```
  Help
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "Hawkes_Intensity_stdndc.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_root.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2010+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(ErraisGieseckeGoldberg)(void *Opt, void
    *Mod)
{
  return NONACTIVE;
}
int CALC(ErraisGieseckeGoldberg)(void *Opt, void *Mod,
    PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

// The calculs of the right hand of the ODE equality
// In:
// indata      : the input ODE at time t
// JS_param    : the parameter associated to the jump size
    distribution
// NbrJump     : the number of the jump aimed to compute
// Lbda_param : the lambda defined in the Laplace transfor
    m E(exp(<lambda,X_t>)), it can be Complex.
// kappa, delta and l_inf are defined with same name in th
    e article
// Out:
// retrun      : the output calculates the right hand function
static dcomplex D_RightH_function(dcomplex in_data, PnlMat*
```

```
     JS_param,int NbrJump,PnlVectComplex* Lbda_param,double ka
    ppa, double delta, double l_inf)
{
  dcomplex tmp1 =CZERO;
  dcomplex tmp2 =CZERO;
  dcomplex tmp3 =CZERO;
  dcomplex tmp4 =CZERO;

  int i=0;
  tmp1 = CRmul(in_data,delta);

  tmp2 =Cadd(pnl_vect_complex_get(Lbda_param,0),tmp1);
  for(i=0;i< NbrJump;i++)
    {
      tmp1 = CRmul(tmp2,pnl_mat_get(JS_param,0,i));
      tmp3 = Cexp(tmp1);
      tmp1 = CRmul(tmp3,pnl_mat_get(JS_param,1,i));
      tmp3 = CRadd(tmp4,0.);
      tmp4 = Cadd(tmp3,tmp1);
    }

  tmp1 =  Cmul( Cexp(pnl_vect_complex_get(Lbda_param,0)),tm
    p4);
  tmp2 = CRmul(in_data,-kappa);
  tmp3= Cadd(tmp2,tmp1);
  tmp4 =CRadd(tmp3,-1.);
  return tmp4;
}

// The calculs of the laplace transform in the complex cas
    e, using Riccati ODE
// In:
// intens_value : the initial value of the intensity
// initialvalue : the initial value of the loss and the
    count procesq (resp.)
// indata       : the input ODE at time t
// JS_param     : the parameter associated to the jump size
     distribution
// NbrJump      : the number of the jump aimed to compute
// Lbda_param   : the lambda defined in the Laplace transf
    orm E(exp(<lambda,X_t>)), it can be Complex.
```

```
// T             : the time value at maturity
// kappa, delta and l_inf are defined with same name in th
    e article
//  retrun       : the output compute the Laplace transform
static dcomplex LaplaceCompute(double intens_Value,PnlVectC
    omplex* Lbda_param,PnlVect* initialvalue,PnlMat* JS_param,
    int NbrJump,double kappa, double delta, double l_inf, double
    T)
{
  int NbrDisc =100;
  int i=0;
  double step=0.;
  dcomplex tmp1 =CZERO;
  dcomplex tmp2 =CZERO;
  dcomplex tmp3 =CZERO;
  dcomplex tmp4 =CZERO;
  dcomplex tmp11 =CZERO;
  dcomplex tmp22 =CZERO;
  step =(double) T/((double )NbrDisc) ;


  for(i=1;i<= NbrDisc;i++)
    {
      tmp1 = tmp11;
      tmp4 = tmp22;
      tmp2 = D_RightH_function(tmp11, JS_param,NbrJump,Lbda
    _param,kappa,delta, l_inf);
      tmp3 = CRmul(tmp2,step);

      tmp11= Cadd(tmp1,tmp3);

      tmp3 = CRmul(tmp1,kappa*l_inf*step);
      tmp22 = Cadd(tmp3,tmp4);

    }

  tmp1 = CRmul(tmp11,intens_Value);
  tmp2 = Cadd(tmp1,tmp22);
  tmp3= CRmul(pnl_vect_complex_get(Lbda_param,0),pnl_vect_
    get(initialvalue,0));
  tmp4= CRmul(pnl_vect_complex_get(Lbda_param,1),pnl_vect_
```

```
    get(initialvalue,1));
  tmp1=Cadd(tmp3,tmp4);
  tmp3 = Cadd(tmp1,tmp2);
  tmp4 =Cexp(tmp3);

  return tmp4;
}
// The calculs of the call option on the loss L_t on diffe
    rent strike
// In:
// strike        : the vector of different strike
// nbrstrike     : the number of strike
// intens_value : the initial value of the intensity
// initialvalue : the initial value of the loss and the
    count procesq (resp.)
// indata        : the input ODE at time t
// JS_param      : the parameter associated to the jump size
     distribution
// NbrJump       : the number of the jump aimed to compute
// Lbda_param    : the lambda defined in the Laplace transf
    orm E(exp(<lambda,X_t>)), it can be Complex.
// T             : the time value at maturity
// kappa, delta and l_inf are defined with same name in th
    e article
// Out:
// callgrid      : the value of call option associated to ea
    ch strike

static void  IFFTtStrike(PnlVect* strike,int nbrStrike,PnlV
    ect*callgrid,double intens_Value,PnlVect* initialvalue,Pn
    lMat* JS_param,int NbrJump,double kappa, double delta,
    double l_inf, double T)
{
  // initialization of the parameter
  double rho=0.05;
  int i;
  int j;
  int M = 100;

  double stepsize= 0.01;
```

```
double tmp;
dcomplex Ctmp=CZERO;
dcomplex Ctmp2=CZERO;
dcomplex Ctmp3=CZERO;
PnlVectComplex* laplace_value = pnl_vect_complex_create_
  from_dcomplex(M,CZERO);
PnlVectComplex* u_param = pnl_vect_complex_create_from_dc
  omplex(2,CZERO);

// Laplace computation in one shot
for(j=0;j<M;j++)
  {
    Ctmp.r = rho;
    Ctmp.i =( (double)(j))*stepsize;
    pnl_vect_complex_set(u_param,0,Ctmp);
    Ctmp= LaplaceCompute(intens_Value,u_param,initialvalu
  e,JS_param,NbrJump,kappa,delta,l_inf, T);
    pnl_vect_complex_set(laplace_value,j,Ctmp);


  }
//   pnl_vect_complex_print(laplace_value);

// Integral computation
for(j=0;j<nbrStrike;j++)
  {
    tmp=0;
    for(i=0;i<M;i++)
      {
        Ctmp.r = rho;
        Ctmp.i =(double) ((double)(i))*stepsize;
        Ctmp2 = Cexp(CRmul(Ctmp,-pnl_vect_get(strike,j)))
  ;
        Ctmp3 = Cdiv(Ctmp2,Cmul(Ctmp,Ctmp));
        Ctmp = Cmul(Ctmp3,pnl_vect_complex_get(laplace_
  value,i));
        tmp = tmp  +Creal(Ctmp)*M_1_PI;

      }

    tmp =tmp * stepsize;
```

```
      pnl_vect_set(callgrid,j,tmp);

    }
  //   pnl_vect_print(callgrid);
  // Desallocation memory
  pnl_vect_complex_free(&laplace_value);
  pnl_vect_complex_free(& u_param);
}
// The calculs of the spread of the CDO
static double   SpreadCompute(int nbr_c_frac,double upf,
    double intens_Value,PnlVect* initialvalue,PnlMat* JS_param,int
    NbrJump,double kappa, double delta, double l_inf, double T,
     double rate, PnlVect*strike)
{
  int nbrStrike;
  PnlVect*callgrid;
  int i;
  double tmp  =0.;
  double tmp2 =0.;
  double Dt=0.;

  PnlVect* StrikeTmp;
  double Ut;
  int nbrsteptime=50;
  double steptime = (double)(T/((double)nbrsteptime));

  // The calculs of the premium leg in the CDO
  nbrStrike =2;
  callgrid  = pnl_vect_create_from_double(nbrStrike,0.);
  StrikeTmp = pnl_vect_create_from_double(nbrStrike,0.);

  pnl_vect_set(StrikeTmp,0,pnl_vect_get(strike,0));
  pnl_vect_set(StrikeTmp,1,0.);


  tmp2=  pnl_vect_get(initialvalue,0)- pnl_vect_get(strike,
    1);
  if(tmp2>=0)
    tmp = tmp +tmp2;
```

```
tmp2 = pnl_vect_get(initialvalue,0)- pnl_vect_get(strike,
  0);
if(tmp2>=0)
  tmp =tmp - tmp2;
//   printf("tmp value %f {n",tmp);


IFFTtStrike(strike,nbrStrike,callgrid,intens_Value,initia
  lvalue,JS_param,NbrJump,kappa,delta,l_inf,T);

tmp = tmp + exp(-rate*T)*(pnl_vect_get(callgrid,0)-pnl_
  vect_get(callgrid,1));
//   printf("tmp value %f {n",tmp);

pnl_vect_set_zero(callgrid);

for(i=0;i<nbrsteptime;i++)
  {

    IFFTtStrike(strike,nbrStrike,callgrid,intens_Value,
  initialvalue,JS_param,NbrJump,kappa,delta,l_inf,(double) (
  steptime*((double)i)));
    Ut = pnl_vect_get(callgrid,0)-pnl_vect_get(callgrid,1
  );
    //   printf("Ut value %f {n",Ut);
    tmp =tmp + rate*exp(-(double)rate*((double) (step
  time*((double)i))))*steptime*Ut;

    pnl_vect_set_zero(callgrid);
  }

// The calculs of the fixed leg of the CDO
steptime =(double) T/(double)(nbr_c_frac);

for(i=0;i<nbr_c_frac;i++)
  {
    IFFTtStrike(strike,nbrStrike,callgrid,intens_Value,
  initialvalue,JS_param,NbrJump,kappa,delta,l_inf,(double) (
  steptime*((double)i)));
    Ut =  pnl_vect_get(callgrid,0)-pnl_vect_get(callgrid,
```

```
    1);

      Dt = Dt + ((pnl_vect_get(strike,1)-pnl_vect_get(stri
    ke,0))-Ut)*steptime*exp(-(double)rate*((double) (steptime*(
    (double)i))));

    }

  // Desallocation memory
  pnl_vect_free(&callgrid);
  pnl_vect_free(&StrikeTmp);

  return (tmp-(pnl_vect_get(strike,1)-pnl_vect_get(strike,0
    ))*upf)/Dt;



}
/* // The calculs of the upfront spread of the CDO
 * static double   UpfrCompute(int nbr_c_frac,double sprea
    d,double intens_Value,PnlVect* initialvalue,PnlMat* JS_para
    m,int NbrJump,double kappa, double delta, double l_inf,
    double T, double rate, PnlVect*strike)
 * {
 *   int nbrStrike;
 *   PnlVect*callgrid;
 *   int i;
 *   double tmp  =0.;
 *   double tmp2 =0.;
 *   double Dt=0.;
 *
 *   PnlVect* StrikeTmp;
 *   double Ut;
 *   int nbrsteptime=50;
 *   double steptime = (double)(T/((double)nbrsteptime));
 *
 *   nbrStrike =2;
 *   callgrid  = pnl_vect_create_from_double(nbrStrike,0.);
 *   StrikeTmp = pnl_vect_create_from_double(nbrStrike,0.);
 *
 *   pnl_vect_set(StrikeTmp,0,pnl_vect_get(strike,0));
```

```
*   pnl_vect_set(StrikeTmp,1,0.);
*
*   // The calculus of the premium leg
*   tmp2=  pnl_vect_get(initialvalue,0)- pnl_vect_get(stri
   ke,1);
*   if(tmp2>=0)
*     tmp = tmp +tmp2;
*
*
*
*   tmp2 = pnl_vect_get(initialvalue,0)- pnl_vect_get(stri
   ke,0);
*   if(tmp2>=0)
*     tmp =tmp - tmp2;
*
*
*   IFFTtStrike(strike,nbrStrike,callgrid,intens_Value,ini
   tialvalue,JS_param,NbrJump,kappa,delta,l_inf,T);
*   tmp = tmp + exp(-rate*T)*(pnl_vect_get(callgrid,0)-pn
   l_vect_get(callgrid,1));
*
*
*   pnl_vect_set_zero(callgrid);
*
*   for(i=0;i<nbrsteptime;i++)
*     {
*
*       IFFTtStrike(strike,nbrStrike,callgrid,intens_Value
   ,initialvalue,JS_param,NbrJump,kappa,delta,l_inf,(double)
   (steptime*((double)i)));
*       Ut = pnl_vect_get(callgrid,0)-pnl_vect_get(callgr
   id,1);
*
*       tmp =tmp + rate*exp(-(double)rate*((double) (step
   time*((double)i))))*steptime*Ut;
*
*       pnl_vect_set_zero(callgrid);
*     }
*   // The calculus of the fixed  leg
*
*   steptime =(double) T/(double)(nbr_c_frac);
```

```
 *
 *   for(i=0;i<nbr_c_frac;i++)
 *     {
 *       IFFTtStrike(strike,nbrStrike,callgrid,intens_Value
    ,initialvalue,JS_param,NbrJump,kappa,delta,l_inf,(double)
    (steptime*((double)i)));
 *       Ut =  pnl_vect_get(callgrid,0)-pnl_vect_get(callgr
    id,1);
 *
 *       Dt = Dt + ((pnl_vect_get(strike,1)-pnl_vect_get(
    strike,0))-Ut)*steptime*exp(-(double)rate*((double) (step
    time*((double)i))));
 *
 *     }
 *   // Desallocation memory
 *   pnl_vect_free(&callgrid);
 *   pnl_vect_free(&StrikeTmp);
 *
 *   return (tmp-spread*Dt)/(pnl_vect_get(strike,1)-pnl_vec
    t_get(strike,0));
 * } */

int CALC(ErraisGieseckeGoldberg)(void *Opt, void *Mod,
    PricingMethod *Met)
{
  TYPEOPT *ptOpt;
  TYPEMOD *ptMod;
  int      n_tranch;
  int      n, n_coupons, i;
  double   T, r;
  PnlVect *tranch;
  double spread;
  int dimstatejump;
  int xgrid_N;
  double kappa;
  double delta;
  double l_inf;
  double intens_Value;
  int nbr_c_frac;
  double upf;
```

```
PnlMat * statejump;
PnlVect* InitialValue;
PnlVect* xgrid;


ptOpt = (TYPEOPT*)Opt;
ptMod = (TYPEMOD*)Mod;


tranch = ptOpt->tranch.Val.V_PNLVECT;
n_tranch  = tranch->size-1;
n = ptMod->Ncomp.Val.V_PINT;
r = ptMod->r.Val.V_DOUBLE;
T = ptOpt->maturity.Val.V_DATE;
n_coupons = ptOpt->NbPayment.Val.V_INT;
kappa = ptMod->kappa.Val.V_DOUBLE;
delta = ptMod->delta.Val.V_DOUBLE;
l_inf = ptMod->c.Val.V_DOUBLE;
intens_Value = ptMod->lambda0.Val.V_DOUBLE;
spread =0.;

dimstatejump = 2;
xgrid_N =2;
nbr_c_frac = (int) (T * n_coupons);
upf  = 0.;

statejump = pnl_mat_create_from_double(2,dimstatejump,1./
  (double )(dimstatejump));
InitialValue= pnl_vect_create_from_double(2,0.);
xgrid = pnl_vect_create_from_double(xgrid_N,0.);
pnl_mat_set(statejump,0,0,0.2);
pnl_mat_set(statejump,0,1,0.4);
pnl_mat_set(statejump,1,0,0.5);
pnl_mat_set(statejump,1,1,0.5);

/* initialize Results. Have been allocated in Init
  method */
pnl_vect_resize (Met->Res[0].Val.V_PNLVECT, n_tranch);

for ( i=0 ; i<n_tranch ; i++ )
  {
```

```
      double strikeup = n * GET(tranch, i+1);
      double strikedn = n * GET(tranch, i);
      pnl_vect_set(xgrid,0,strikedn);
      pnl_vect_set(xgrid,1,strikeup);
      spread =  SpreadCompute(nbr_c_frac, upf,intens_Value,
    InitialValue,
                              statejump,dimstatejump,kappa,
    delta,l_inf,T,r,xgrid);
      LET(Met->Res[0].Val.V_PNLVECT, i) = 10000*spread;
    }

  pnl_mat_free(&statejump);
  pnl_vect_free(&xgrid);
  pnl_vect_free(&InitialValue);

  return OK;
}


static int CHK_OPT(ErraisGieseckeGoldberg)(void *Opt, void
    *Mod)
{
  Option*  ptOpt   = (Option*)Opt;
  if (strcmp (ptOpt->Name, "CDO_HAWKES_INTENSITY") != 0)
    return WRONG;
  return OK;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  TYPEOPT *ptOpt = (TYPEOPT*)Opt->TypeOpt;
  int      n_tranch;
  if ( Met->init == 0)
    {
      Met->init=1;
      n_tranch = ptOpt->tranch.Val.V_PNLVECT->size-1;

      Met->Res[0].Val.V_PNLVECT = pnl_vect_create_from_
    double (n_tranch, 0.);
    }
```

```
  return OK;
}

PricingMethod MET(ErraisGieseckeGoldberg) =
{
  "EGG_CDO_Pricing",
  {{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(ErraisGieseckeGoldberg),
  {{"Price(bp)",PNLVECT,{100},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(ErraisGieseckeGoldberg),
  CHK_ok,
  MET(Init)
};
```

References