离散傅立叶变换及快速傅立叶变换（DFT 与 FFT）

李军伟（郑州恩普特设备诊断工程有限公司技术支持部）

傅里叶变换概述

傅里叶变换（Transformée de Fourier）能将满足一定条件的某个函数表示成三角函数（正弦和/或余弦函数）或者它们的积分的线性组合。在不同的研究领域，傅里叶变换具有多种不同的变体形式，如连续傅里叶变换和离散傅里叶变换。最初傅里叶分析是作为热过程的解析分析的工具被提出的。傅里叶变换是一种线性的积分变换。因其基本思想首先由法国学者傅里叶系统地提出，所以以其名字来命名以示纪念。

一般情况下,若“傅立叶变换”一词的前面未加任何限定语，则指的是“连续傅里叶变换”。

连续傅里叶变换是将平方可积的函数 f(t) 表示成复指数函数的积分或级数形式。

$$F(\omega) = \mathcal{F}[f(t)] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt.$$

这是将频率域的函数 F(ω)表示为时间域的函数 f(t)的积分形式。

一般可称函数 f(t)为原函数,而称函数 F(ω)为傅里叶变换的像函数,原函数和像函数构成一个傅立叶变换对(transform pair)。

离散傅里叶变换

为了在科学计算和数字信号处理等领域使用计算机进行傅里叶变换，必须将函数 x(t) 定义在离散点而非连续域内，且须满足有限性或周期性条件。进而实现正、逆傅立叶变换运算。这样，在时域和频域中都只取有限个离散数据，这些离散数据分别构成周期性的离散时间函数和频率函数。离散傅立叶变换可以作为模拟信号傅立叶变换的一种特殊情况来实现。这种情况下，使用离散傅里叶变换。

模拟信号的离散傅立叶变换包括时域采样、时域裁断等过程。

时域采样

采样的基本问题是如何确定合理的采样间隔 Δt 以及采样长度 T，以保证采样所得的数字信号能真实地代表原来的连续信号 x(t) 。

一般说，采样频率 fm 越高，采点越密，所获得的数字信号越逼近原信号。然而，当采样长度 T 一定时， fm 越高，数据量 N = T Δt 越大，所需的计算机存储量和计算量就越大；反之，当采样频率降低到一定程度，就会丢失或歪曲原来信号的信息。

Shannon 采样定理给出了带限信号不丢失信息的最低采样频率为

$$f_s \geq 2f_m$$

式中:

fm——原信号中最高频率成分的频率

若不满足此采样定理，将会产生频率混淆现象。

解决频率混叠办法

频率混淆是由于采样以后采样信号频谱发生变化，而出现高、低频成分发生混淆的一种现象。

解决频率混叠的办法有：

（1） 提高采样频率以满足采样定理，一般工程中取f_s =（2.56~4）f_m

（2） 用低通滤波器滤掉不必要的高频成分以防频混产生，此时的低通滤波器也称为抗频混滤波器。

时域裁断

时域裁断是对采样信号进行截断，使其仅有有限个样本点的过程。截断是必然的，选择的采样长度不合适，裁断后真谱可能被歪曲，这种现象称为泄露。泄露是影响频谱分析精度的重要因素之一。

抑制泄露的方法

（1） 选择合适的采样长度 T。对于周期性随机过程或确定性的周期函数，应使 T 精确地等于信号基本周期的整数倍。这样不会产生新的频率分量，只使幅值的大小有所改变。

（2） 选择合适的窗函数。窗口宽度和窗口的形状决定了窗函数的特征，在条件允许的情况下，尽可能减少窗函数频谱主瓣的宽度；尽可能增大窗函数主瓣高度与旁瓣高度之比，压低旁瓣，特别是负旁瓣的大小，并使得旁瓣衰减得越快越好。

离散傅里叶变换式

离散傅里叶变换式：

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}} \qquad k = 0,1,2 \ldots N-1$$

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} x(k) e^{\frac{j2\pi nk}{N}} \qquad n = 0,1,2 \ldots N-1$$

工程信号分类

信号可分为两大类，即模拟信号和数字信号。模拟信号是随时间连续变化的，通常从传感器获得的信号都是模拟信号。数字信号是由离散的数字组成的，定期的观察值或模拟信号经过模／数(A／D)转换得到的一串数字都是数字信号，数字式传感器获得的信号也是数字信号。

数字信号处理就是以数字信号为基础所进行的各种分析处理。近 20 年来，其发展十分迅速。一方面由于计算机技术的飞跃发展，另一方面，FFT(Fast Fourier Transform)的问世及快速发展，为数字信号处理提供了强有力的手段和工具。目前，除了在通用计算机上研制的各种数字信号处理软件以外，还有多种专用硬件的数字处理机。由于其具有速度快，精度高、灵活、实时性强等优点，因此获得越来越广泛的应用。

快速傅里叶变换——FFT

在数字信号处理中，离散傅里叶变换(Discrete Fourier Transform, DFT)是常用的变换方法，它在各种数字信号处理系统中扮演着重要的角色。快速傅里叶变换〔Fast Fourier Transform, FFT〕并不是与离散傅里叶变换不同的另一种变换，而是为了减少 DFT 计算次数的一种快速有效的算法。它是根据离散傅氏变换的奇、偶、虚、实等特性，对离散傅立叶变换的算法进行改进获得的。它对傅氏变换的理论并没有新的发现，但是对于在计算机系统或者说数字系统中应用离散傅立叶变换，可以说是进了一大步。

快速傅里叶变换(Fast Fourier Transform，简称 FFT)方法是 1955 年由美国的库利—图基(J.W.Coo1ey—J.W.Tukey)首先提出来的。FFT 方法的诞生，被认为是信号

分析、数据处理技术的划时代的进步。

早在 FFT 方法提出之前，DFT 方法为离散信号的分析与处理从理论上提供了工具，但遗憾的是由于需要很长的计算时间很难得以实现。比如对采样点 N＝1000 的离散数据，DFT 运算工作量大约为 200 万次。FFT 是一种计算离散傅里叶变换的新方法，它大大地减少了运算次数，缩短了运算时间，比如对上述数据，FFT 仅需约 1.5 万次，使得实时分析处理成为可能。

设 x(n)为 N 项的复数序列，由 DFT 变换，任一 X(m)的计算都需要 N 次复数乘法和 N-1 次复数加法，而一次复数乘法等于四次实数乘法和两次实数加法，一次复数加法等于两次实数加法，即使把一次复数乘法和一次复数加法定义成一次“运算”（四次实数乘法和四次实数加法），那么求出 N 项复数序列的 X(m)，即 N 点 DFT 变换大约就需要 N^2 次运算。当 N=1024 点甚至更多的时候，需要 N2=1048576 次运算，在 FFT 中，利用 W^N 的周期性和对称性，把一个 N 项序列（设 N=2k,k 为正整数），分为两个 N/2 项的子序列，每个 N/2 点 DFT 变换需要 $(N/2)^2$ 次运算，再用 N 次运算把两个 N/2 点的 DFT 变换组合成一个 N 点的 DFT 变换。这样变换以后，总的运算次数就变成 $N+2(N/2)^2=N+N^2/2$。继续上面的例子，N=1024 时，总的运算次数就变成了 525312 次，节省了大约 50%的运算量。而如果我们将这种“一分为二”的思想不断进行下去，直到分成两两一组的 DFT 运算单元，那么 N 点的 DFT 变换就只需要 $N\log_2 N$ 次的运算，N 在 1024 点时，运算量仅有 10240 次，是先前的直接算法的 1%，点数越多，运算量的节约就越大，这就是 FFT 的优越性。

应用注意事项

傅立叶变换是对信号作数字频谱分析及实现数字滤波的基本方法。它在频谱分析、数字通信、语音信号分析、图象处理、雷达、声纳、地震、生物医学工程等各个领域都有着日益广泛的应用。

快速傅里叶变换的算法有多种，通过教科书及相关资料很容易得到其源程序。但是在实际的应用中要注意以下几点：

（1） 一般算法本身没有考虑各参数所代表的物理意义，在实际的表述中要有这种意识，比如在信号处理中，用到快速傅里叶变换，要注意点数的问题，得出的变换后的一组值是否除以了总的 FFT 变换点数。

（2） 傅里叶变换对采样的要求较高，合适的采样长度、采样频率及其窗函数的选择，可提高变换后频谱的精度和分辨率。

（3） FFT 是要求采样点数为 2 的次幂形式，采样时要尽量满足次条件，如果万一不满足，通过在数据后面补零的方法来实现。但补零后对变换

后的幅值将产生一定的误差。

（4） 在数字信号的分析处理应用中，DSP 比其它的处理器有绝对的优势，因为 DSP 有足够的能力运算 FFT 算法，这一处理器在一个指令周期能完成乘和累加的工作，因为复数运算要多次查表相乘才能实现。其二就是间接寻址，可以实现增/减 1 个变址量，方便各种查表方法。还具备反序间接寻址的能力。这就是单片机（51 系列，AVR，PIC 等等）或 ARM 处理器很少用来进行数字信号分析的原因。

（5） 在满足采样点数为 2 的整次幂及采样定理的情况下，整周期取样和非整周期取样的区别：整周期取样可以保证点数和频率的准确对位，非整周期取样不能保证这一点，可能产生一定误差。所以在条件允许的情况下，尽量整周期取样。

附 FFT 的一种 c 语言算法

参数：fr：采样点的实部，fi：采样点的虚部，n：采样点个数，flag：flag=0 表示求 Fourier 变换，flag=1 表示求逆 Fourier 变换

```
void FFT(double *fr, double *fi, int n, int flag)
{
 int mp,arg,q,cntr,p1,p2;
 int i,j,a,b,k,m;
 double sign,pr,pi,harm,x,y,t;
 double *ca,*sa;
 ca=(double *)calloc(n,sizeof(double));
 sa=(double *)calloc(n,sizeof(double));
 if(ca==NULL||sa==NULL) Error("calloc error in _FFT!");
 j=0;
 if(flag!=0)
  {
   sign=1.0;
  for(i=0;i<=n-1;i++)
  {
   fr[i]=fr[i]/n;
   fi[i]=fi[i]/n;
  }
  }
 else
```

```
 sign=-1.0;
for(i=0;i<=n-2;i++)
 {
 if(i<j)
   {
      t=fr[i];
      fr[i]=fr[j];
      fr[j]=t;
      t=fi[i];
      fi[i]=fi[j];
      fi[j]=t;
   }
   k=n/2;
   while(k<=j)
   {
         j-=k;
         k/=2;
    }
     j+=k;
  }
mp=0;
i=n;
while(i!=1)
{
   mp+=1;
    i/=2;
}
harm=2*M_PI/n;
for(i=0;i<=n-1;i++)
{
sa[i]=sign*sin(harm*i);
ca[i]=cos(harm*i);
}
a=2;
b=1;
```

```
        for(cntr=1;cntr<=mp;cntr++)
        {
        p1=n/a;
        p2=0;
        for(k=0;k<=b-1;k++)
        {
         i=k;
         while(i<n)
         {
         arg=i+b;
         if(k==0)
         {
         pr=fr[arg];
         pi=fi[arg];
         }
         Else
         {
         pr=fr[arg]*ca[p2]-fi[arg]*sa[p2];
         pi=fr[arg]*sa[p2]+fi[arg]*ca[p2];
         }
         fr[arg]=fr[i]-pr;
         fi[arg]=fi[i]-pi;
         fr[i]+=pr;
         fi[i]+=pi;
         i+=a;
         }
         p2+=p1;
         }
         a*=2;
         b*=2;
         }
  free(ca);
   free(sa);
}
```