快速傅氏变换，是离散傅氏变换的快速算法，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。

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 N2 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 WN 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+N2/2。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 Nlog2N 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.
傅里叶变换（Transformée de Fourier）是一种积分变换。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。

应用
傅里叶变换在物理学、数论、组合数学、信号处理、概率论、统计学、密码学、声学、光学、海洋学、结构动力学等领域都有着广泛的应用（例如在信号处理中，傅里叶变换的典型用途是将信号分解成幅值分量和频率分量）。

概要介绍
傅里叶变换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（参见：林家翘、西格尔著《自然科学中确定性问题的应用数学》，科学出版社，北京。原版书名为 C. C. Lin & L. A. Segel, Mathematics Applied to Deterministic Problems in the Natural Sciences, Macmillan Inc., New York, 1974）。
傅里叶变换属于谐波分析。
傅里叶变换的逆变换容易求出,而且形式与正变换非常类似;
正弦基函数是微分运算的本征函数,从而使得线性微分方程的求解可以转化为常系数的代数方程的求解.在线性时不变的物理系统内,频率是个不变的性质,从而系统对于复杂激励的响应可以通过组合其对不同频率正弦信号的响应来获取;
卷积定理指出:傅里叶变换可以化复杂的卷积运算为简单的乘积运算,从而提供了计算卷积的一种简单手段;
离散形式的傅里叶变换可以利用数字计算机快速的算出(其算法称为快速傅里叶变换算法(FFT)).

快速傅立叶变换 FFT 的 C 语言程序
2008 年 08 月 29 日 星期五 22:59
```
// 入口参数：
// I：I = 0, 傅立叶变换；I = 1, 逆傅立叶变换
```

```
// il: il = 0,不计算傅立叶变换或逆变换模和幅角；il = 1,计算模和幅角
// n: 输入的点数，为偶数，一般为32，64，128，...,1024等
// k: 满足n=2^k(k>0),实质上k是n个采样数据可以分解为偶次幂和奇次幂的次数
// pr[]: l=0时，存放N点采样数据的实部
// l=1时，存放傅立叶变换的N个实部
// pi[]: l=0时，存放N点采样数据的虚部
// l=1时，存放傅立叶变换的N个虚部
//
// 出口参数：
// fr[]: l=0，返回傅立叶变换的实部
// l=1，返回逆傅立叶变换的实部
// fi[]: l=0，返回傅立叶变换的虚部
// l=1，返回逆傅立叶变换的虚部
// pr[]: il = 1,l = 0 时，返回傅立叶变换的模
// il = 1,l = 1 时，返回逆傅立叶变换的模
// pi[]: il = 1,l = 0 时，返回傅立叶变换的辐角
// il = 1,l = 1 时，返回逆傅立叶变换的辐角


void kbfft(double *pr,double *pi,int n,int k,double *fr,double *fi,int l,int il)
{
    int it,m,is,i,j,nv,l0;
    double p,q,s,vr,vi,poddr,poddi;

    //排序
    for (it=0; it<=n-1; it++)
      { m=it; is=0;
        for (i=0; i<=k-1; i++)
          { j=m/2; is=2*is+(m-2*j); m=j;
            fr[it]=pr[is]; fi[it]=pi[is];
          }
      }

    //蝶形运算
    pr[0]=1.0; pi[0]=0.0;
    p=6.283185306/(1.0*n);
    pr[1]=cos(p); pi[1]=-sin(p);
    if (l!=0) pi[1]=-pi[1];
    for (i=2; i<=n-1; i++)
      { p=pr[i-1]*pr[1]; q=pi[i-1]*pi[1];
        s=(pr[i-1]+pi[i-1])*(pr[1]+pi[1]);
        pr[i]=p-q; pi[i]=s-p-q;
      }
    for (it=0; it<=n-2; it=it+2)
      { vr=fr[it]; vi=fi[it];
```

```
        fr[it]=vr+fr[it+1]; fi[it]=vi+fi[it+1];
        fr[it+1]=vr-fr[it+1]; fi[it+1]=vi-fi[it+1];
      }
    m=n/2; nv=2;
    for (l0=k-2; l0>=0; l0--)
      { m=m/2; nv=2*nv;
        for (it=0; it<=(m-1)*nv; it=it+nv)
          for (j=0; j<=(nv/2)-1; j++)
            { p=pr[m*j]*fr[it+j+nv/2];
              q=pi[m*j]*fi[it+j+nv/2];
              s=pr[m*j]+pi[m*j];
              s=s*(fr[it+j+nv/2]+fi[it+j+nv/2]);
              poddr=p-q; poddi=s-p-q;
              fr[it+j+nv/2]=fr[it+j]-poddr;
              fi[it+j+nv/2]=fi[it+j]-poddi;
              fr[it+j]=fr[it+j]+poddr;
              fi[it+j]=fi[it+j]+poddi;
            }
      }
    if (l!=0)
      for (i=0; i<=n-1; i++)
        { fr[i]=fr[i]/(1.0*n);
          fi[i]=fi[i]/(1.0*n);
        }
    if (il!=0)
      for (i=0; i<=n-1; i++)
        { pr[i]=sqrt(fr[i]*fr[i]+fi[i]*fi[i]);
          pr[i]=(pr[i]/(n/2)); //各次谐波幅值，其中pr[1]为基波幅值
          if (fabs(fr[i])<0.000001*fabs(fi[i]))//fabs()是取绝对值函数,浮点型的0 在
内存中并不是严格等于0,可以认为当一个浮点数

离原点足够近时,也就是f>0.00001 && f<-0.00001,认为f是0
            { if ((fi[i]*fr[i])>0) pi[i]=90.0;
              else pi[i]=-90.0;
            }
          else
            pi[i]=atan(fi[i]/fr[i])*360.0/6.283185306;
        }
    return;
}
```

快速傅立叶变换FFT的C语言程序

2008年08月29日 星期五 22:59

```
// 入口参数：
// l: l = 0, 傅立叶变换; l = 1, 逆傅立叶变换
```

```
// il: il = 0,不计算傅立叶变换或逆变换模和幅角；il = 1,计算模和幅角
// n: 输入的点数，为偶数，一般为32，64，128，...,1024等
// k: 满足n=2^k(k>0),实质上k是n个采样数据可以分解为偶次幂和奇次幂的次数
// pr[]: l=0时，存放N点采样数据的实部
// l=1时，存放傅立叶变换的N个实部
// pi[]: l=0时，存放N点采样数据的虚部
// l=1时，存放傅立叶变换的N个虚部
//
// 出口参数：
// fr[]: l=0，返回傅立叶变换的实部
// l=1，返回逆傅立叶变换的实部
// fi[]: l=0，返回傅立叶变换的虚部
// l=1，返回逆傅立叶变换的虚部
// pr[]: il = 1,l = 0 时，返回傅立叶变换的模
// il = 1,l = 1 时，返回逆傅立叶变换的模
// pi[]: il = 1,l = 0 时，返回傅立叶变换的辐角
// il = 1,l = 1 时，返回逆傅立叶变换的辐角


void kbfft(double *pr,double *pi,int n,int k,double *fr,double *fi,int l,int il)
{
    int it,m,is,i,j,nv,l0;
    double p,q,s,vr,vi,poddr,poddi;

    //排序
    for (it=0; it<=n-1; it++)
      { m=it; is=0;
        for (i=0; i<=k-1; i++)
          { j=m/2; is=2*is+(m-2*j); m=j;
            fr[it]=pr[is]; fi[it]=pi[is];
          }
      }

    //蝶形运算
    pr[0]=1.0; pi[0]=0.0;
    p=6.283185306/(1.0*n);
    pr[1]=cos(p); pi[1]=-sin(p);
    if (l!=0) pi[1]=-pi[1];
    for (i=2; i<=n-1; i++)
      { p=pr[i-1]*pr[1]; q=pi[i-1]*pi[1];
        s=(pr[i-1]+pi[i-1])*(pr[1]+pi[1]);
        pr[i]=p-q; pi[i]=s-p-q;
      }
    for (it=0; it<=n-2; it=it+2)
      { vr=fr[it]; vi=fi[it];
```

```
        fr[it]=vr+fr[it+1]; fi[it]=vi+fi[it+1];
        fr[it+1]=vr-fr[it+1]; fi[it+1]=vi-fi[it+1];
      }
    m=n/2; nv=2;
    for (l0=k-2; l0>=0; l0--)
      { m=m/2; nv=2*nv;
        for (it=0; it<=(m-1)*nv; it=it+nv)
          for (j=0; j<=(nv/2)-1; j++)
            { p=pr[m*j]*fr[it+j+nv/2];
              q=pi[m*j]*fi[it+j+nv/2];
              s=pr[m*j]+pi[m*j];
              s=s*(fr[it+j+nv/2]+fi[it+j+nv/2]);
              poddr=p-q; poddi=s-p-q;
              fr[it+j+nv/2]=fr[it+j]-poddr;
              fi[it+j+nv/2]=fi[it+j]-poddi;
              fr[it+j]=fr[it+j]+poddr;
              fi[it+j]=fi[it+j]+poddi;
            }
      }
    if (l!=0)
      for (i=0; i<=n-1; i++)
        { fr[i]=fr[i]/(1.0*n);
          fi[i]=fi[i]/(1.0*n);
        }
    if (il!=0)
      for (i=0; i<=n-1; i++)
        { pr[i]=sqrt(fr[i]*fr[i]+fi[i]*fi[i]);
          pr[i]=(pr[i]/(n/2)); //各次谐波幅值，其中pr[1]为基波幅值
          if (fabs(fr[i])<0.000001*fabs(fi[i]))//fabs()是取绝对值函数,浮点型的0 在
内存中并不是严格等于0,可以认为当一个浮点数

离原点足够近时,也就是f>0.00001 && f<-0.00001,认为f是0
            { if ((fi[i]*fr[i])>0) pi[i]=90.0;
              else pi[i]=-90.0;
            }
          else
            pi[i]=atan(fi[i]/fr[i])*360.0/6.283185306;
        }
    return;
}
```