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

#include "dynamic_stdndc.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_integration.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(EberleinFreyVHammerstein)(void *Opt, voi
    d *Mod)
{
  return NONACTIVE;
}
int CALC(EberleinFreyVHammerstein)(void *Opt, void *Mod,
    PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

typedef struct
{
  double lambda_0;
  double lambda_1;
  double lambda_2;
  double Index_Spread;
  int Nb_company;
  double recovery;
  int j;
} params;

/*
 Implementation of the inhomogenous model (see relation (20
    ) of Eberlein,Rudiger,Von hammerstein)
 */

static double h(double x, void *p)
```

```
{
  double lambda_0;
  double lambda_1;
  double lambda_2;
  double Index_Spread;
  int Nb_company;
  double recovery;
  int j;
  params *par = (params*)p;

  lambda_0 = par->lambda_0;
  lambda_1 = par->lambda_1;
  lambda_2 = par->lambda_2;
  Index_Spread = par->Index_Spread;
  Nb_company = par->Nb_company;
  recovery = par->recovery;
  j = par->j;

  return lambda_0 +(lambda_1/lambda_2)*(exp(lambda_2*MIN(
    MAX(j-Nb_company*(1-exp(-Index_Spread*x/(1-recovery))),0.0)/
    Nb_company,0.37))-1.) ;
}


static double primitive_h(double x, params *par)
{
  double result,abserr;
  int neval;
  PnlFunc func;
  func.function = h;
  func.params = par;
  pnl_integration_GK(&func,0.0,x,0.0001,0.0001,&result,&ab
    serr,&neval);
  return result;
}

/*
 **********************Computation of {mathbb E(L_t) using
    the forward Kolmogorov equation*************************
 */
```

```
double EV_lu(double t,double l,double u, double recovery,
             double lambda_0, double lambda_1, double lambd
    a_2,
             double Index_Spread, int Nb_company)
{
  int i;
  double Esp;
  double pp;
  double proba;
  PnlMat *M, *EXPM;
  params par;

  par.lambda_0 = lambda_0;
  par.lambda_1 = lambda_1;
  par.lambda_2 = lambda_2;
  par.Index_Spread = Index_Spread;
  par.Nb_company = Nb_company;
  par.recovery = recovery;


  M=pnl_mat_create_from_double(Nb_company+1,Nb_company+1,0.
    );
  EXPM=pnl_mat_create(0,0);

  for (i=0; i<Nb_company; i++)
    {
      par.j = i;
      pp=primitive_h(t,&par);
      pnl_mat_set(M,i,i,-(Nb_company-i)*pp);
      pnl_mat_set(M,i+1,i,(Nb_company-i)*pp);
    }
  pnl_mat_exp (EXPM, M);


  Esp=0;
  for(i=0;i<=Nb_company;i++)
    {
      proba=MGET(EXPM,i,0);   /*transitions probability of
    the process M_t */
      Esp+=(MAX((1-recovery)*i/(double)Nb_company-l,0.0)-
    MAX((1-recovery)*i/(double)Nb_company-u,0.0))*proba; /*tranc
```

```
    he loss*/
    }

   pnl_mat_free(&M);
   pnl_mat_free(&EXPM);

   return Esp;

}

static int eber(double r,double maturity,PnlVect *tranches,
                double recovery,double lambda_0, double lam
    bda_1, double lambda_2,
                double Index_Spread, int Nb_company,
    double frequency,
                PnlVect *prices, PnlVect *dleg, PnlVect *pl
    eg)
{

  int Ndate, N_tranches;
  int i,k;
  double Ti,Tj,A,B;
  double pl,dl;
  PnlVect *ee;

  N_tranches = tranches->size-1;
  Ndate=(int)(maturity/frequency);
  ee=pnl_vect_create_from_double(Ndate+1,0.);
  for(k=0;k<N_tranches;k++)
    {
      //Interval of tranches
      A=GET(tranches,k);
      B=GET(tranches,k+1);;

      //Compute Payment Leg  and Default Leg
      Ti=0;
      Tj=0.;
      pl=0;
      dl=0;
      for(i=0;i<=Ndate;i++)
        {
```

```
          pnl_vect_set(ee,i,EV_lu(Ti,A,B,recovery,lambda_0,
    lambda_1,lambda_2,
                                  Index_Spread,Nb_company))
    ;
          Ti+=frequency;
        }
      for(i=0;i<Ndate;i++)
        {

          Tj+=frequency;
          pl+=frequency*exp(-r*Tj)*((B-A)-pnl_vect_get(ee,
    i));
          dl+=exp(-r*Tj)*(pnl_vect_get(ee,i+1)-pnl_vect_get
    (ee,i));
        }

      pnl_vect_set (pleg, k, pl);
      pnl_vect_set (dleg, k, dl);
      pnl_vect_set (prices, k,10000*dl/pl);
    }
  pnl_vect_free(&ee);
  return OK;
}


int CALC(EberleinFreyVHammerstein)(void *Opt, void *Mod,
    PricingMethod *Met)
{
  TYPEOPT *ptOpt    = (TYPEOPT*)Opt;
  TYPEMOD *ptMod    = (TYPEMOD*)Mod;
  PnlVect *tranches;
  int      n_tranch = ptOpt->tranch.Val.V_PNLVECT->size-1;
  double   recovery, r, frequency, maturity;
  double   lambda_0, lambda_1, lambda_2, Index_Spread;
  int      Nb_company;

  /* initialize Results. Have been allocated in Init
    method */
  pnl_vect_resize (Met->Res[0].Val.V_PNLVECT, n_tranch);
  pnl_vect_resize (Met->Res[1].Val.V_PNLVECT, n_tranch);
```

```
  pnl_vect_resize (Met->Res[2].Val.V_PNLVECT, n_tranch);

  tranches = ptOpt->tranch.Val.V_PNLVECT;
  Nb_company = ptMod->Ncomp.Val.V_PINT;
  r = ptMod->r.Val.V_DOUBLE;
  lambda_0 = Met->Par[0].Val.V_DOUBLE;
  lambda_1 = Met->Par[1].Val.V_DOUBLE;
  lambda_2 = Met->Par[2].Val.V_DOUBLE;
  Index_Spread = Met->Par[3].Val.V_SPDOUBLE;

  maturity = ptOpt->maturity.Val.V_DATE;
  recovery = ptOpt->p_recovery.Val.V_DOUBLE;
  frequency = 1. / ptOpt->NbPayment.Val.V_INT;

  eber(r,maturity,tranches,recovery,lambda_0,
       lambda_1,lambda_2, Index_Spread,Nb_company, frequenc
    y,
       Met->Res[0].Val.V_PNLVECT,Met->Res[1].Val.V_PNLVECT,
       Met->Res[2].Val.V_PNLVECT);

  return OK;
}

static int CHK_OPT(EberleinFreyVHammerstein)(void *Opt, voi
    d *Mod)
{
  Option*  ptOpt   = (Option*)Opt;
  TYPEOPT *TypeOpt = (TYPEOPT*)ptOpt->TypeOpt;
  int      status  = 0;

  //return NONACTIVE;
  if (strcmp (ptOpt->Name, "CDO") != 0) return WRONG;
  if (TypeOpt->t_nominal.Val.V_ENUM.value != 1)
    {
      printf ("Only homogeneous nominals are accepted{n");
      status ++;
    }

  if (status) 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;
       Met->HelpFilenameHint = "EFVH_pricing_cdo";
      n_tranch = ptOpt->tranch.Val.V_PNLVECT->size-1;

      Met->Par[0].Val.V_PDOUBLE =0.004668;
      Met->Par[1].Val.V_PDOUBLE = 0.1921;
      Met->Par[2].Val.V_SPDOUBLE = 20.73;
      Met->Par[3].Val.V_SPDOUBLE =0.0039 ;

      Met->Res[0].Val.V_PNLVECT = pnl_vect_create_from_
    double (n_tranch, 0.);
      Met->Res[1].Val.V_PNLVECT = pnl_vect_create_from_
    double (n_tranch, 0.);
      Met->Res[2].Val.V_PNLVECT = pnl_vect_create_from_
    double (n_tranch, 0.);
    }
  return OK;
}

PricingMethod MET(EberleinFreyVHammerstein) =
{
  "EberleinFreyVHammerstein",
  {{"lambda_0", PDOUBLE,{0},FORBID},
   {"lambda_1", PDOUBLE,{0},FORBID},
   {"lambda_2", SPDOUBLE,{0},FORBID},
   {"Index Spread", SPDOUBLE,{0},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(EberleinFreyVHammerstein),
  {{"Price(bp)",PNLVECT,{100},FORBID},
   {"D_leg",PNLVECT,{100},FORBID},
   {"P_leg",PNLVECT,{100},FORBID},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(EberleinFreyVHammerstein),
```

```
  CHK_ok,
  MET(Init)
};
```

References