Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

#include "fft.h"
#include "numerics.h"
#include "levy_fd.h"


Grid::Grid(const double dAl, const double dAr, const int dN
    )
  :
  Al(dAl), Ar(dAr), N(dN)
{
  dx = (Ar-Al)/(N-1);
}

double init_cond(const double x, const double S0,
                 const double K, const int product)
{
  double S = S0*exp(x);

  switch(product){
  case 1: return (S-K > 0) ? (S-K) : 0; // Call
  case 2: return (K-S > 0) ? (K-S) : 0; // Put
  case 3: return  S-K; // forward
  default: myerror("Invalid product number");
  }
  /* just to avoid a warning */
  return 0;
}
double bound_cond(const double x, const double S0, const
    double K,const double rebate,
```

```
                  const double ttm, const double r,
                  const int product, const int product_type
    )
{
  switch(product_type){
  case 1: return init_cond(x+r*ttm,S0,K,product); // Europe
    an
  case 2: return (x>0) ? rebate *exp(r*ttm) : init_cond(x+
    r*ttm,S0,K,product); // Up-and-Out
  case 3: return (x>0) ? init_cond(x+r*ttm,S0,K,product):
    rebate*exp(r*ttm); // Down-and-Out
  case 4: return rebate*exp(r*ttm); // Double barrier
  default: myerror("Invalid option type number");
  }
  /* just to avoid a warning */
  return 0;
}

vector<double> price2(int am,const Levy_measure & measure,
    int product,
                      const int product_type, double r,
    double divid,
                      double S0, double K,double rebate,
    double Al,
                      double Ar, int Nspace, double T, int
    Ntime,
                      double & price0, double & delta0)
{

  double dt = T/Ntime;

  if((Al > 0) || (Ar < 0)) myerror("Error: (Al > 0)  or  (
    Ar < 0)!");
  const double dx = (Ar-Al)/(Nspace-1);
  if(dx <= 0) myerror("Error: dx = 0!");

  if(dt>dx/(fabs(measure.alpha)+measure.lambda*dx))
    cout << "Stability Condition is not satisfied!" << end
    l <<
    "Time Discretization Step is changed" << endl ;
```

```
while(dt>dx/(fabs(measure.alpha)+measure.lambda*dx)){
  Ntime += 10;
  dt=T/Ntime;
}

const int Kmin = measure.Kmin;
const int Kmax = measure.Kmax;

const Grid grid(Al,Ar,Nspace);


vector<double> u(Nspace), v(Nspace);

// condition initiale
for(int i=0;i<Nspace;i++)
  {
    u[i] = init_cond(grid.x(i),S0,K,product);
  }


//some useful coefficients
double ss = measure.sigmadiff_squared;
double dxx = dx*dx;
double aux = ss/2-(r-divid);

double a,b,c;
/*matrix coefficients of the implicit part*/
if(dx < ss/fabs(aux))
  {
    a = ss/dxx;
    b = ss/2./dxx - aux/2./dx;
    c = ss/2./dxx + aux/2./dx;
  }
else if(aux<0)
  {
    a = ss/dxx - aux/dx;
    b = ss/2./dxx - aux/dx;
    c = ss/2./dxx;
  }
else
  {
```

```
    a = ss/dxx + aux/dx;
    b = ss/2./dxx;
    c = ss/2./dxx + aux/dx;
  }


const int N = Nspace+Kmax-Kmin; //number of non-zero val
  ues of u involved in computation
//zero-padding to obtain NN = N + Nz = 2^p
int p = 1;
int NN = 2; //size of auxiliary vectors
while(NN<N){
  p++;
  NN = 2*NN;
}
const int Nz = NN - N; // number of extra zeros

double* mu = new double [NN];
double* mu_img = new double [NN];
double* uaux = new double [NN];
double* uaux_img = new double [NN];
double* somme = new double [NN];
double* somme_img = new double [NN];

/*computation of the Fourier transform of nu*/
for(int i=0; i<Kmax-Kmin+1; i++)
  {
    mu[i] = (*measure.nu_array)[Kmax-Kmin-i];
    mu_img[i] = 0;
  }
for(int i=Kmax-Kmin+1; i<NN; i++)
  {
    mu[i] = 0;
    mu_img[i] = 0;
  }
fft1d(mu,mu_img,NN,-1);

double ttm; //time to maturity
for(int n=0; n<Ntime; n++) //time iterations
  {
    ttm = n*dt;
```

```
  /*calculation of the discretized integral using FFT*/
  for(int i=0; i<Nspace-1; i++){
    if((Kmax+1+i<0)||(Kmax+1+i>=Nspace)){
      uaux[i] = bound_cond(grid.x(Kmax+1+i),S0,K,rebate
,ttm,r-divid,product,product_type);

    }
    else uaux[i] = u[Kmax+1+i];
    uaux_img[i]=0;
  }
  for(int i=Nspace-1; i<Nspace+Nz-1; i++)
    {
      uaux[i] = 0; //zero-padding
      uaux_img[i] = 0;
    }
  for(int i=Nspace+Nz-1; i<NN; i++){
    if((Kmin-Nspace-Nz+1+i<0)||(Kmin-Nspace-Nz+1+i>=Ns
pace)){
      uaux[i] = bound_cond(grid.x(Kmin-Nspace-Nz+1+i),
S0,
                           K,rebate,ttm,r-divid,produc
t,product_type);
    }
    else uaux[i] = u[Kmin-Nspace-Nz+1+i];
    uaux_img[i]=0;
  }

  fft1d(uaux,uaux_img,NN,-1);

  for(int i=0; i<NN; i++)
    {
      somme[i] = mu[i]*uaux[i] - mu_img[i]*uaux_img[i];
      somme_img[i] = mu_img[i]*uaux[i] + mu[i]*uaux_img
[i];
    }
  fft1d(somme,somme_img,NN,1);


  /*computation of the right-hand side vector v */
```

```
  if(measure.alpha < 0){ //backward discretization of
the first order derivative
    v[0] = u[0] + dt*(somme[NN-1]-measure.alpha*(u[1]-
u[0])/dx
                      -measure.lambda*u[0]) +
    c*dt*bound_cond(grid.x(-1),S0,K,rebate,ttm+dt,r-div
id,product,product_type);

    v[Nspace-1] = u[Nspace-1] +
    dt*(somme[Nspace-2]-measure.alpha*(bound_cond(grid.
x(Nspace),S0,K,rebate,ttm,r-divid,product,product_type)
                                       -u[Nspace-1])/dx
 -measure.lambda*u[Nspace-1])
    + b*dt*bound_cond(grid.x(Nspace),S0,K,rebate,ttm+dt
,r-divid,product,product_type);

    for(int i=1; i<Nspace-1; i++)
      v[i] = u[i] + dt*(somme[i-1]-measure.alpha*(u[i+1
]-u[i])/dx -measure.lambda*u[i]);
  }
  else{ //forward discretization of the first order de
rivative
    v[0] = u[0] + dt*(somme[NN-1]-measure.alpha*(u[0]-
                                                 bound_
cond(grid.x(-1),S0,K,rebate,ttm,r-divid,product,product_type)
)/dx
                      -measure.lambda*u[0])+
    c*dt*bound_cond(grid.x(-1),S0,K,rebate,ttm+dt,r-div
id,product,product_type);

    for(int i=1; i<Nspace-1; i++)
      v[i] = u[i] + dt*(somme[i-1]-measure.alpha*(u[i]-
u[i-1])/dx -measure.lambda*u[i]);
    v[Nspace-1] = u[Nspace-1] + dt*(somme[Nspace-2]-
                                  measure.alpha*(u[Ns
pace-1]
                                                -u[
Nspace-2])/dx -measure.lambda*u[Nspace-1])
    + b*dt*bound_cond(grid.x(Nspace),S0,K,rebate,ttm+dt
,r-divid,product,product_type);
  }
```

```
    /*computation of de u^(n+1) using LU-decomposition
  realized in the routine progonka*/
    u = progonka(Nspace,-dt*c,1+dt*a,-dt*b,v);
    if(am)
  for(int i=Nspace-1;i>=0;i--)
  u[i]=MAX(u[i],exp(r*(ttm+dt))*init_cond(grid.x(i),S0,
  K,product));
  }//end of time iterations


  double actu = exp(-r*T);
  int N0 = (int) floor(-Al/dx);
  double Sl = S0*exp(grid.x(N0-1));
  double Sm = S0*exp(grid.x(N0));
  double Sr = S0*exp(grid.x(N0+1));

  // S0 is between Sm and Sr
  double pricel = actu*u[N0-1];
  double pricem = actu*u[N0];
  double pricer = actu*u[N0+1];

  //quadratic interpolation
  double A = pricel;
  double B = (pricem-pricel)/(Sm-Sl);
  double C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);
  price0 = A+B*(S0-Sl)+C*(S0-Sl)*(S0-Sm);
  delta0 = B + C*(2*S0-Sl-Sm);

  delete [] mu;
  delete [] mu_img ;
  delete [] uaux;
  delete [] uaux_img;
  delete [] somme;
  delete [] somme_img;

  return u;
}//end price2


vector<double> price2c(int am,const Levy_measure & measure,
```

```
     int product,
                         int product_type, double r,double
    divid,
                         double S0, double K,double rebate,
                         double Al, double Ar, const int Ns
    pace, double T,
                         int Ntime, double & price0, double &
     delta0)
{

  double dt = T/Ntime;

  if((Al > 0) || (Ar < 0)) myerror("Error: (Al > 0)  or  (
    Ar < 0)!");
  const double dx = (Ar-Al)/(Nspace);
  if(dx <= 0) myerror("Error: dx = 0!");

  if(dt>dx/(fabs(measure.alpha)+measure.lambda*dx))
    cout << "Stability Condition is not satisfied!" << end
    l <<
    "Time Discretization Step is changed" << endl ;

  while(dt>dx/(fabs(measure.alpha)+measure.lambda*dx)){
    Ntime += 10;
    dt=T/Ntime;
  }

  const int Kmin = measure.Kmin;
  const int Kmax = measure.Kmax;

  const Grid grid(Al,Ar,Nspace);


  vector<double> u(Nspace), v(Nspace);

  // initial condition
  for(int i=0;i<Nspace;i++)
    {
      u[i] = init_cond(grid.x(i),S0,K,product);
    }
```

```
//some useful coefficients
double ss = measure.sigmadiff_squared;
double dxx = dx*dx;
double aux = ss/2-(r-divid);

double a,b,c;
if(dx < ss/fabs(aux))
  {
    a = ss/dxx;
    b = ss/2./dxx - aux/2./dx;
    c = ss/2./dxx + aux/2./dx;
  }
else if(aux<0)
  {
    a = ss/dxx - aux/dx;
    b = ss/2./dxx - aux/dx;
    c = ss/2./dxx;
  }
else
  {
    a = ss/dxx + aux/dx;
    b = ss/2./dxx;
    c = ss/2./dxx + aux/dx;
  }

const int N = Nspace+Kmax-Kmin; //number of non-zero val
  ues of u involved in computation
//zero-padding to obtain NN = N + Nz = 2^p
int p = 1;
int NN = 2; //size of auxiliary vectors
while(NN<N){
  p++;
  NN = 2*NN;
}
const int Nz = NN - N; // number of extra zeros

double* mu = new double [NN];
double* mu_img = new double [NN];
double* uaux = new double [NN];
double* uaux_img = new double [NN];
```

```
double* somme = new double [NN];
double* somme_img = new double [NN];

/*computation of the Fourier transform of nu*/
for(int i=0; i<Kmax-Kmin+1; i++)
  {
    mu[i] = (*measure.nu_array)[Kmax-Kmin-i];
    mu_img[i] = 0;
  }
for(int i=Kmax-Kmin+1; i<NN; i++)
  {
    mu[i] = 0;
    mu_img[i] = 0;
  }
fft1d(mu,mu_img,NN,-1);

double ttm; //time to maturity
for(int n=0; n<Ntime; n++) //time iterations
  {
    ttm = n*dt;

    /*calculation of the discretized integral using FFT*/
    for(int i=0; i<Nspace-1; i++){
      if((Kmax+1+i<0)||(Kmax+1+i>=Nspace)){
        uaux[i] = bound_cond(grid.x(Kmax+1+i),S0,K,rebate
  ,ttm,r-divid,product,product_type);

      }
      else uaux[i] = u[Kmax+1+i];
      uaux_img[i]=0;
    }
    for(int i=Nspace-1; i<Nspace+Nz-1; i++)
      {
        uaux[i] = 0; //zero-padding
        uaux_img[i] = 0;
      }
    for(int i=Nspace+Nz-1; i<NN; i++){
      if((Kmin-Nspace-Nz+1+i<0)||(Kmin-Nspace-Nz+1+i>=Ns
  pace)){
        uaux[i] = bound_cond(grid.x(Kmin-Nspace-Nz+1+i),
  S0,
```

```
                              K,rebate,ttm,r-divid,produc
t,product_type);
    }
    else uaux[i] = u[Kmin-Nspace-Nz+1+i];
    uaux_img[i]=0;
  }

  fft1d(uaux,uaux_img,NN,-1);

  for(int i=0; i<NN; i++)
    {
      somme[i] = mu[i]*uaux[i] - mu_img[i]*uaux_img[i];
      somme_img[i] = mu_img[i]*uaux[i] + mu[i]*uaux_img
[i];
    }
  fft1d(somme,somme_img,NN,1);

  /*computation of the right-hand side vector v */
  /*centered discretization of the first order derivati
ve*/
  v[0] = u[0] + dt*(somme[NN-1]-measure.alpha*(u[1]-
                                               bound_
cond(grid.x(-1),S0,K,rebate,ttm,r,product,product_type))/2/dx

                    -measure.lambda*u[0]) +
  c*dt*bound_cond(grid.x(-1),S0,K,rebate,ttm+dt,r,prod
uct,product_type);
  v[Nspace-1] = u[Nspace-1] + dt*(somme[Nspace-2]-
                                measure.alpha*(bound_
cond(grid.x(Nspace),S0,K,rebate,ttm,r,product,product_type)
                                                 -u[Ns
pace-2])/2/dx -measure.lambda*u[Nspace-1])
  + b*dt*bound_cond(grid.x(Nspace),S0,K,rebate,ttm+dt,
r,product,product_type);
  for(int i=1; i<Nspace-1; i++)
    v[i] = u[i] + dt*(somme[i-1]-measure.alpha*(u[i+1]-
u[i-1])/2/dx -measure.lambda*u[i]);

  /*computation of u^(n+1)*/
  u = progonka(Nspace,-dt*c,1+dt*a,-dt*b,v);
  if(am)
```

```
  for(int i=Nspace-1;i>=0;i--)
   u[i]=MAX(u[i],exp(r*(ttm+dt))*init_cond(grid.x(i),S0,K,
    product));

    }//end of time iterations


  double actu = exp(-r*T);
  int N0 = (int) floor(-Al/dx);
  double Sl = S0*exp(grid.x(N0-1));
  double Sm = S0*exp(grid.x(N0));
  double Sr = S0*exp(grid.x(N0+1));

  // S0 is between Sm and Sr
  double pricel = actu*u[N0-1];
  double pricem = actu*u[N0];
  double pricer = actu*u[N0+1];

  //quadratic interpolation
  double A = pricel;
  double B = (pricem-pricel)/(Sm-Sl);
  double C = (pricer-A-B*(Sr-Sl))/(Sr-Sl)/(Sr-Sm);
  price0 = A+B*(S0-Sl)+C*(S0-Sl)*(S0-Sm);
  delta0 = B + C*(2*S0-Sl-Sm);

  delete [] mu;
  delete [] mu_img ;
  delete [] uaux;
  delete [] uaux_img;
  delete [] somme;
  delete [] somme_img;

  return u;
}//end price2c

#endif //PremiaCurrentVersion
```

References