```
   Help
/*  To calculate the discrete Fourier Transform using FFT
    algorithm

N : (input) Number of points, which must be a power of 2
CG : (input/output) Array of length 2*N containing the data
     points
After execution it will contain the Fourier transform of CG
CG is complex and hence the array should
contain the real and imaginary parts.
IFLG : (input) Flag to decide whether to calculate forward
    or inverse
transform. If IFLG>=0 then Fourier transform is calculated
IF IFLG<0 then inverse Fourier transform is calculated

Error status is returned by the value of the function FFT.
0 value implies successful execution
611 implies that N<2, no calculations are done
631 implies that N is not a power of 2, in this case
contents of CG will be destroyed but will not
contain the Fourier transform.

Required functions : None
*/

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <time.h>

#include "pnl/pnl_complex.h"
#include "nrutil.h"
#include "rncf.h"
#include "pnl/pnl_mathtools.h"
```

```
int fft(long n, double **cg, int iflg)
{
  int i,j,m,j0,k0,iw,i1,jr;
  double r,ct[2],th,cwj[2],cwf[2];

  //double** cg = dmatrix(0, n - 1, 0, 2);

  if(n<2) return 611;

  /*  Bit reversal */
  j=0;
  for(i=0; i<n; ++i) {
    if(j>i) {
      /*  exchange CG[I] with CG[J] */
      r=cg[i][0]; cg[i][0]=cg[j][0]; cg[j][0]=r;
      r=cg[i][1]; cg[i][1]=cg[j][1]; cg[j][1]=r;
    }
    m=n/2;
    while(m>=1 && j>=m) {j=j-m; m=m/2;}
    /*  J-1 is the bit reverse of I */
    j=j+m;
  }

  j0=1; k0=n/2;
  th=M_PI/k0;
  if(iflg>=0) iw=1;
  else iw=-1;
  cwf[0]=-1; cwf[1]=0.0;

  /*  Main loop for FFT executed Log_2(N) times */
  do {
    cwj[0]=1.0; cwj[1]=0.0;
    /*  Inner loop over all elements */
    for(jr=0; jr<j0; ++jr) {
      for(i=jr; i<n; i += 2*j0) {
        i1=i+j0;
        ct[0]=cg[i1][0]*cwj[0]-cg[i1][1]*cwj[1];
        ct[1]=cg[i1][0]*cwj[1]+cg[i1][1]*cwj[0];
        cg[i1][0]=cg[i][0]-ct[0]; cg[i1][1]=cg[i][1]-ct[1];
        cg[i][0]=cg[i][0]+ct[0]; cg[i][1]=cg[i][1]+ct[1];
      }
      r=cwj[0]*cwf[0]-cwj[1]*cwf[1];
      cwj[1]=cwj[0]*cwf[1]+cwj[1]*cwf[0];
      cwj[0]=r;
    }

    j0=2*j0; k0=k0/2;
    cwf[0]=cos(iw*k0*th); cwf[1]=sin(iw*k0*th);
  } while(j0<n);
  if(j0==n) return 0;
  else return 631;
}


int dft(long n, double cg[][2], double cf[][2], int iflg)

{
  int i,iw,j;
  double cs[2],cw;

  if(n<2) return 611;

  if(iflg>=0) iw=1;
  else iw=-1;

  /*  Loop for Fourier transform */
  for(i=0; i<n; ++i) {
    cw=2.0*iw*i*M_PI/n;
    cs[0]=0.0; cs[1]=0.0;
    for(j=0; j<n; ++j) {
      cs[0]=cs[0]+cg[j][0]*cos(j*cw)-cg[j][1]*sin(j*cw);
      cs[1]=cs[1]+cg[j][1]*cos(j*cw)+cg[j][0]*sin(j*cw);
    }
    cf[i][0]=cs[0]; cf[i][1]=cs[1];
  }
  return 0;
}



void TableIFT(int choice, int model, double rf, double dt,
    long N,
              double dx, double aa, double parameters[],
              double inv[], double logk[])
{

  int j;
  dcomplex term1;
  double wj;
  dcomplex ft;
  //double cg[N-1][2];
  double eta;
  double** cg = dmatrix(0, N - 1, 0, 2);

  eta = 2 * M_PI / (N * dx);

  for (j=0; j<=N-1; j++)
    {
      if( j == 0){
        wj = 0.5;}
      else if (j == N - 1) {
        wj = 0.5;}
      else {
        wj = 1;}

      if( choice == 1){
        //        'construct the density
        ft = cfrn(model, rf, dt, Complex(j * eta, 0), para
    meters);}
      if( choice == 2){
        //'construct the call option price
        ft = cfrncall(model, rf, dt, Complex(j * eta, 0),
    aa, parameters);
      }
      if( choice == 3){
        //'construct the put option price
        ft = cfrncall(model, rf, dt, Complex(j * eta, 0), -
    aa, parameters);}
      if( choice == 4){
        //'construct the cdf
        ft = cfCDF(model, dt, Complex(j * eta, 0), aa, para
    meters);}
      if( choice == 5){
        //'construct the Levy density without drift adjustm
    ent
        ft = cfLevy(model, dt, Complex(j * eta, 0), para
    meters);}
      if( choice == 6){
        //'construct the Levy density with a shifting
        ft = cfrnshifted(model,aa,rf,dt,Complex(j * eta, 0)
    , parameters);}


      term1 = Cexp(Complex(0, M_PI * j));
      ft = Cmul(term1, ft);
      cg[j][0] = wj * Creal(ft) * eta / M_PI;
      cg[j][1] = wj * Cimag(ft) * eta / M_PI;

      ///    printf("cg%.9f cg2%.9f eta%.9f {n",cg[j][0],
    Creal(ft) ,eta );
    }

  //'FFT inversion
  j = fft(N, cg, -1);

  if( choice == 1){
    for(j=0; j<=N-1; j++)
      {logk[j]=-N*dx/2+j*dx;
        inv[j]=cg[j][0];      }
  }

  if( choice == 2){
    for(j=0; j<=N-1; j++)
      {logk[j]=-N*dx/2+j*dx;
        inv[j]=exp(-aa*logk[j])*cg[j][0];}
  }

  if( choice == 3){
    for(j=0; j<=N-1; j++)
      {logk[j]=-N*dx/2+j*dx;
        inv[j]=exp(aa*logk[j])*cg[j][0];}
  }
```

```
  if( choice == 4){
    for(j=0; j<=N-1; j++)
      {logk[j]=-N*dx/2+j*dx;
        inv[j]=1-exp(-aa*logk[j])*cg[j][0];}
  }

  if( choice == 5){
    for(j=0; j<=N-1; j++)
      {logk[j]=-N*dx/2+j*dx;
        inv[j]=cg[j][0];}
  }

  if( choice == 5){
    for(j=0; j<=N-1; j++)
      {logk[j]=-N*dx/2+j*dx;
        inv[j]=cg[j][0];}
  }


  free_dmatrix(cg, 0, N - 1, 0,2);

  //  '  If j = 0 Then MsgBox ("Successful execution of th
    e FFT inversion")
  //  '  If j = 611 Then MsgBox ("The number of points N ha
    s to be greater than 2")
  //  '  If j = 631 Then MsgBox ("The number of points N ha
    s to be a power of 2")
}



//%%find the value umax for which cf(umax)<10^-10
double find_umax_cf(int model,double  rf,double dt, double
    aa, double parameters[])
{
  double epsilon_cf,abs_cf,step,umax;
  dcomplex cf;

  step = 1.5;
  umax = 5.0; //   ' Inizializza la variabile.
  cf = cfrn(model, rf, dt, Complex(umax, 0), parameters);
  abs_cf = sqrt(Creal(cf)*Creal(cf) + Cimag(cf)*Cimag(cf));
  epsilon_cf = POW(10, -10);

  do{
    umax = umax * step; // Incrementa step
    cf = cfrn(model, rf, dt, Complex(umax, 0), parameters);
    abs_cf = sqrt(Creal(cf)*Creal(cf) + Cimag(cf)*Cimag(cf)
    ); //compute abs error
  } while (abs_cf > epsilon_cf); //abs_cf > epsilon_cf


  return  (umax + umax / step) / 2;

}

void TableIFRT(int choice, int model, double rf,double  dt,
    long N,
               double b, double aa, double parameters[],
               double inv[], double logk[])
{

  int j;
  double wj;
  dcomplex term1, ft,z_j_after_N,y_j,z_j,cgyz ;
  //double cg[N-1][2];
  double eta;
  double** cg = dmatrix(0, N - 1, 0, 2);
  double** cg_yz = dmatrix(0, 2*N - 1, 0, 2);
  double** cg_out = dmatrix(0, N - 1, 0, 2);
  double** y = dmatrix(0, 2*N - 1, 0, 2);
  double** z = dmatrix(0, 2*N - 1, 0, 2);



  double alpha, dx,umax;


  //'%find umax
  umax = find_umax_cf(model, rf, dt, aa, parameters);
  eta = umax / N;
  dx = 2 * b / N;
```

```
alpha = eta * dx / (2 * M_PI);




for (j=0; j<=N-1; j++)
  {
    if( j == 0){
      wj = 0.5;}
    else if (j == N - 1) {
      wj = 0.5;}
    else {
      wj = 1;}

    if( choice == 1){
      //        'construct the density
      ft = cfrn(model, rf, dt, Complex(j * eta, 0), para
  meters);}
    if( choice == 2){
      //'construct the call option price
      ft = cfrncall(model, rf, dt, Complex(j * eta, 0),
  aa, parameters);}
    if( choice == 3){
      //'construct the put option price
      ft = cfrncall(model, rf, dt, Complex(j * eta, 0), -
  aa, parameters);}
    if( choice == 4){
      //'construct the cdf
      ft = cfCDF(model, dt, Complex(j * eta, 0), aa, para
  meters);}
    if( choice == 5){
      //'construct the Levy density without drift adjustm
  ent
      ft = cfLevy(model, dt, Complex(j * eta, 0), para
  meters);}
    if( choice == 6){
      //'construct the Levy density with a shift
      ft = cfrnshifted(model,aa,rf,dt,Complex(j * eta, 0)
  , parameters);}
    if( choice == 7){
```

```
   //'construct the Levy density with standardization
   ft = cfrnstandardized(model,rf, dt, Complex(j * et
 a, 0), parameters);}

  term1 = Cexp(Complex(0,b*eta* j));
  ft = Cmul(term1, ft);
  cg[j][0] = wj * Creal(ft) * eta / M_PI;
  cg[j][1] = wj * Cimag(ft) * eta / M_PI;


  ///    printf("cg%.9f cg2%.9f eta%.9f {n",cg[j][0],
 Creal(ft) ,eta );
  y_j = Cmul(Complex(cg[j][0], cg[j][1]), (Complex(cos(
 -j * alpha * j * M_PI), sin(-j * alpha * j * M_PI))));
  y[j][0] = Creal(y_j);
  y[j][1] = Cimag(y_j);
  y[j + N][ 0] = 0;
  y[j + N][ 1] = 0;



  z_j = Complex(cos(j * alpha * j * M_PI), sin(j * alp
 ha * j * M_PI));
  z[j][0] = Creal(z_j);
  z[j][1] = Cimag(z_j);

  z_j_after_N = Complex(cos((N - j) * alpha * (N - j) *
  M_PI), sin((N - j) * alpha * (N - j) * M_PI)); //Cexp(
 Complex(0, (N - j) * (N - j) * M_PI * alpha));
  z[j+N][0] = Creal(z_j_after_N);
  z[j+N][1] = Cimag(z_j_after_N);
 }

fft(2 * N, y, -1);
fft(2 * N, z, -1);

for( j = 0; j<= 2 * N - 1;j++)
 {

  cgyz = Cmul(Complex(y[j][ 0], y[j][ 1]), Complex(z[j]
 [ 0], z[j][ 1]));
```

```
    cg_yz[j][0] = Creal(cgyz);
    cg_yz[j][1] = Cimag(cgyz);
    ///  printf("cgyz%d %.9f %.9f %.9f %.9f{n",j, Creal(
  cgyz),Cimag(cgyz),y[j][ 1], z[j][ 1]);

  }

//'FFT inversion
j = fft(2 * N, cg_yz, 1);


for( j = 0; j<= N - 1;j++)
  {
    cgyz = RCmul(1.0/(2.0*N),Cmul(Complex(cg_yz[j][0],
  cg_yz[j][1]), Cexp(Complex(0, -j * alpha * j * M_PI))));


    if( choice == 1){
      logk[j]=-b+j*dx;
      inv[j]=Creal(cgyz);
    }

    if( choice == 2){
      logk[j]=-b+j*dx;
      inv[j]=exp(-aa*logk[j])*Creal(cgyz);
    }

    if( choice == 3){
      logk[j]=-b+j*dx;
      inv[j]=exp(aa*logk[j])*Creal(cgyz);
    }

    if( choice == 4){
      logk[j]=-b+j*dx;
      inv[j]=1-exp(-aa*logk[j])*Creal(cgyz);
    }

    if( choice == 5){
      logk[j]=-b+j*dx;
      inv[j]=cg[j][0];
    }
```

```
      if( choice == 6){
        logk[j]=-b+j*dx;
        inv[j]=cg[j][0];
      }

      if( choice == 7){
        logk[j]=-b+j*dx;
        inv[j]=Creal(cgyz);
      }
    }

  free_dmatrix(cg, 0, N - 1, 0,2);
  free_dmatrix(cg_yz, 0, 2*N - 1, 0,2);
  free_dmatrix(cg_out, 0, N - 1, 0,2);
  free_dmatrix(z, 0, 2*N - 1, 0,2);
  free_dmatrix(y, 0, 2*N - 1, 0,2);

}


/*
   void TableDensity(int choice, int model, double rf,
    double  dt, long N,
   double b, double parameters[],
   double dens[], double logk[])
   {

   int j,j_y, j_z;
   double dx = 2 * b / N;






   for (j=0; j<=N-1; j++)
   {
   logk[j]=-b+j*dx;
   dens[j]=w0Levy(model, -b+j*dx, dt,  rf, parameters);
   }
   }*/
#endif //PremiaCurrentVersion
```

# References