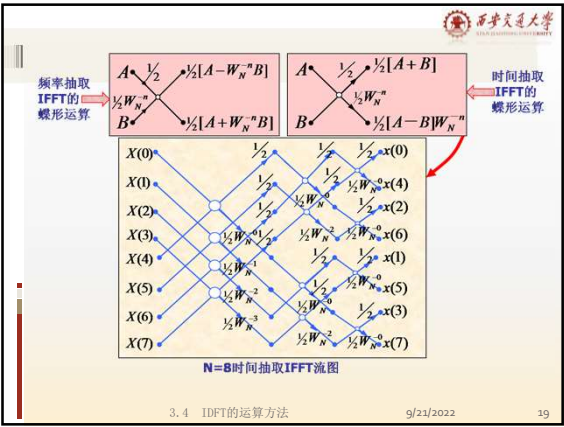
---

---

---

---

---



---

---

---

---

---

---

---

---

另外还有一种IFFT算法可以完全不用改动FFT的程序。

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

这就是说，如果我们先将 $X(k)$ 取共轭变换，即将 $X(k)$ 的虚部乘以-1，然后就可以直接访问FFT的子程序，最后再对运算结果取一次共轭变换并乘以常数 $1/N$ 即可得到 $x(n)$ 值。这样，FFT和IFFT可以共用一个子程序块，这在使用通用计算机时是比较方便的。

3.4 IDFT的运算方法

---

---

---

---

---

---

---

---

### 3.5 任意基数的算法

前面讨论的都是以2为基数的FFT算法，即  $N = 2^M$  的情况，本节将讨论 $N$ 的数值不是2的整数次幂时的处理办法。

$N \neq 2^M$  时，一般有两种处理办法：

- ◆ 将 $x(n)$ 用补零的办法延长，以使 $N$ 增长到最近的一个 $2^M$  数值。有限长序列补零以后并不影响其频谱 $X(e^{j\omega})$ ，只是频谱的采样点数增加了。
- ◆ 如果要求准确的 $N$ 点DFT值，则可以用以任意数为基数的FFT算法来计算。

下面我们来讨论一下以任意数为基数的FFT算法的基本原则。

同以2为基数时一样，快速傅里叶变换的基本思想就是要将DFT的运算量尽量减小。因此，如果 $N$ 可以分解为两个整数 $p$ 与 $q$ 的乘积，即  $N = p \cdot q$ ，也希望将 $N$ 点的DFT分解为 $p$ 个 $q$ 点DFT或 $q$ 个 $p$ 点的DFT，这样就可以减小运算量。

3.5 任意基数的算法

---

---

---

---

---

---

---

---

为此, 我们先将 $x(n)$ 分成 $p$ 组, 即

$$p \text{ 组 } \begin{cases} x(pr) \\ x(pr+1) \\ x(pr+2) \\ \vdots \\ x(pr+p-1) \end{cases} \quad r = 0, 1, \dots, q-1$$

这 $p$ 组序列每一组都是一个长度为 $q$ 的有限长序列。

然后将 $N$ 点DFT也相应分解为 $p$ 组:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{nk} = \sum_{r=0}^{q-1} \sum_{l=0}^{p-1} x(pr+l)W_N^{(pr+l)k} + \dots \\ &+ \sum_{r=0}^{q-1} x(pr+p-1)W_N^{(pr+p-1)k} = \sum_{r=0}^{q-1} x(pr)W_N^{prk} \\ &+ W_N^{pk} \sum_{r=0}^{q-1} x(pr+1)W_N^{prk} + W_N^{2k} \sum_{r=0}^{q-1} x(pr+2)W_N^{prk} + \dots \\ &+ W_N^{(p-1)k} \sum_{r=0}^{q-1} x(pr+p-1)W_N^{prk} = \sum_{l=0}^{p-1} W_N^{lk} \sum_{r=0}^{q-1} x(pr+l)W_N^{prk} \end{aligned}$$

3.5 任意基数的算法 9/21/2022 22

由于 $W_N^{pk} = W_q^{rk} = W_q^{rk}$ , 因此上式中第二个 $\sum$ 完全代表一个 $q$ 点的DFT, 即

$$X(k) = \sum_{l=0}^{p-1} W_N^{lk} Q_l(k)$$

其中 $Q_l(k)$ 就是第 $l$ 组序列的 $q$ 点DFT:

$$Q_l(k) = DFT[x(pr+l)] = \sum_{r=0}^{q-1} x(pr+l)W_q^{rk} \quad (3-7)$$

这样, 式(3-7)就表明了一个 $N=p \cdot q$ 点的DFT可以用 $p$ 组的 $q$ 点DFT来组成, 这个关系如右图所示:

3.5 任意基数的算法 9/21/2022 23

例3.1  $p=3, q=2, N=6$ 的DFT运算流程图如下所示:

3.5 任意基数的算法 9/21/2022 24



实际上，很少有像例3.1那样简单的实例，但是这种分解的原则对于任意基数的任何更加复杂的情况都是适用的。

例如当N如果可以分解为m个质数因子  $P_1, P_2, \dots, P_m$  时，即

$$N = p_1 p_2 \dots p_m$$

那么我们第一步可以把N先分解为两个因子  $N = p_1 q_1$ ，其中  $q_1 = p_2 p_3 \dots p_m$ ，并用以上所讨论的方法将DFT分解为  $p_1$  个  $q_1$  点DFT，然后第二步再将  $q_1$  分解为  $q_1 = p_2 q_2$ ，其中  $q_2 = p_3 p_4 \dots p_m$ 。

将每一个  $q_1$  点DFT再分解为  $p_2$  个  $q_2$  点DFT，这样可以通过m次分解一直分到最少点数的DFT运算，从而使运算获得最高的效率。

3.5 任意基数的算法 9/21/2022 25

---

---

---

---

---

---

---

---

---

---

### 3.7 线性卷积的FFT算法

FIR滤波器除了可以通过数字网络来实现外，也可以通过FFT变换来实现。

(a) 数字网络实现方法

(b) FFT实现方法

滤波器的两种实现方法

下面我们来分析一下第二种方法是怎样工作的，并比较一下两种方法的特点。

(a) 数字实现的滤波器

数字网络类FIR滤波器中，普遍使用的横截型结构如右图所示：

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

3.7 线性卷积的FFT算法 9/21/2022 26

---

---

---

---

---

---

---

---

---

---

设  $h(n)$  的点长度为  $N_1$ ，而信号  $x(n)$  的点长度为  $N_2$ ，即

$$h(n) = \begin{cases} h(n), & 0 \leq n \leq N_1 - 1 \\ 0, & n < 0, n > N_1 - 1 \end{cases} \quad x(n) = \begin{cases} x(n), & 0 \leq n \leq N_2 - 1 \\ 0, & n < 0, n > N_2 - 1 \end{cases}$$

卷积结果  $y(n)$  也是一个有限长序列，点长度为  $N = N_1 + N_2 - 1$ ，即

$$y(n) = \begin{cases} y(n), & 0 \leq n \leq N - 1 \\ 0, & n < 0, n > N - 1 \end{cases}$$

根据上页FIR滤波器的横截型结构，可计算出线性卷积公式的运算量：

乘法： $m_d = N_1 N_2$     加法： $a_d = (N_1 N_2 - 1) - (N - 1) = (N_1 - 1)(N_2 - 1)$

而对于线性相位滤波器，由于其  $h(n)$  具有偶对称或奇对称特性，即

$$h(n) = \pm h(N_1 - 1 - n)$$

这里加权系数减少了一半，所以线性相位滤波器，相乘次数  $m_d = N_1 N_2 / 2$ 。

线性相位FIR滤波器横截型结构( $N_1$ 为偶数)

3.7 线性卷积的FFT算法 9/21/2022 27

---

---

---

---

---


---

---

---

---

---


 西安交通大学  
XIDIAN UNIVERSITY

## (b) FFT实现的滤波器

第二章已经介绍过，两个有限长序列的卷积可以用它们的圆周卷积来代替。圆周卷积结果不产生混淆的条件是使  $x(n)$ ,  $h(n)$  都至少补零加长到  $N$  点，即  $N = N_1 + N_2 - 1$

$$x(n) = \begin{cases} x(n), & 0 \leq n \leq N_2 - 1 \\ 0, & N_2 \leq n \leq N - 1 \end{cases} \quad h(n) = \begin{cases} h(n), & 0 \leq n \leq N_1 - 1 \\ 0, & N_1 \leq n \leq N - 1 \end{cases}$$

这样  $y(n)$  的值就可以通过以下四步来实现：

- (1) 求  $H(k) = DFT[h(n)]$ ,  $N$  点；
- (2) 求  $X(k) = DFT[x(n)]$ ,  $N$  点；
- (3) 运算  $Y(k) = H(k)X(k)$ ；
- (4) 反变换  $y(n) = IDFT[Y(k)]$ 。

从这四步可以看出，**这样处理的大部分工作量都可以用FFT完成。**


假定  $N$  是以2为基数的整数幂，从以上四步看需要三次FFT运算。但对FIR滤波器来说，第一步实际上是不需要的，因此实际需要两个FFT运算共需  $2 \cdot N \cdot 2(\log_2 N)$  次乘法和  $2 \cdot N(\log_2 N)$  次加，另外第(3)步还需要  $N$  次相乘，因此总的运算量为：

$$m_p = N \log_2 N + N = N(1 + \log_2 N) \quad a_p = 2 \cdot N \log_2 N$$

3.7 线性卷积的FFT算法

9/23/2022

28


西安交通大学  
XI'AN JIAOTONG UNIVERSITY

下面，我们就以线性相位FIR滤波器的相乘运算来比较两种方法的快慢。将两种方法相乘运算的工作量相比较，定义

$$c_m = m_d / m_F = N_1 N_2 / [2N(1 + \log_2 N)]$$

$$= N_1 N_2 / \{2(N_1 + N_2 - 1)[1 + \log_2(N_1 + N_2 - 1)]\}$$

- 当 $x(n)$ 与 $h(n)$ 长度差不多时，例如  $N_1 = N_2 = N$   $2N - 1 \approx 2N$   
 此时  $c_m = N_1 / [4(1 + \log_2 2N)] = N_1 / [4(2 + \log_2 N)]$

$N_1$	8	32	512	4096
$c_m$	1/2.5	1.1	11.6	73

由上表可知，当长度  $N_1$  超过32以后， $N_1$  越长，圆周卷积的优越性越大，因此常将圆周卷积称为快速卷积。

- 当信号 $x(n)$ 很长时，即  $N_2 \gg N_1$ ， $N \approx N_2$ ，此时  $c_m = N_1 / [2(1 + \log_2 N_2)]$

显然  $N_2$  大长时，会使  $c_m$  下降，从而使圆周卷积的优点不能发挥出来。

3.7 线性卷积的FFT算法
9/23/2022

29


**西安交通大学**  
 XIDIAN UNIVERSITY

克服这个困难的办法是采用分段卷积或称为分段过滤的办法。

当 $x(n)$ 是一个长序列时，用圆周卷积所以不利是因为 $h(n)$ 必须补很多的零头，以至于 $N$ 点中大部分是零，因此很不经济，要改进它就必须将 $x(n)$ 分为一段一段和 $h(n)$ 长度相仿的片段。

下图就是将 $x(n)$ 分成为一段段长度为 $N_2$ 的小片段，然后对每一段 $x(n)$ 再分别用FFT来处理。处理办法一般有两种：重叠相加法和重叠保存法。



西安交通大学  
XI'AN JIAOTONG UNIVERSITY

### 一、重叠相加法

假定以  $x_i(n)$  表示上页图中第  $i$  段  $x(n)$  序列:

$$x_i(n) = \begin{cases} x(n), & iN_2 \leq n \leq (i+1)N_2 - 1 \\ 0, & \text{其它 } n \end{cases}$$

则输入序列  $x(n) = \sum_{i=-\infty}^{\infty} x_i(n)$

这样输出序列可分解为

$$y(n) = x(n) * h(n) = \sum_{i=-\infty}^{\infty} x_i(n) * h(n) = \sum_{i=-\infty}^{\infty} y_i(n) \quad (3-14)$$

其中  $y_i(n) = x_i(n) * h(n) \quad (3-15)$

式(3-14)、(3-15)表明, 只要将  $x(n)$  的每一段分别与  $h(n)$  卷积, 然后再将这些卷积结果相加就可得到输出序列。这样, 每一片段的卷积都可由快速卷积方法来计算。

3.7 线性卷积的FFT算法      9/21/2022      31

---

---

---

---

---

---

---

---

---

---

西安交通大学  
XI'AN JIAOTONG UNIVERSITY

先对  $h(n)$  及  $x_i(n)$  补零, 补到具有  $N$  点长度为  $N = N_1 + N_2 - 1$

一般总是将  $N$  选为  $N = 2^M$

然后用以 2 为基数的 FFT 运算来求  $y_i(n)$

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

由于  $y_i(n)$  的点长度为  $N$ , 而  $x_i(n)$  点长度为  $N_2$ , 因此相邻两段  $y_i(n)$  必有  $N - N_2 = N_1 - 1$  的部分重叠, 如右图所示。

图中所示:  $N_1=8$ ,  $N_2=15$ ,  $N=N_1+N_2-1=22$ 。  
而  $N_2-1=14$ ,  $N-1=21$ ,  $x_i(n)$  后补零的点数  $N_1-1=7$ 。

3.7 线性卷积的FFT算法      9/21/2022      32

---

---

---

---

---

---

---

---

---

---

西安交通大学  
XI'AN JIAOTONG UNIVERSITY

### 重叠相加法的整个运算分五步完成:

- (1) 事先准备好滤波器的参数  $H(k) = DFT[h(n)]$ ,  $N$  点
- (2) 用  $N$  点 FFT 运算  $X_i(k) = DFT[x_i(n)]$
- (3)  $Y_i(k) = X_i(k)H(k)$
- (4) 用  $N$  点 IFFT 运算  $y_i(n) = IDFT[Y_i(k)]$
- (5) 将重叠部分相加起来  $y(n) = \sum_{i=-\infty}^{\infty} y_i(n)$

### 二、重叠保存法

如果将上面的分段序列中补零的部分不是补零而是保存原来的输入序列值, 那么显然圆周卷积以后将出现混淆现象。这个混淆部分只发生在  $y_i(n)$  的起始一段:

$$y(n) = x_i(n) \otimes h(n) = \sum_{m=0}^{N-1} x_i(m)h((n-m))_N R_N(n)$$

3.7 线性卷积的FFT算法      9/21/2022      33

---

---

---

---

---

---

---

---

---

---

由于  $h(n)$  的点长度为  $N_1$ , 因此当  $0 \leq n \leq N_1 - 2$  时,  $h((n-m))_N$  将在  $x_1(m)$  的尾部出现非零值, 如图(b)显示了  $n=1$  时的情况就是这样。因而在  $0 \leq n \leq N_1 - 2$  这一部分的  $y_1(n)$  值将混入  $x_1(m)$  尾部与  $h((n-m))_N$  的卷积值, 显然这一部分的  $y_1(n)$  将不同于直接卷积值, 但是当  $N_1 - 1 \leq n \leq N - 1$  时,  $h((n-m))_N = h(n-m)$  如图(c)及图(d)所示, 所以圆周卷积与直接卷积完全相同, 没有混淆存在。

补零部分保存信号序列后的局部混淆现象

这样, 我们就知道在  $y_1(n)$  的后面  $N_2$  个点是正确的卷积值, 因此可以将每一段运算结果的前  $N_1 - 1$  点去掉。

3.7 线性卷积的FFT算法 9/21/2022 34

先将  $x(n)$  分解为  $x_1(n) = \begin{cases} x(n+iN_2-N_1+1), & 0 \leq n \leq N-1 \\ 0, & \text{其它}n \end{cases}$

然后利用FFT算法求出  $y_1(n) = x_1(n) \otimes h(n)$

再抛弃  $y_1(n)$  的前  $N_1 - 1$  点, 最后将各  $y_1(n)$  顺次连接起来就得到输出序列  $y(n)$ 。

重叠保存法和重叠相加法工作量差不多, 但可以省去重叠相加法的最后一道相加运算。一般用FFT过滤波信号只用于FIR滤波器阶数大于32的情况, 且  $N_2$  一般选在  $N_1$  的5到10倍左右, 即  $N_2 \approx (5 \sim 10)N_1$ , 这样可接近最高效的运算。

重叠保存处理过程示意图


3.7 线性卷积的FFT算法 9/21/2022 35

### 3.8 FFT的流水线工作原理

在许多信号实时处理中, 要求信号输入的同时, 及时地、逐段地完成对信号的FFT运算。完成一个  $N$  点FFT运算需  $m_T = \frac{N}{2} (\log_2 N)$  次复乘, 因此实时工作时, 计算机的速度至少必须高于每采样间隔执行  $m_T / N = \frac{1}{2} (\log_2 N)$  次复乘。当信号速率很高时, 数字硬件往往难以达到那样高的速度。解决办法之一就是设计专用FFT处理器时采用流水线工作方式。


FFT算法的一个重要特点是具有分级运算结构, 因此, 为提高运算速度可以在每级运算中采用单独的运算器, 第一级运算器算完后送给第二级, 自己再算新来的数据, 第二级算完后送第三级……, 如此形成流水线的工作方式。

3.8 FFT的流水线工作原理 9/21/2022 36


西安交通大学

## 时域抽取法

一个8点的DFT的完整的按时间抽取运算的流程图如下图所示：



由于每一步分解都是按输入序列在时域上的次序是偶数还是奇数来抽取的，所以称为“按时间抽取法”或“时间抽取法”。

对于任何一个2的整数幂  $N = 2^M$ ，总是可以通过M次的分解最后完全成为2点的DFT运算的。这样的M次分解就构成从  $X(n)$  到  $X(k)$  的M级运算过程。

而每级运算都由  $N/2$  个蝶形运算构成。每一级蝶形运算都需要  $N/2$  次复乘和  $N$  次复加，这样M级总运算量为：

$$\text{复乘数: } m_F = \frac{N}{2} \cdot M = \frac{N}{2} \log_2 N$$

$$\text{复加数: } a_F = N \cdot M = N \log_2 N$$

**结论：用时间抽取法所需的运算量与  $N \log_2 N$  成正比。**


 清华大学  
TSINGHUA UNIVERSITY

下面以N=8的FFT算法为例介绍流水线工作原理。

N=8时FFT流水线工作示意图

按照上图，完成N=8点FFT运算需要**三级流水线**，信号输入时，开始必须将前四个数据寄存在第一级的存储器中，因为第一级运算器只有在 $x(4)$ 到来之后才能开始运算第一个蝶形结构 $x(0)$ 、 $x(4)$ 组，算完后将数据送入第二级的存储器，如果计算机的速度能满足每采样周期完成一个蝶形结构运算（即1次乘法/采样周期），则当 $x(5)$ 到来时，就可以及时地计算 $x(5)$ 对应的蝶形结构 $x(1)$ 、 $x(5)$ 组……这样，在第二组8点到来以前，第一级运算器正好可以完成它对第一组信号的全部运算，第二组信号再像第一组信号一样，开始先寄存一半数据，等 $x(12)$ 到达时，第一级运算器又开始下一个循环的运算。

3.8 FFT的流水线工作原理

9/23/2022

38

同样，第二级、第三级的工作由于蝶形结配对的要求，都必须等前一级工作一半以后才能开始工作，如果各级都按同一节拍工作，则每一级都可以在半个周期的时间内完成各自的运算，因而整个运算可以连续进行下去。

一个N点FFT流水线运算，需要  $M = \log_2 N$  级运算器，每级的速度只要求1次乘法/每采样周期就够了，从而大大降低了对于硬件的速度要求，或者同样的硬件可以完成更高速度的信号处理。

这样做的代价是运算器需要增加M倍，存储单元也要增加约M/2倍。  
另外，最后所获得的结果也必须有一定的时延。

从FFT流水线工作示意图中可以看到，每级运算器实际上只有一半时间在工作，工作效率只有一半，这是因为，一个N点FFT要求运算速度为  $\frac{1}{2} \log_2 N$  次乘/采样周期，这样的运算量被速度为1次乘/采样周期的运算器分担时，只要  $\frac{1}{2} \log_2 N = \frac{M}{2}$  个就够了，或者说由  $\frac{M}{2}$  个运算器分担的话，每个运算器的速度只要目前的一半就够了，这说明即使采用流水线方案也还有一半的潜力可以挖掘，这种潜力往往可以根据处理信号的具体要求加以进一步利用。