第 5 章

CHAPTER 5

快速傅里叶变换

DFT 和 IDFT 的求和可以简单地看成是在一个域中取出 N 个数并且在另一个域中产生 N 个(复)数的一种计算方法。虽然 DFT 是一种重要的数字信号处理工具,能实现信号时域和频域的转换,但是几乎很少使用。因为直接计算 DFT 的计算量与变换区间长度 N 的平方成正比,当 N 较大时,计算量太大,所以在快速傅里叶变换(Fast Fourier Transform,FFT)出现以前,直接用 DFT 算法进行谱分析和信号的实时处理是不切实际的。

由于 FFT 的输出与 DFT 的输出相同,但运算量却少很多,所以被广泛用于计算机计算。本章将介绍 FFT 的经典算法并给出一些应用举例,内容包括:

- DFT 的运算量及基本对策;
- 按时域抽选(DIT-FFT)的基-2FFT 算法;
- FFT 算法与 DFT 运算量的对比;
- 按频域抽选的基-2FFT 算法;
- IDFT 的高效算法;
- 进一步减少运算量;
- FFT 应用举例。

5.1 DFT 的运算量及基本对策

下面分析一下,为什么直接计算 DFT,当 N 较大时,计算量会很大。

设 $x(n)$为有限长序列,其 N 点 DFT 为

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}\quad k=0,1,\cdots,N-1 \tag{5.1}$$

因为序列 $x(n)$不一定是实数序列,所以考虑 $x(n)$为复数序列的一般情况。若直接按式(5.1)计算,对任一个 k 值,比如求 $k=0$ 时 $X(k)$的值,

$$X(k=0)=x(0)W_N^{0\times 0}+x(1)W_N^{1\times 0}+x(2)W_N^{2\times 0}+\cdots+x(N-1)W_N^{(N-1)\times 0}$$

则需要 N 次复数乘法和$(N-1)$次复数加法。因此,计算出 $X(k)$全部的 N 个值,共需要 N^2 次复数乘法和 $N(N-1)$次复数加法运算。当 $N\gg 1$ 时,$N(N-1)\approx N^2$。这样看来,对于一个 N 点 DFT 的复数乘法和复数加法的运算次数大概都为 N^2 次。DFT 运算量如表 5.1 所示。

表 5.1　DFT 的运算量

	复数乘法的次数	复数加法的次数
求一个 $X(k)$	N	$N-1$
求所有的 $X(k)$	$N \cdot N$	$N \cdot (N-1)$

从上面的分析可以知道，当 N 较大时，DFT 的运算量将非常大。例如 $N=1024$ 时，$N^2=1\,048\,576$，这对于实时信号处理滤波器的计算速度来说，将是难以实现的。因此，必须将 DFT 的运算量尽可能地减少，才能使 DFT 在各种科学和工程计算中得到真正的应用。

从表 5.1 可以看出，N 点 DFT 的复数乘法次数与 N^2 有关，如果能把 N 点 DFT 分解为几个较短点数的 DFT，将使计算复数乘法的次数大大减少。

通过分析 DFT 变换公式中 W_N^m 的表达式，可以发现旋转因子 W_N^m 具以下这些性质：

(1) 周期性

$$W_N^{(n+lN)\cdot k}=W_N^{n\cdot(k+lN)}=W_N^{nk} \quad l \text{ 为整数} \tag{5.2}$$

(2) 共轭对称性

$$(W_N^{nk})^*=W_N^{-nk} \tag{5.3}$$

(3) 可约性

$$W_N^{nk}=W_{mN}^{mnk}=W_{\frac{N}{m}}^{\frac{nk}{m}} \quad m \text{ 为整数},\frac{N}{m} \text{ 为整数} \tag{5.4}$$

由上面这些性质，还可得出一些特殊值

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

利用这些性质，可以将 DFT 中的一些项合并。

因此，FFT 算法减少运算量的基本途径就是不断地把长序列的 DFT 分解成几个短序列的 DFT，并利用旋转因子 W_N^m 的周期性和对称性来减少 DFT 的运算次数。基-2FFT 算法是 FFT 算法中最简单最常用的，该算法分为两类：时域抽选法 FFT(Decimation-In-Time Fast Fourier Transform，DIT-FFT)；频域抽选法 FFT(Decimation-In-Frequency Fast Fourier Transform，DIF-FFT)。

【随堂练习】

试证明旋转因子 W_N^{nk} 的下列性质：①$W_N^{(n+lN)\cdot k}=W_N^{n\cdot(k+lN)}=W_N^{nk}$　l 为整数；②$(W_N^{nk})^*=W_N^{-nk}$；③$W_N^{nk}=W_{mN}^{mnk}=W_{\frac{N}{m}}^{\frac{nk}{m}}$　m 为整数，$\frac{N}{m}$ 为整数。

5.2　按时间抽选的基-2FFT 算法

按时间抽选(Decimation-In-Time，DIT)的基-2FFT 算法可以简写为 DIT-FFT 算法，也称为库利-图基算法。设序列 $x(n)$ 长度为 N，且满足 $N=2^M$，M 为自然数。按 n 的奇偶性把 $x(n)$ 分成以下两组长度为 $\frac{N}{2}$ 点的子序列：

$$x_1(r)=x(2r) \quad r=0,1,\cdots,\frac{N}{2}-1 \tag{5.5}$$

$$x_2(r)=x(2r+1) \quad r=0,1,\cdots,\frac{N}{2}-1 \tag{5.6}$$

则 $x(n)$的 N 点 DFT 为

$$\begin{aligned}X(k)&=\sum_{n=\text{偶数}}x(n)W_N^{nk}+\sum_{n=\text{奇数}}x(n)W_N^{nk}\\&=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2kr}+\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_N^{k(2r+1)}\\&=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_N^{2kr}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_N^{2kr}\end{aligned}\tag{5.7}$$

利用旋转因子 W_N^{nk} 的可约性，即 $W_N^{2kr}=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}2kr}=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N/2}kr}=W_{N/2}^{kr}$，上式可以表示为

$$\begin{aligned}X(k)&=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{kr}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_{N/2}^{kr}\\&=X_1(k)+W_N^kX_2(k)\quad k=0,1,\cdots,N-1\end{aligned}\tag{5.8}$$

其中 $X_1(k)$和 $X_2(k)$分别为 $x_1(r)$和 $x_2(r)$的 $N/2$ 点 DFT，即

$$X_1(k)=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{N/2}^{kr}=\mathrm{DFT}[x_1(r)]_{N/2}\tag{5.9}$$

$$X_2(k)=\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_{N/2}^{kr}=\mathrm{DFT}[x_2(r)]_{N/2}\tag{5.10}$$

并且 $X_1(k)$和 $X_2(k)$均以 $N/2$ 为周期。因为 $X(k)$有 N 个点，用式(5.8)计算得到的只是 $X(k)$的前面一半项数的值，若要用 $X_1(k)$和 $X_2(k)$表达出所有 $X(k)$的值，还需利用旋转因子 W_N^{nk} 的性质 $W_N^{k+\frac{N}{2}}=-W_N^k$，把 $X(k)$表示为前后两部分：

前半部分 $X(k)\left(k=0,1,\cdots,\dfrac{N}{2}-1\right)$可表示为

$$X(k)=X_1(k)+W_N^kX_2(k)\quad k=0,1,\cdots,\frac{N}{2}-1\tag{5.11}$$

后半部分 $X(k)\left(k=\dfrac{N}{2},\cdots,N-1\right)$可表示为

$$\begin{aligned}X\left(k+\frac{N}{2}\right)&=X_1\left(k+\frac{N}{2}\right)+W_N^{\left(k+\frac{N}{2}\right)}X_2\left(k+\frac{N}{2}\right)\\&=X_1(k)-W_N^kX_2(k)\quad k=0,1,\cdots,\frac{N}{2}-1\end{aligned}\tag{5.12}$$

这样，只需求出两个长度为 $N/2$ 序列 $x_1(r)$和 $x_2(r)$的 $N/2$ 点 DFT $X_1(k)$和 $X_2(k)$的值，就可以求出 $X(k)$在 $0\leqslant k\leqslant N-1$ 内的所有值，运算量大大减少。

式(5.11)和式(5.12)的运算可用图 5.1 所示的流图符号表示，称为蝶形运算符号。采用这种图示法，经过一次奇偶抽取分解后，一个 8 点 DFT 运算可用图 5.2 表示。图中，$N=2^3=8$，$X(0)\sim X(3)$由式(5.11)给出，而 $X(4)\sim X(7)$则由式(5.12)给出。

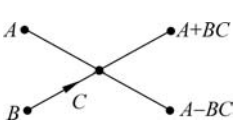

图 5.1 蝶形运算符号

由图 5.1 可见，要完成一个蝶形运算，需要一次复数乘法

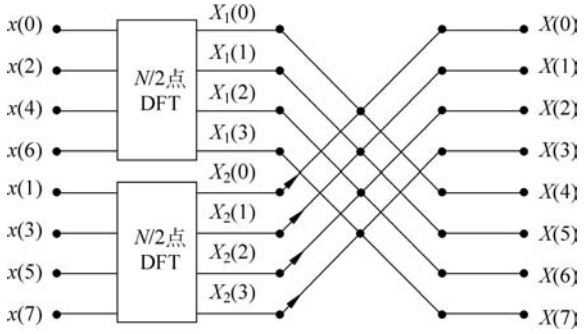

图 5.2　8 点 DFT 一次时域抽取分解运算流图

和两次复数加法运算。由图 5.2 容易看出，经过一次分解后，计算 1 个 N 点 DFT 共需要计算两个 $N/2$ 点 DFT 和 $N/2$ 个蝶形运算。而计算一个 $N/2$ 点 DFT 需要 $(N/2)^2$ 次复数乘法和 $(N/2)(N/2-1)$ 次复数加法。所以，按图 5.2 计算 N 点 DFT 时，总的复数乘法次数为

$$2\left(\frac{N}{2}\right)^2+\frac{N}{2}=\frac{N(N+1)}{2}\bigg|_{N\gg 1}\approx\frac{N^2}{2} \tag{5.13}$$

复数加法次数为

$$N\left(\frac{N}{2}-1\right)+\frac{2N}{2}=\frac{N^2}{2} \tag{5.14}$$

由此可见，仅仅经过一次分解，就使运算量减少近一半。既然这样分解对减少 DFT 的运算量是有效的，且 $N=2^M$，$N/2$ 仍然是偶数，故可以对 $N/2$ 点 DFT 再作进一步分解。表 5.2 列出了经过一次分解与不进行分解 DFT 运算量的比较。

表 5.2　经过一次分解与不进行分解 DFT 运算量的比较

		不分解 $X(k)$（长度为 N）	一次分解	
			$X_1(k)\left(\text{长度为}\frac{N}{2}\right)$	$X_2(k)\left(\text{长度为}\frac{N}{2}\right)$
DFT 里的运算	复数乘法	N^2 次	$\left(\frac{N}{2}\right)^2$ 次	$\left(\frac{N}{2}\right)^2$ 次
	复数加法	$N(N-1)$次	$\frac{N}{2}\left(\frac{N}{2}-1\right)$次	$\frac{N}{2}\left(\frac{N}{2}-1\right)$次
蝶形里的运算	复数乘法	—	$\frac{N}{2}$次	
	复数加法	—	$\frac{N}{2}+\frac{N}{2}=N$ 次	
合计	复数乘法	N^2 次	$2\left(\frac{N}{2}\right)^2+\frac{N}{2}=\frac{N(N+1)}{2}\Big\|_{N\gg 1}\approx\frac{N^2}{2}$次	
	复数加法	$N(N-1)$次	$2\times\frac{N}{2}\left(\frac{N}{2}-1\right)+2\times\frac{N}{2}=\frac{N^2}{2}$次	

与第一次分解相同，将 $x_1(r)$ 按奇偶分解成两个 $N/4$ 点的子序列 $x_3(l)$ 和 $x_4(l)$，即

$$x_3(l)=x_1(2l)\quad l=0,1,\cdots,\frac{N}{4}-1 \tag{5.15}$$

$$x_4(l)=x_1(2l+1)\quad l=0,1,\cdots,\frac{N}{4}-1 \tag{5.16}$$

$X_1(k)$又可表示为

$$\begin{aligned}X_1(k)&=\sum_{l=0}^{\frac{N}{4}-1}x_1(2l)W_{N/2}^{2kl}+\sum_{l=0}^{\frac{N}{4}-1}x_1(2l+1)W_{N/2}^{k(2l+1)}\\&=\sum_{l=0}^{\frac{N}{4}-1}x_3(l)W_{N/4}^{kl}+W_{N/2}^{k}\sum_{l=0}^{\frac{N}{4}-1}x_4(l)W_{N/4}^{kl}\\&=X_3(k)+W_{N/2}^{k}X_4(k)\quad k=0,1,\cdots,\frac{N}{2}-1\end{aligned} \tag{5.17}$$

其中

$$X_3(k)=\sum_{l=0}^{\frac{N}{4}-1}x_3(l)W_{N/4}^{kl}=\mathrm{DFT}[x_3(l)]_{N/4} \tag{5.18}$$

$$X_4(k)=\sum_{l=0}^{\frac{N}{4}-1}x_4(l)W_{N/4}^{kl}=\mathrm{DFT}[x_4(l)]_{N/4} \tag{5.19}$$

同理，由 $X_3(k)$和 $X_4(k)$的周期性和 $W_{N/2}^{m}$ 的对称性($W_{N/2}^{m+\frac{N}{4}}=-W_{N/2}^{m}$)最后得到：

$$X_1(k)=X_3(k)+W_{N/2}^{k}X_4(k)\quad k=0,1,\cdots,\frac{N}{4}-1 \tag{5.20}$$

$$X_1\left(k+\frac{N}{4}\right)=X_3(k)-W_{N/2}^{k}X_4(k)\quad k=0,1,\cdots,\frac{N}{4}-1 \tag{5.21}$$

用同样的方法可计算出

$$X_2(k)=X_5(k)+W_{N/2}^{k}X_6(k)\quad k=0,1,\cdots,\frac{N}{4}-1 \tag{5.22}$$

$$X_2\left(k+\frac{N}{4}\right)=X_5(k)-W_{N/2}^{k}X_6(k)\quad k=0,1,\cdots,\frac{N}{4}-1 \tag{5.23}$$

其中

$$X_5(k)=\sum_{l=0}^{\frac{N}{4}-1}x_5(l)W_{N/4}^{kl}=\mathrm{DFT}[x_5(l)]_{N/4} \tag{5.24}$$

$$X_6(k)=\sum_{l=0}^{\frac{N}{4}-1}x_6(l)W_{N/4}^{kl}=\mathrm{DFT}[x_6(l)]_{N/4} \tag{5.25}$$

$$x_5(l)=x_2(2l)\quad l=0,1,\cdots,\frac{N}{4}-1 \tag{5.26}$$

$$x_6(l)=x_2(2l+1)\quad l=0,1,\cdots,\frac{N}{4}-1 \tag{5.27}$$

这样，经过第二次分解，又将一个 $N/2$ 点 DFT 分解为 2 个 $N/4$ 点 DFT 以及式(5.20)和式(5.21)所示的 $N/4$ 个蝶形运算，如图 5.3 所示。以此类推，经过 M 次分解，最后将 N 点 DFT 分解 N 个 1 点 DFT 和 M 级蝶形运算(每级有 $N/2$ 个蝶形运算)，而 1 点 DFT 就

是时域序列本身。一个完整的 8 点 DIT-FFT 运算流图如图 5.4 所示。图中用到关系式 $W_{N/m}^{k}=W_{N}^{mk}$。图中输入序列虽不是顺序排列，但其排列也是有规律的。图中的数组 A 用于存放输入序列和每级运算结果。

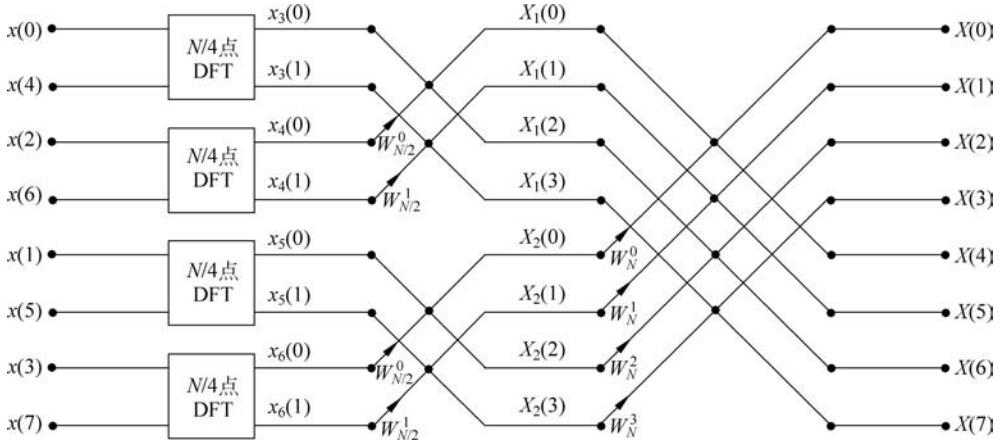

图 5.3　将一个 N 点 DFT 按时域抽取分解为 4 个 $N/4$ 的分解运算流图($N=8$)

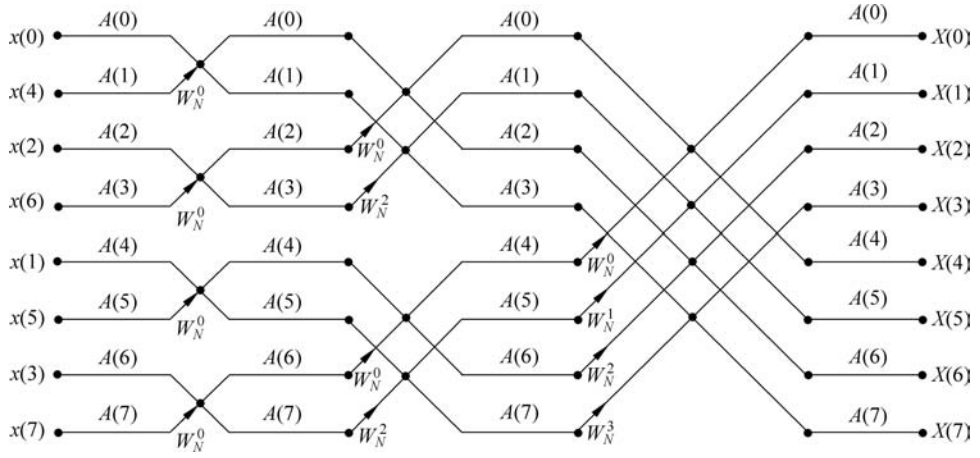

图 5.4　8 点 DIT-FFT 运算流图

在利用时域抽取法的过程中，若序列 $x(n)$ 长度不满足 $N=2^{M}$，则可以在序列后面补零，使序列长度满足。补零后，时域点数增加，但有效数值不变，不会增加信号的信息，也不会提高 DFT 频谱的准确性，故频谱 $X(e^{j\omega})$ 不变，只是频谱的抽样点数增加，抽样点的位置有所改变，频率间隔减小而已。

【例 5.1】 假定现有 8 点按时间抽取的 FFT 芯片，如何利用这些芯片计算一个 24 点 DFT?

解：一个 24 点 DFT 的定义是

$$X(k)=\sum_{n=0}^{23}x(n)W_{24}^{nk}$$

对 $x(n)$ 以因子 3 进行分解，就可以将这个 DFT 分解为如下的 3 个 8 点 DFT：

$$X(k)=\sum_{n=0}^{7}x(3n)W_{24}^{3nk}+\sum_{n=0}^{7}x(3n+1)W_{24}^{(3n+1)k}+\sum_{n=0}^{7}x(3n+2)W_{24}^{(3n+2)k}$$

$$=\sum_{n=0}^{7}x(3n)W_{8}^{nk}+W_{24}^{k}\sum_{n=0}^{7}x(3n+1)W_{8}^{nk}+W_{24}^{2k}\sum_{n=0}^{7}x(3n+2)W_{8}^{nk}$$

所以，如果组成 3 个序列

$$x_1(n)=x(3n)\quad n=0,1,\cdots,7$$

$$x_2(n)=x(3n+1)\quad n=0,1,\cdots,7$$

$$x_3(n)=x(3n+2)\quad n=0,1,\cdots,7$$

并利用 8 点 FFT 芯片计算出它们所对应的 $X_1(k)$、$X_2(k)$和 $X_3(k)$，那么 $x(n)$的 24 点 DFT 就可以通过将 8 点 FFT 的输出组合起来求得，组合的方法如下：

$$X(k)=X_1(k)+W_{24}^{k}X_2(k)+W_{24}^{2k}X_3(k)$$

【随堂练习】

假定现有 5 点和 3 点按时间抽取 FFT 芯片，如何利用这些芯片计算 15 点 DFT?

5.3 DIT-FFT 算法与直接计算 DFT 运算量的对比

根据表 5.1 所列，对于每个点，标准的 DFT 要计算 N 次复数乘法和$(N-1)$次复数加法。那么，如果要计算所有 N 个点，总共要 N^2 次复数乘法和 $N(N-1)$次复数加法。运算的次数常用来评价该运算的难易程度。因此，DFT 的运算难度系数与 N^2 成正比。根据上节 DIT-FFT 算法的分解过程及图 5.4 可知，当 $N=2^M$ 时，其运算流图最多可分解为 M 级蝶形，每一级都有 $N/2$ 个蝶形运算构成。因此，FFT 的每一级运算包括需要 $N/2$ 次复数乘法和 N 次复数加法(每个蝶形运算需要两次复数加法)。所以，M 级运算总的复数乘法的次数为

$$C_M=\frac{N}{2}\cdot M=\frac{N}{2}\log_2 N \tag{5.28}$$

复数加法的次数为

$$C_A=N\cdot M=N\log_2 N \tag{5.29}$$

可以看出，DIT-FFT 的运算难度系数与 $N\log_2 N$ 成正比。

当 $N\gg 1$ 时，$N^2\gg\frac{N}{2}\log_2 N$，所以 DIT-FFT 算法比直接计算 DFT 的复数乘法的运算次数大大减少。例如，$N=2^{10}=1024$ 时，

$$\frac{N^2}{\frac{N}{2}\log_2 N}=\frac{1\,048\,576}{5120}=204.8$$

这样，就使复数乘法的运算效率提高 200 多倍。同理，DIT-FFT 的复数加法的运算效率也可以提高 100 多倍。

由于 DIT-FFT 运算如此高效，它的计算几乎总是在采样点数为 2 的整数次幂的基础上进行。当信号的采样点数不是 2 的整数次幂时，需要在信号的末尾补零。根据式(3.2)，多

加的零值不会影响信号的 DTFT。又因为 DFT 或 FFT 只是 DTFT 的在频域的采样形式，补零对信号的 DFT 或 FFT 特性没有影响。

图 5.5 为 DIT-FFT 算法和直接计算 DFT 所需复数乘法次数 C_M 与变换点数 N 的关系曲线。由此图更加直观地看出 FFT 算法的优越性，显然，N 越大时，优越性就越明显。

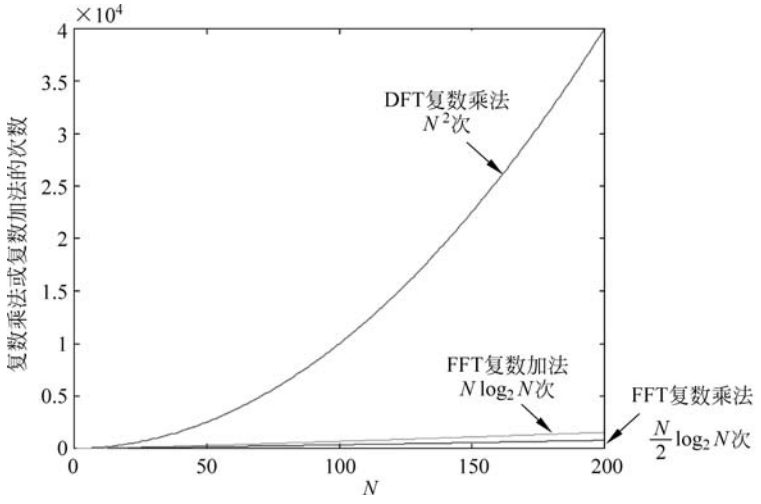

图 5.5　DIT-FFT 算法与直接计算 DFT 运算效率比较曲线

5.4　按频率抽选的基-2FFT 算法

由于 DFT 和 IDFT 的变换前后时域和频域序列的长度相同，求解表达式形式相近，因此，对照时域抽取法，又产生了基-2FFT 算法中的另一种按频域抽选（Decimation-In-Frequency，DIF）的基-2FFT 算法（DIF-FFT），又称为桑德-图基算法。频域抽取法的实现过程如下：

设序列 $x(n)$ 长度为 $N=2^M$，首先将 $x(n)$ 分成前后两半，其 DFT 表示为如下形式：

$$\begin{aligned}X(k)&=\mathrm{DFT}[x(n)]=\sum_{n=0}^{N-1}x(n)W_N^{kn}\\&=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{kn}+\sum_{n=\frac{N}{2}}^{N-1}x(n)W_N^{kn}\\&=\sum_{n=0}^{\frac{N}{2}-1}x(n)W_N^{kn}+\sum_{n=0}^{\frac{N}{2}-1}x\left(n+\frac{N}{2}\right)W^{k\left(n+\frac{N}{2}\right)}\\&=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+W_N^{k\frac{N}{2}}x\left(n+\frac{N}{2}\right)\right]W_N^{kn}\end{aligned}\tag{5.30}$$

式中

$$W_N^{k\frac{N}{2}}=(-1)^k=\begin{cases}1 & k\text{ 为偶数}\\ -1 & k\text{ 为奇数}\end{cases}$$

将 $X(k)$分成了偶数组和奇数组两组。当 k 取偶数$\left(k=2m,m=0,1,\cdots,\dfrac{N}{2}-1\right)$时，

$$\begin{aligned}X(2m)&=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_N^{2mn}\\&=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_{N/2}^{mn}\end{aligned}\tag{5.31}$$

当 k 取奇数$\left(k=2m+1,m=0,1,\cdots,\dfrac{N}{2}-1\right)$时，

$$\begin{aligned}X(2m+1)&=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^{(2m+1)n}\\&=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^{n}W_{N/2}^{mn}\end{aligned}\tag{5.32}$$

令

$$\left.\begin{aligned}x_1(n)&=x(n)+x\left(n+\frac{N}{2}\right)\\x_2(n)&=\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^n\end{aligned}\right\}\quad n=0,1,\cdots,\frac{N}{2}-1$$

将 $x_1(n)$和 $x_2(n)$分别代入式(5.31)和式(5.32)中，可得

$$\left.\begin{aligned}X(2m)&=\sum_{n=0}^{\frac{N}{2}-1}x_1(n)W_{N/2}^{mn}\\X(2m+1)&=\sum_{n=0}^{\frac{N}{2}-1}x_2(n)W_{N/2}^{mn}\end{aligned}\right\}\tag{5.33}$$

式(5.33)表明，$X(k)$按奇偶 k 值分为两组，其偶数组是 $x_1(n)$的 $N/2$ 点 DFT，奇数组是 $x_2(n)$的 $N/2$ 点 DFT。$x_1(n)$、$x_2(n)$和 $x(n)$之间的关系也可用图 5.6 所示的蝶形运算流图符号表示。图 5.7 表示 $N=8$ 时第一次分解的运算流图。

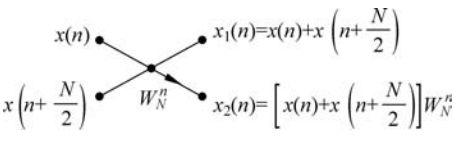

图 5.6 DIF-FFT 蝶形运算流图符号

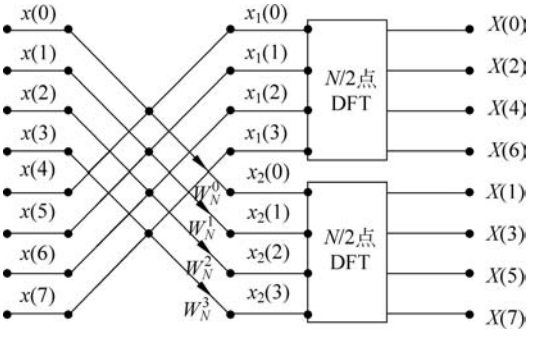

图 5.7 DIF-FFT 第一次分解运算流图($N=8$)

由于 $N=2^M$，可将 $N/2$ 点 DFT 再分成偶数组和奇数组，每个 $N/2$ 点的 DFT 又可分成两个 $N/4$ 点 DFT，其输入序列分别是 $x_1(n)$ 和 $x_2(n)$ 各自按上下对半分开形成的 4 个子序列。图 5.8 表示了 $N=8$ 时第二次分解运算流图，经过两次分解，便分解为 4 个两点 DFT。图 5.9 表示了序列长度 $N=8$ 时的完整 DIF-FFT 运算流图。

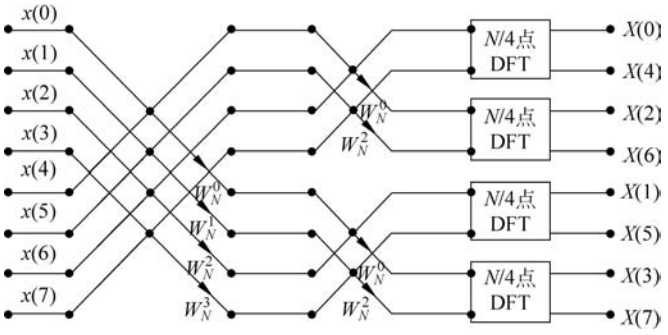

图 5.8　DIF-FFT 第二次分解运算流图($N=8$)

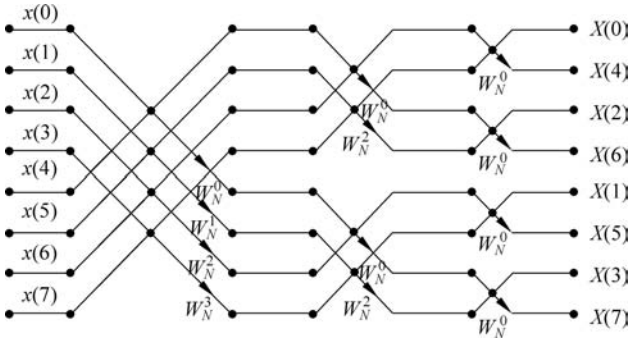

图 5.9　DIF-FFT 第三次分解运算流图($N=8$)

若序列 $x(n)$ 长度为 $N=2^M$，经过 $M-1$ 次分解，最后分解为 $N/2$ 个两点 DFT。在 DIF-FFT 中，两点 DFT 就是一个基本蝶形运算。

这种算法是对 $X(k)$ 进行奇偶抽取分解，因此被称为频域抽取法 FFT。观察图 5.4 和图 5.9 可知，DIF-FFT 算法与 DIT-FFT 算法有下列相同点：可以原位计算；共有 M 级运算；每级共有 $N/2$ 个蝶形运算；两种算法的运算次数也相同。

不同点是：

(1) 输入和输出的顺序不同。DIF-FFT 算法输入序列为自然顺序，而输出为倒序排列；DIT-FFT 算法输入序列为倒序排列，而输出为自然排列。

(2) 蝶形运算不同。DIT-FFT 蝶形先相乘求 DFT 再相加(减)；而 DIF-FFT 蝶形先加(减)再相乘求 DFT。

5.5　IDFT 的高效算法

DIT-FFT 算法和 DIF-FFT 算法也可以用于计算 IDFT。比较 DFT 和 IDFT 的运算公式：

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

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

可以得到求解 IFFT 的第一种算法：将 DFT 表达式中的变换核 W_N^{kn} 换成 W_N^{-kn}，最后再乘以 $1/N$，就是 IDFT 运算公式。此外，DFT 的流图输入的是 $x(n)$，输出是 $X(k)$；而 IDFT 的流图输入的是 $X(k)$，输出的是 $x(n)$。因此，原来的 DIT-FFT 改为 IFFT 后，称为 DIF-IFFT 更合适；DIF-FFT 改为 IFFT 后，称为 DIT-IFFT 更合适。以 DIF-FFT 流图为例，可得到图 5.10 所示的 IFFT 流图。

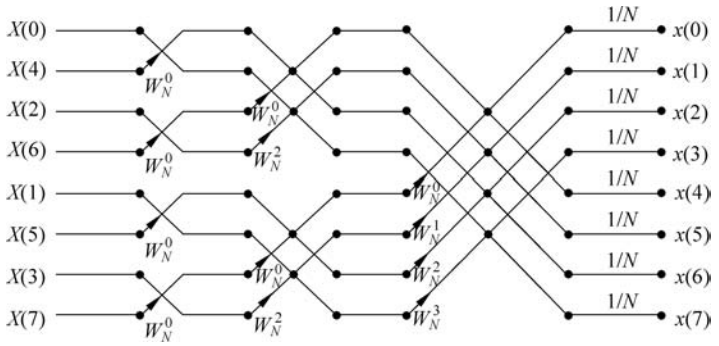

图 5.10　$N=8$ 基-2IFFT 流图

第二种算法可以直接利用 FFT 程序。由于 IDFT 计算公式为

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

两边取共轭，可得

$$\begin{aligned}x^*(n)&=\frac{1}{N}\left[\sum_{k=0}^{N-1}X(k)W_N^{-kn}\right]^*\\&=\frac{1}{N}\sum_{k=0}^{N-1}X^*(k)W_N^{kn}\\&=\frac{1}{N}\mathrm{DFT}[X^*(k)]\end{aligned}$$

$$x(n)=\frac{1}{N}\left[\sum_{k=0}^{N-1}X^*(k)W_N^{kn}\right]^*=\frac{1}{N}\{\mathrm{DFT}[X^*(k)]\}^* \tag{5.34}$$

所以，求解 IFFT 可以先将 $X(k)$取复共轭，然后直接调用 FFT 子程序，进行 DFT 运算，最后将所得结果取复共轭并乘以 $1/N$，便得到序列 $x(n)$。这种方法虽然用了两次取复共轭运算，但可以与 FFT 共用同一程序，因此用起来也比较方便。

第三种算法也是利用现有的 FFT 程序。令

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

则有

$$x(n)=\frac{1}{N}p(N-n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{(N-n)k}=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-nk} \tag{5.35}$$

所以这种共用 FFT 程序的步骤是：

(1) 利用 FFT 程序由 $X(k)$求出 $p(n)$；

(2) 计算$\frac{1}{N}p(N-n)$即为 $x(n)$[注意 $n=0$ 时 $p(N)=p(0),x(0)=p(0)/N$]。

5.6　进一步减少运算量

通过观察以上 DIT-FFT 和 DIF-FFT 算法，工程师们发现还可以进一步减少运算量，下面简单介绍几种方法。

5.6.1　多类蝶形单元运算

由 DIT-FFT 运算流图可知，$N=2^M$ 点 FFT 共需要 $MN/2$ 次复数乘法。由旋转因子

$$W_N^p=W_{N\cdot 2^{L-M}}^{J}=W_N^{J\cdot 2^{M-L}}\quad J=0,1,2,\cdots,2^{L-1}-1 \tag{5.36}$$

其中，

$$p=J\cdot 2^{M-L} \tag{5.37}$$

当 $L=1$ 时，只有一种旋转因子 $W_N^0=1$，所以第一级不需要乘法运算。当 $L=2$ 时，共有两个旋转因子 $W_N^0=1$ 和 $W_N^{N/4}=-\mathrm{j}$，因此，第二级也不需要乘法运算。在 DFT 中，又称其值为±1 和±j 的旋转因子为无关紧要的旋转因子，如 $W_N^0=1,W_N^{N/2},W_N^{N/4}$ 等。

因此，除去第一、二两级后，所需复数乘法次数应是

$$C_M=\frac{N}{2}(M-2) \tag{5.38}$$

再来找一下各级中的无关紧要旋转因子。当 $L=2$ 时，有两个无关紧要的旋转因子 W_N^0 和 $W_N^{N/4}$，因为同一旋转因子对应着 $2^{M-L}=\frac{N}{2^L}$蝶形运算，所以第二级共有$\frac{N}{2^2}=\frac{N}{4}$个蝶形不需要复数乘法运算。当 $L\geqslant 3$ 时，第 L 级的两个无关紧要的旋转因子减少复数乘法的次数为$\frac{2N}{2^L}=\frac{N}{2^{L-1}}$。因此，从 $L=3$ 至 $L=M$ 共减少的复数乘法次数为

$$\sum_{L=3}^{M}\frac{N}{2^{L-1}}=2N\sum_{L=3}^{M}\left(\frac{1}{2}\right)^L=\frac{N}{2}-2 \tag{5.39}$$

这样，DIT-FFT 的复数乘法次数为

$$C_M=\frac{N}{2}(M-2)-\left(\frac{N}{2}-2\right)=\frac{N}{2}(M-3)+2 \tag{5.40}$$

进一步观察 FFT 中存在一些特殊的复数运算以进一步减少复数乘法次数。一般实现一次复数乘法运算需要四次实数乘，两次实数加。但对 $W_N^{N/8}=(1-\mathrm{j})\frac{\sqrt{2}}{2}$这一特殊复数，任一复数$(x+\mathrm{j}y)$与其相乘时，有以下等式：

$$(1-\mathrm{j})\frac{\sqrt{2}}{2}(x+\mathrm{j}y)=\frac{\sqrt{2}}{2}(x+\mathrm{j}y-\mathrm{j}x+y)$$

$$=\frac{\sqrt{2}}{2}[(x+y)-\mathrm{j}(x-y)]=R+\mathrm{j}I$$

其中，$R=\sqrt{2}/2(x+y)$，$I=-\sqrt{2}/2(x-y)=\sqrt{2}/2(y-x)$，它只需要两次实数加和两次实数乘就可实现。这样，$W_N^{N/8}$ 对应的每个蝶形节省两次实数乘。在 DIT-FFT 运算流图中，从 $L=3$ 至 $L=M(M>3)$级，每级都包含旋转因子 $W_N^{N/8}$，第 L 级中，$W_N^{N/8}$ 对应$\frac{N}{2^L}$个蝶形运算。因此从第三级至最后一级，旋转因子 $W_N^{N/8}$ 节省的实数乘次数与式(5.40)相同。所以从实数乘运算考虑，计算 $N=2^M$ 点 DIT-FFT 所需实数乘法次数为

$$R_M=4\left[\frac{N}{2}(M-3)+2\right]-\left(\frac{N}{2}-2\right)=N\left(2M-\frac{13}{2}\right)+10 \tag{5.41}$$

在基-2FFT 程序中，将蝶形单元运算分为 N 类，包含了所有旋转因子的称为一类蝶形单元运算；去掉 $W_N^m=\pm1$ 的旋转因子的称为二类蝶形单元运算；再去掉 $W_N^m=\pm\mathrm{j}$ 的旋转因子的称为三类蝶形单元运算；如果再判断处理 $W_N^m=(1-\mathrm{j})\frac{\sqrt{2}}{2}$，则称为四类蝶形单元运算。后三种运算称为多类蝶形单元运算。显然，蝶形单元类型越多，编程的复杂程度越高，但当 N 较大时，可大大减少乘法运算量。例如，$N=4096$ 时，三类蝶形单元运算的乘法次数为一类蝶形单元运算量的 75%。

对于其他旋转因子 $W_N^m=\cos\left(\frac{2\pi m}{N}\right)-\mathrm{j}\sin\left(\frac{2\pi m}{N}\right)$，由于这些是需要正弦和余弦函数值的计算，因此计算量很大。可以在 FFT 程序开始前，预先计算出 W_N^m，$m=0,1,\cdots,\frac{N}{2}-1$，存放在数组中，作为旋转因子表，在程序执行过程中直接查表，这样可使运算速度大大提高，但占用的内存较多。

5.6.2 减少实序列的 FFT 的方法

对于许多信号，如数字语音信号，数据 $x(n)$是实数序列。如果把 $x(n)$看成一个虚部为零的复序列进行计算，这就增加了存储量和运算时间。根据实序列的 FFT 的特点可以有多种 FFT 运算量的方法，其中之一是用 $N/2$ 点 FFT 计算一个 N 点实序列的 DFT。以下介绍这种方法。

设 $x(n)$为 N 点实序列，取 $x(n)$的偶数点和奇数点分别作为新构造序列 $y(n)$的实部和虚部，即

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

$$x_2(n)=x(2n+1)\quad n=0,1,\cdots,\frac{N}{2}-1$$

$$y(n)=x_1(n)+\mathrm{j}x_2(n)\quad n=0,1,\cdots,\frac{N}{2}-1$$

对 $y(n)$进行 $N/2$ 点 FFT，输出 $Y(k)$，则

$$X_1(k)=\mathrm{DFT}[x_1(n)]=Y_{ep}(k)\quad k=0,1,\cdots,\frac{N}{2}-1$$

$$X_2(k)=\mathrm{DFT}[x_2(n)]=-\mathrm{j}Y_{op}(k)\quad k=0,1,\cdots,\frac{N}{2}-1$$

根据 DIT-FFT 及式(4.16)和(4.17),可得到 $X(k)$的前$\frac{N}{2}+1$ 个值:

$$X(k)=X_1(k)+W_N^kX_2(k)\quad k=0,1,\cdots,\frac{N}{2} \tag{5.42}$$

式中,$X_1\left(\frac{N}{2}\right)=X_1(0)$,$X_2\left(\frac{N}{2}\right)=X_2(0)$。由于 $x(n)$为实序列,因此 $X(k)$具有共轭对称性,$X(k)$的另外 $N/2$ 点的值为

$$X(N-k)=X^*(k)\quad k=0,1,\cdots,\frac{N}{2}-1$$

下面来计算按以上方法运算速度的提高程度。计算 $N/2$ 点 FFT 的复乘次数为$\frac{N}{4}(M-1)$,计算式(5.41)的复乘次数为$\frac{N}{2}$,因此用这种算法,计算 $X(k)$所需复数乘法次数为$\frac{N}{4}(M-1)+\frac{N}{2}=\frac{N}{4}(M+1)$。相对一般的$N$ 点 FFT 算法,上述算法的运算效率为 $\eta=\frac{\frac{N}{2}M}{\frac{N}{4}(M+1)}=\frac{2M}{M+1}$,例如当 $N=2^M=2^{10}$ 时,$\eta=\frac{20}{11}$,运算速度提高了 82%。

自 1965 年基-2FFT 算法出现以来,经过人们不断研究探索,现在已提出了多种快速算法。例如分裂基 FFT 算法、离散哈特莱变换(DHT)、基-4FFT、基-8FFT、基-rFFT、混合基 FFT,以及进一步减少运算量的途径等内容,它们对研究新的快速算法都是很有用的。这里就不再赘述,相关内容请读者阅读相关的文献。

5.7　FFT 应用举例

在 MATLAB 信号处理工具箱中提供了函数 fft 和 ifft 以进行快速傅里叶变换和逆快速傅里叶变换。快速傅里叶变换函数 fft 的一种调用形式为:

$$\boldsymbol{y}=\mathrm{fft}(\boldsymbol{x}) \tag{5.43}$$

式中,$\boldsymbol{x}$ 是序列,$\boldsymbol{y}$ 是序列的快速傅里叶变换,$\boldsymbol{x}$ 可以为一向量或矩阵,若 $\boldsymbol{x}$ 为向量,则 $\boldsymbol{y}$ 为相同长度的向量;若 $\boldsymbol{x}$ 为矩阵,则 $\boldsymbol{y}$ 是对矩阵的每一列向量进行 FFT 运算。

快速傅里叶变换函数 fft 的另一种调用形式为:

$$\boldsymbol{y}=\mathrm{fft}(\boldsymbol{x},N) \tag{5.44}$$

式中,N 表示函数执行 N 点的 FFT。若 $\boldsymbol{x}$ 为向量且长度小于 N,则函数将 $\boldsymbol{x}$ 补零到长度 N,若向量 $\boldsymbol{x}$ 的长度大于 N,则函数截断 $\boldsymbol{x}$ 使之长度为 N。

函数 fft 是用机器语言写成,不是用 MATLAB 命令写成,因此执行起来非常快。并且它是作为一种混合基算法写成的,如果运算的长度为 2^n,则就能使用一个高速的基-2FFT 算法。如果运算的长度不是 2^n,则 fft 执行一种称为混合基的 FFT 算法,计算速度慢。

应用 ifft 函数进行逆快速傅里叶变换,它与 fft 具用同样的特性,这里不再赘述。以下通过例题来说明用 fft 函数得到的频谱的特点。

【例 5.2】 已知信号 $x(t)=0.5\sin(2\pi f_1 t)+2\sin(2\pi f_2 t)$，其中 $f_1=20\text{Hz}$，$f_2=50\text{Hz}$，采用频率为 200Hz，以 N 表示数据个数，FFT 采样点数用 L 表示，试分别绘制出在 $N=128$ 点和 $N=1024$ 点的幅频图。程序如下：

```
fs = 200
N = 128
n = 0:N - 1
t = n/fs
x = 0.5 * sin(2 * pi * 20 * t) + 2 * sin(2 * pi * 50 * t)
y = fft(x,N)
m1 = abs(y)
f = (0:N - 1) * fs/N
subplot(2,1,1)
plot(f,m1)
title('N = 128')
grid on
fs = 200
N = 1024
n = 0:N - 1
t = n/fs
x = 0.5 * sin(2 * pi * 20 * t) + 2 * sin(2 * pi * 50 * t)
y = fft(x,N)
m1 = abs(y)
f = (0:N - 1) * fs/N
subplot(2,1,2)
plot(f,m1)
title('N = 1024')
grid on
```

运行结果如图 5.11 所示，显然，整个频谱图以采样频率(200Hz)的一半(100Hz)为对称轴。因此，在利用 fft 函数对信号作谱分析时，取零频率到采样频率的一半即可。另一方面，$N=128$ 和 $N=1024$ 均能很好地分辨两种频率成分：20Hz 和 50Hz。根据式(4.70)，

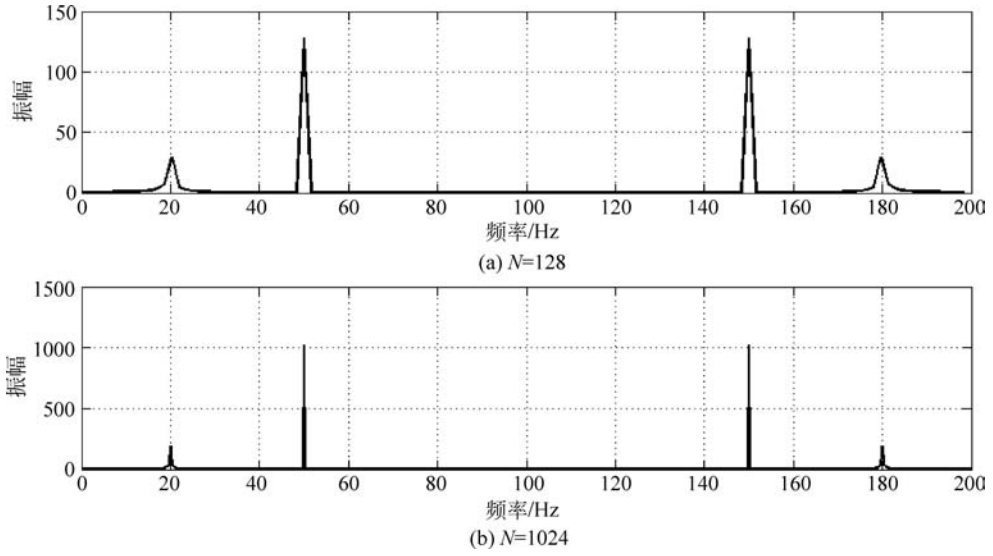

图 5.11 对信号分别做 128 点和 1024 点的快速傅里叶变换幅频图

$N=128$ 比 $N=1024$ 的频谱分辨率低，因此 $N=1024$ 的幅频图中更尖端。最后，幅频图的振幅大小与采样点数直接相关，如果要得到真实振幅，则需将变换后的结果乘以 $2/N$。为此，可将上述作图命令修改为：plot(f(1：N/2)，m1(1：N/2)＊2/N)，重新得到的幅频图如图 5.12 所示。

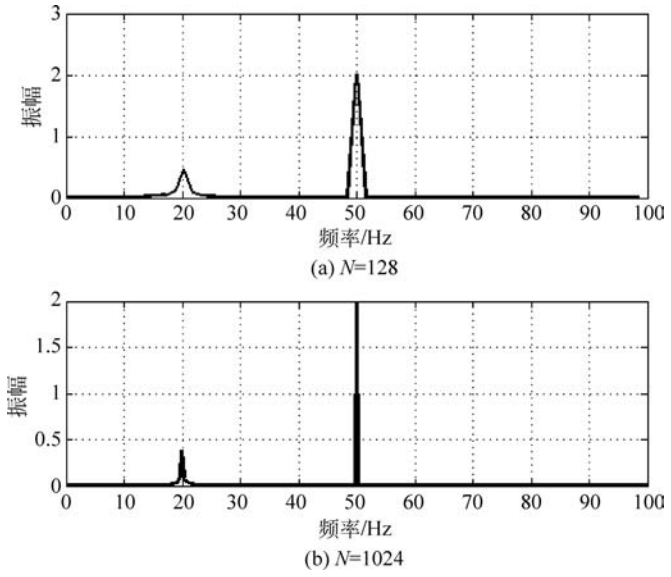

图 5.12　对幅度进行了修订且只取采样频率的一半的快速傅里叶变换幅频图

【例 5.3】　已知信号 $x(t)=0.5\sin(2\pi f_1 t)+2\sin(2\pi f_2 t)$，其中 $f_1=15\text{Hz}$，$f_2=40\text{Hz}$，采用频率为 100Hz，以 N 表示数据个数，FFT 采样点数用 L 表示，在下列情况下绘制其幅频图。

(1) $N=32$，$L=32$；

(2) $N=32$，$L=128$；

(3) $N=136$，$L=128$；

(4) $N=136$，$L=512$。

试分析所用数据长度不同时对傅里叶变换结果的影响。程序如下：

```
fs = 100
l1 = 32
N = 32
n = 0:l1 - 1
t = n/fs
x = 0.5 * sin(2 * pi * 15 * t) + 2 * sin(2 * pi * 40 * t)
y = fft(x,N)
m1 = abs(y)
f = (0:N - 1) * fs/N
subplot(2,2,1)
plot(f(1:N/2),m1(1:N/2) * 2/N)
title('N = 32 L = 32')
grid on
l2 = 128
```

```
N = 32
n = 0:l2 - 1
t = n/fs
x = 0.5 * sin(2 * pi * 15 * t) + 2 * sin(2 * pi * 40 * t)
y = fft(x,N)
m1 = abs(y)
f = (0:N - 1) * fs/N
subplot(2,2,2)
plot(f(1:N/2),m1(1:N/2) * 2/N)
title('N = 32 L = 128')
grid on
l3 = 128
N = 136
n = 0:l3 - 1
t = n/fs
x = 0.5 * sin(2 * pi * 15 * t) + 2 * sin(2 * pi * 40 * t)
y = fft(x,N)
m1 = abs(y)
f = (0:N - 1) * fs/N
subplot(2,2,3)
plot(f(1:N/2),m1(1:N/2) * 2/N)
title('N = 136 L = 128')
grid on
l4 = 512
N = 136
n = 0:l4 - 1
t = n/fs
x = 0.5 * sin(2 * pi * 15 * t) + 2 * sin(2 * pi * 40 * t)
y = fft(x,N)
m1 = abs(y)
f = (0:N - 1) * fs/N
subplot(2,2,4)
plot(f(1:N/2),m1(1:N/2) * 2/N)
title('N = 136 L = 512')
grid on
```

运行结果如图 5.13 所示。

现对以上结果进行分析：

(1) $N=32$，$L=32$ 时，频率分辨率较低；但由于数据个数与 FFT 采用的数据个数相等，因此，不需要添加零而导致其他的频率成分。将振幅乘以 $2/N$ 后，得到了绝对大小的振幅；

(2) $N=32$，$L=128$ 时，FFT 运算需加零，致使振幅中出现了很多其他的成分，其振幅的幅度也由于加零而明显减少；

(3) $N=136$，$L=128$ 时，FFT 运算需要截断 128 个数据，这时频率分辨率较高，将振幅乘以 $2/N$ 后，得到了绝对大小的振幅；

(4) $N=136$，$L=512$ 时，FFT 运算需加零，致使振幅中出现了很多其他的成分，其振幅的幅度也由于加零而明显减少；但是由于含有信号的数据个数足够多，因此，其振幅谱分辨率仍较高。

【例 5.4】 运用 fft 函数对信号 $x=0.5\sin(2\pi \cdot 3 \cdot n \cdot \mathrm{d}t)+\cos(2\pi \cdot 10 \cdot n \cdot \mathrm{d}t)$ 进

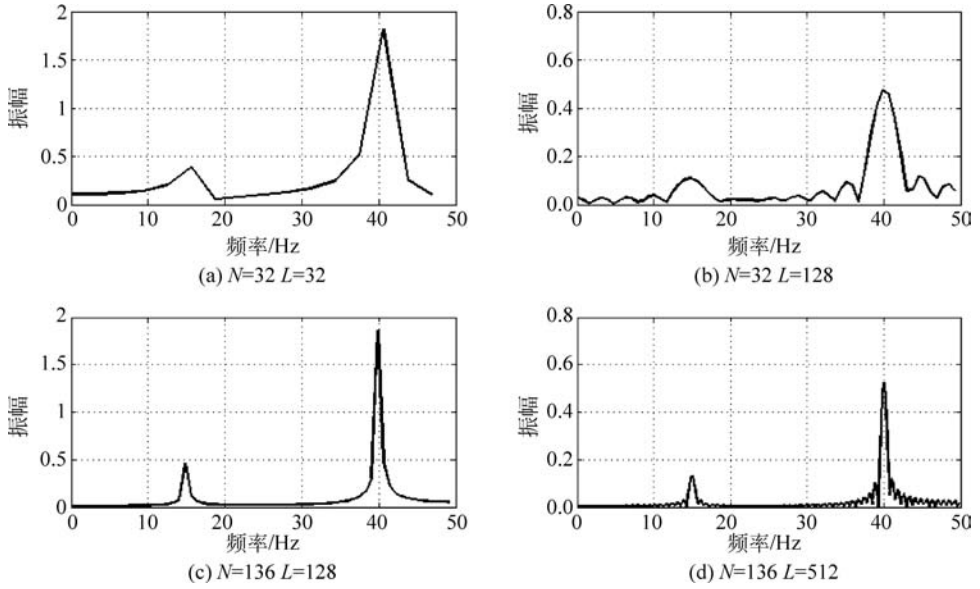

图 5.13　不同采样个数和 fft 计算数据个数对所得频谱的影响

行 512 点的 FFT 运算，并滤去此信号中频率为 8～15Hz 的波。采样间隔 dt＝0.02。绘出滤波前后的振幅谱以及滤波后的时域信号。程序如下：

```
dt = 0.02
N = 512
n = 0:N - 1
t = n * dt
f = n/(N * dt)
f1 = 3
f2 = 10
x = 0.5 * sin(2 * pi * f1 * dt) + cos(2 * pi * f2 * dt)
subplot(2,2,1)
plot(t,x)
y = fft(x,N)
subplot(2,2,2)
plot(f,abs(y) * 2/N)
f3 = 8
f4 = 15
yy = zeros(1,length(y))
for m = 0:N - 1
    if(m/(N * dt)> f3&m/(N * dt)< f4|m/(N * dt)>(1/dt - f4)&m/(N * dt)<(1/dt - f3))
        yy(m + 1) = 0
    else
        yy(m + 1) = y(m + 1)
    end
end
subplot(2,2,4)
plot(f,abs(yy) * 2/N)
x1 = ifft(yy)
subplot(2,2,3)
plot(t,real(x1))
```

运行结果如图 5.14 所示。

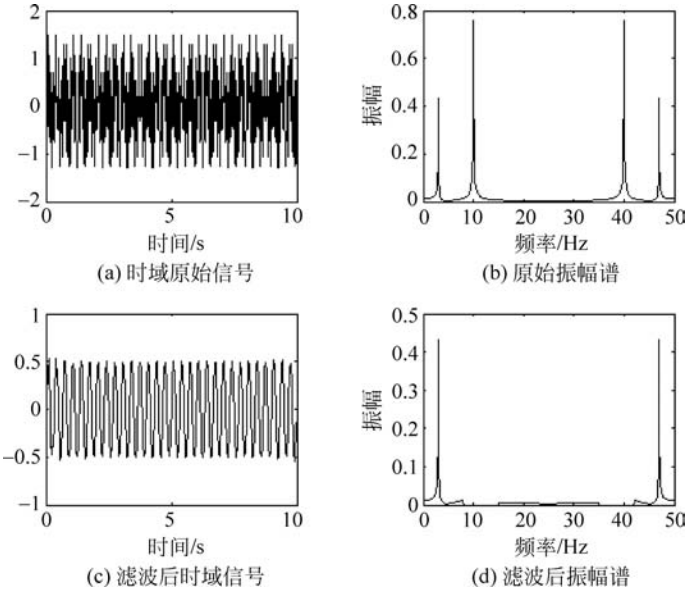

图 5.14　运用 fft 进行信号的滤波

习题

1. 设一个信号 $x(n)=\{0,1,2,3,4,5,6,7\}$。

(1) 用时域抽取法计算其 FFT；

(2) 用(1)中的结果，计算并画出信号的幅度频谱和相位频谱。

2. 对一个长度为 1024 点的信号：

(1) 分别进行 DFT 和 FFT 计算，需要多少次复数乘和复数加？

(2) DFT 的运算量是 FFT 的多少倍？

3. 试证明下面关于 FFT 的旋转因子的等式：

(1) $W_2^0=W_4^0=W_8^0$；

(2) $W_4^1=W_8^2$。

4. 信号以 8kHz 进行采样，进行 512 点的 FFT。

(1) 求 FFT 的频率间隔；

(2) 将信号补零为 4096 个采样点，再次计算 FFT，频率间隔又是多少？

5. 如果一台通用计算机的速度为平均每次复乘用时 40ns，每次复加用时 5ns，用它来计算 512 点的 DFT$[x(n)]$，问直接计算需要多少时间？用 FFT 运算需要多少时间？若做 128 点快速卷积运算，问所需最少时间是多少？

6. 设 $x(n)=[1,2,1,2,1]$，$h(n)=[1,2,2,1]$。

(1) 在时域求 $y(n)=x(n)*h(n)$；

(2) 用 FFT 流图法来求 $y(n)$，即求出 $H(k)$，$Y(k)=H(k)X(k)$，$y(n)=\mathrm{IFFT}[Y(k)]$。