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

#include "copula_stdndc.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_mathtools.h"
#include "math/cdo/copulas.h"
#include "math/cdo/cdo.h"


/*
 * July 2008.
 * Qi Zong (second year ENSTA student) who worked under th
    e supervision of
 * Céline Labart.
 *
 * This code has been modified by Jérôme Lelong to use the
    PremiaCopula
 * structure.
 */

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2009+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 (Stein) (void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC (Stein) (void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/*
   n - No.of company
   r - interst rate
```

```
   */

static double  n, r, intensity;

// calculer l'esperance approximation sachant V
static double esperance_V (const copula *cop, double R,
    double k, double temp_frac, double pr)
{
  double Ky, E_poisson, E_normal, E_normal_R;
  double correction, lamda, esp_R, var_R;
  double sigma2, m, temp, exp_lamda, floor_m;
  double pow_lamda, var_X, temp_R, flag; //flag controle fact
  int i;
  int mode = 0;

  esp_R = 0.4;
  var_R = 0.0001;

  temp = 1;
  E_normal = 0;
  E_poisson = 0;

  m = n * k / (1 - R);

  if (pr * n < 10) mode = 1;
  if (pr *n >= 10) mode = 2;
  /*-------------------------------poisson-----------------
    --*/
  if (mode == 1)
    {
      //esperance sans correction
      lamda = n * pr;
      pow_lamda = 1;
      exp_lamda = exp (-lamda);
      flag = n;
      if (n > 100) flag = 100;
      for (i = 0; i <= flag; i++)
        {
          E_poisson = E_poisson + MAX (i - m, 0) * pow_lam
    da * exp_lamda / temp;
          temp = temp * (i + 1);
```

```
        pow_lamda = pow_lamda * lamda;
      }
    //correction
    sigma2 = pr * (1 - pr) * n;
    if (m == 0 || temp_frac == 0)
      correction = 0;
    else
      {
        floor_m = floor (m);
        correction = (sigma2 - lamda) * exp_lamda * pow (
  lamda, floor_m - 1) / (2 * temp_frac) * (m * lamda / floor_m
   - lamda + floor_m - m + 1);

      }
    return (1 - R) * (E_poisson + correction) / n; //ret
  ourner l'esperance avec correcion
  }

/*-------------------------------normal-----------------
  -*/
if (mode == 2)
  {
    if (pr == 1) return 0.0;
    Ky = ( k - (1 - R) * pr ) * sqrt (n) / ( (1 - R) * sq
  rt ( pr * (1 - pr) ) );
    //esperance sans correction
    E_normal = ( (1 - R) * sqrt (pr * (1 - pr) ) / sqrt (
  n)  ) * ( exp (-Ky * Ky / 2) / sqrt (2 * M_PI) - Ky * (1 -
  cdf_nor (Ky) ) );
    correction = ( (1 - R) * (1 - 2 * pr) * Ky * exp (-Ky
   * Ky / 2) ) / ( 6 * n * sqrt (2 * M_PI) );
    return E_normal + correction; //retourner l'esperanc
  e avec correcion
  }

/*-------------------------------stochastic recovery ra
  te gaussian case------------------*/
//esp_R(esperance de R),var_R(variance de R),var_X
if (mode == 3)
  {
    var_X = pr * (var_R + (1 - pr) * (1 - esp_R) * (1 -
```

```
    esp_R) ) / (n * n);
      Ky = k - (1 - esp_R) * pr;
      E_normal_R = sqrt ( n * var_X / (2 * M_PI) ) *
        exp (-Ky * Ky / (2 * n * var_X) ) - Ky +
        Ky * cdf_nor (Ky / sqrt (n * var_X) );
      temp_R = pr / (n * n * n) * ( pow (1 - esp_R, 3) * (1
     - pr) *
                                    (1 - 2 * pr) + 3 * (1 -
     pr) *
                                    (1 - esp_R) * var_R );
       correction = 1 / (6 * var_X) * (temp_R) * Ky *
         exp (-Ky * Ky / (2 * n * var_X) ) / (sqrt (2 * M_PI
     * n * var_X) );
       return E_normal_R + correction;
    }

  return 0;
}

//calculer l'esperance approximation finale
static double esperance_homo (const copula *cop, double R,
    double k, double t)
{
  double f_t, sum, temp_m, temp_frac;
  double *pr_t;
  int i, j;

  sum = 0;
  temp_m = n * k / (1 - R);

  temp_frac = 0;
  if (temp_m >= 1 && temp_m <= 100)
    temp_frac = pnl_fact ( floor (temp_m) - 1 );

  f_t = 1 - exp (-intensity * t);

  pr_t = cop->compute_cond_prob (cop, f_t);
  if ( cop->nfactor == 1 )
    {
      int i;
      for ( i = 0 ; i < cop->size ; i++ )
```

```
        {
          sum += esperance_V (cop, R, k, temp_frac, pr_t[i]
    ) * cop->weights[i];
        }
    }
  else /* nfactor == 2 in this case */
    {
      for ( i = 0 ; i < cop->size ; i++ )
        {
          double sum_i = 0.;
          for ( j = 0 ; j < cop->size ; j++ )
            {
              sum_i += esperance_V (cop, R, k, temp_frac,
    pr_t[i * cop->size + j]) * cop->weights[j];
            }
          sum += sum_i * cop->weights[cop->size + i];
        }
    }
  free (pr_t);
  return sum;
}

//esperance de trance (A,B)
static double tranche_homo (const copula *cop, double R,
    double A, double B, double t)
{
  return esperance_homo (cop, R, A, t) - esperance_homo (
    cop, R, B, t);
}

//calculer le default leg
static double compute_d_leg (const copula *cop, double R,
    double A, double B,
                             double time_begin, double
    time_end, int num)
{
  double length, sum, t_1, t_0;
  int i;
  sum = 0;
  length = (time_end - time_begin) / num;
```

```
  for (i = 0; i < num; i++)
    {
      t_0 = time_begin + length * (i);
      t_1 = t_0 + length;

      sum = sum + exp ( -r * (t_1 + t_0 ) / 2 ) *
        ( tranche_homo (cop, R, A, B, t_1 ) - tranche_homo
    (cop, R, A, B, t_0) );
    }

  return sum;
}



//payment leg
static double compute_p_leg (const copula *cop, double R,
    double A, double B,
                                double time_begin, double
    time_end, int num, int sub )
{
  double length, sum, t_1, t_0, temp, length_big;
  int i;

  sum = 0;
  length = (time_end - time_begin) / num;
  for (i = 0; i < num; i++)
    {
      t_0 = time_begin + length * (i);
      t_1 = t_0 + length;

      sum = sum + exp ( -r * t_1  ) * ( B - A - tranche_ho
    mo (cop, R, A, B, t_1) ) * (t_1 - t_0);
    }
  length_big = length;
  length = length / sub;

  for (i = 0; i < num * sub; i++)
    {
      t_0 = time_begin + length * (i);
      t_1 = t_0 + length;
```

```
      temp = (t_1 + t_0) / 2;
      sum = sum + exp ( -r * temp ) * ( tranche_homo (cop,
    R, A, B, t_1 ) - tranche_homo (cop, R, A, B, t_0) ) *
        ( temp - ( (int) (temp / length_big) ) * length_bi
    g );
    }
  return sum;
}


void for_pr_Hmat (const copula *cop, double T, int sub,
    int N_time, int N_comp,
                  const PnlVect *h_V, PnlHmat *pr_Hmat)
{
  double f_t, length = 0;
  int i, j, iv;
  int place[3] = {0, 0, 0}; /* V, company, time */
  length = (double) T / (sub * N_time);
  for (i = 0; i < N_comp; i++) // company
    {
      place[1] = i;
      for (j = 0; j <= N_time * sub; j++) //time
        {
          double *pr_temp;
          const int pr_size = pnl_pow_i (cop->size, cop->nf
    actor);
          place[2] = j;
          f_t = 1 - exp (-GET (h_V, i) * j * length);
          pr_temp = cop->compute_cond_prob (cop, f_t);
          for ( iv = 0 ; iv < pr_size ; iv++ )
            {
              place[0] = iv;
              pnl_hmat_set (pr_Hmat, place, pr_temp[iv]);
            }
          free (pr_temp);
        }
    }
}

/**
 * Used by esperance in the inner loop.
```

```
 * Computes the expectation before the numerical integration
 */
double esperance_aux (const copula *cop,
                      double k, int *place, int N_comp,
                      const PnlVect *nom_V, const PnlVect *
    R_V,
                      const PnlHmat *pr_Hmat)
{
  double Ky, E_normal, correction;
  double SIGMA2, sigma2, omega, R, pr, sum, temp;
  int i;
  sum = 0;
  SIGMA2 = 0;
  correction = 0;
  for (i = 0; i < N_comp; i++)
    {
      place[1] = i;
      pr = pnl_hmat_get (pr_Hmat, place);
      omega = GET (nom_V, i);
      R = GET (R_V, i);
      temp = omega * (1 - R) * pr;
      sum += temp;
      sigma2 = temp * omega * (1 - R) * (1 - pr);
      SIGMA2 += sigma2;

      /*
       * We need this tweak because it may happen that pr =
    = 0
       * especially at time == 0 and in this case correctio
    n returns
       * NaN
       */
      if ( pr <= 1E-18 )
        correction = 0.;
      else
        correction = correction + sqrt (sigma2) * sigma2 *
    (1 - 2 * pr)
          / sqrt (pr * (1 - pr) );
    }
  if ( SIGMA2 <= 1E-18 )
    {
```

```
      /*
       * This means that pr was always 0 which is the case
    at time == 0
       */
      E_normal = 0.;
      correction = 0.;
    }
  else
    {
      Ky = (k - sum) / sqrt (SIGMA2);
      E_normal = sqrt (SIGMA2) * ( exp (-Ky * Ky / 2) / sq
    rt (2 * M_PI) - Ky * ( 1 - cdf_nor (Ky) ) );
      correction *= Ky * exp (-Ky * Ky / 2) / (sqrt (2 * M_
    PI) * 6 * SIGMA2);
    }
  return E_normal + correction;
}

double esperance (const copula *cop,
                  double k, int time, int N_comp,
                  const PnlVect *nom_V, const PnlVect *R_V,
                  const PnlHmat *pr_Hmat)
{
  double res, sum, sum_final;
  int iv, jv;
  int place[3];
  place[0] = 0;
  place[1] = 0;
  place[2] = time;
 sum_final = 0;

  if ( cop->nfactor == 1 )
    {
      for ( iv = 0 ; iv < cop->size ; iv++ )
        {
          place[0] = iv;
          res = esperance_aux (cop, k, place, N_comp, nom_
    V, R_V, pr_Hmat);
          sum_final += res * cop->weights[iv];
        }
    }
```

```
  else /* nfactor == 2 */
    {
      for ( iv = 0 ; iv < cop->size ; iv++ )
        {
          sum = 0.;
          for ( jv = 0 ; jv < cop->size ; jv++ )
            {
              place[0] = iv * cop->size + jv;
              res = esperance_aux (cop, k, place, N_comp,
    nom_V, R_V, pr_Hmat);
              sum += res * cop->weights[jv];
            }
          sum_final += res * cop->weights[iv + cop->size];
        }
    }
  return sum_final;
}

//esperance de trance (A,B)
double tranche (const copula *cop,
                double A, double B, int time, int n, const
    PnlVect *nom_V,
                const PnlVect *R_V, const PnlHmat *pr_Hmat)
{
  return ( esperance (cop, A, time, n, nom_V, R_V, pr_Hmat)
           - esperance (cop, B, time, n, nom_V, R_V, pr_Hm
    at) );
}

//default leg inhomogeneous
double D_leg (const copula  *cop,
              double r, double A, double B,
              double T, int num, int n,
              const PnlVect *nom_V, const  PnlVect *R_V,
    const PnlHmat *pr_Hmat)
{
  double length = 0, sum = 0;
  int i, t_0, t_1;
  sum = 0;
  length = T / num;
  for (i = 0; i < num; i++)
```

```
    {
      t_0 = i;
      t_1 = i + 1;
      sum = sum + exp ( -r * (t_1 + t_0 ) * length / 2 ) *
        ( tranche ( cop, A, B, t_1, n, nom_V, R_V, pr_Hmat
    ) - tranche (cop, A, B, t_0, n, nom_V, R_V, pr_Hmat) );
    }
  return sum;
}



//payment leg inhomogeneous
double P_leg (const copula  *cop,
              double r, double A, double B, double T,
              int num, int sub, int n,
              const PnlVect *nom_V, const PnlVect *R_V,
    const PnlHmat *pr_Hmat )
{
  double length = 0, sum = 0, temp = 0, length_big = 0;
  int i;
  int t_1, t_0;
  sum = 0;
  length = T / num;
  for (i = 0; i < num; i++)
    {
      t_0 = i;
      t_1 = i + 1;
      sum = sum + exp ( -r * t_1 * length  ) *
        ( B - A - tranche (cop, A, B, (int) (t_1 * sub), n,
     nom_V, R_V, pr_Hmat ) )
        * (t_1 - t_0) * length;
    }
  length_big = length;
  length = length / sub;

  for (i = 0; i < num * sub; i++)
    {
      t_0 = i;
      t_1 = i + 1;
      temp = (t_1 + t_0) * length / 2;
```

```
      sum = sum + exp ( -r * temp ) * ( tranche (cop, A, B,
     t_1, n, nom_V, R_V, pr_Hmat ) -
                                        tranche (cop, A, B,
     t_0, n, nom_V, R_V, pr_Hmat) )
        * ( temp - ( (int) (temp / length_big) ) * length_
    big   );
    }
  return sum;
}

static void price_cdo_inhomo (const copula *cop, int N_
    time, int sub, double r,
                              double T, int nb_comp,
                              const PnlVect   *tranches,
                              const PnlVect *names, const
    PnlVect *intensity,
                              const PnlVect *recovery,
                              PnlVect *Price, PnlVect *Dleg
    , PnlVect *Pleg )
{
  int taille[3];
  double A, B;
  int i;
  PnlHmat *pr_Hmat;

  taille[0] = pnl_pow_i (cop->size, cop->nfactor);
  taille[1] = nb_comp; /* Number of companies */
  taille[2] = N_time * sub + 1; /* Number of time steps */

  //set pr_Hmat
  pr_Hmat = pnl_hmat_create (3, taille);
  for_pr_Hmat (cop, T, sub, N_time, nb_comp, intensity, pr_
    Hmat);

  /*
   * In the case of the Clayton copula, the conditional
    probability is
   * always 0 at time 0 whatever the value of V. This value
     of the
   * conditional probability is not compatible with the
    correction used in
```

```
   * the non homogeneous case
   */

  for (i = 0; i < tranches->size - 1; i++)
    {
      A = GET (tranches, i);
      B = GET (tranches, i + 1);
      LET (Dleg, i) = D_leg (cop, r, A, B, T, N_time * sub,
     nb_comp, names, recovery, pr_Hmat);
      LET (Pleg, i) = P_leg (cop, r, A, B, T, N_time, sub,
    nb_comp, names, recovery, pr_Hmat);
      LET (Price, i) = GET (Dleg, i) / GET (Pleg, i) * 1000
    0;
    }
  pnl_hmat_free (&pr_Hmat);
}



static void price_cdo_homo (const copula *cop, double R,
    int N_time,
                            int sub, int T, const PnlVect
     *tranches,
                            PnlVect *Price, PnlVect *D_leg,
     PnlVect *P_leg )
{
  int i;
  double A, B;
  double p_leg, d_leg;
  for (i = 0; i < tranches->size - 1; i++)
    {
      A = pnl_vect_get (tranches, i);
      B = pnl_vect_get (tranches, i + 1);
      d_leg = compute_d_leg (cop, R, A, B, 0, T, N_time *
    sub);
      p_leg = compute_p_leg (cop, R, A, B, 0, T, N_time, su
    b);
      LET (D_leg, i) = d_leg;
      LET (P_leg, i) = p_leg;
      pnl_vect_set (Price, i, d_leg / p_leg * 10000);
    }
```

```
}


int CALC (Stein) (void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT          *ptOpt     = (TYPEOPT *) Opt;
  TYPEMOD          *ptMod     = (TYPEMOD *) Mod;
  VAR              *Par;
  int               n_tranch  = ptOpt->tranch.Val.V_PNLVEC
    T->size - 1;
  int               sub         , N_time, t_copula, is_ho
    mo = 1;
  int               generator = Met->Par[1].Val.V_ENUM.val
    ue;
  double           *p_copula    , R, maturity;
  copula           *cop;
  PnlVect          *nominal   = NULL, *v_intensity = NULL,
    *recovery = NULL;

  /* 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);
  pnl_rand_init (generator, 0, 0);

  t_copula = (ptMod->t_copula.Val.V_ENUM.value);
  Par = lookup_premia_enum_par(&(ptMod->t_copula), t_    copula);
  p_copula = Par[0].Val.V_PNLVECT->array;
  cop = init_copula (t_copula, p_copula);

  n = ptMod->Ncomp.Val.V_PINT;
  r = ptMod->r.Val.V_DOUBLE;


  maturity = ptOpt->maturity.Val.V_DATE;

  sub = Met->Par[0].Val.V_INT;
  N_time = maturity * ptOpt->NbPayment.Val.V_INT;
```

```
/* nominal */
if (ptOpt->t_nominal.Val.V_ENUM.value == 1)
  {
    nominal = pnl_vect_create_from_double (n, 1. / (
  double) n);
  }
else
  {
    is_homo = 0;
    Par = lookup_premia_enum_par (&(ptOpt->t_nominal), 2)
  ;
    nominal = pnl_vect_create_from_file (Par[0].Val.V_FIL
  ENAME);
  }

/* intensity */
if (ptMod->t_intensity.Val.V_ENUM.value == 1)
  {
    Par = lookup_premia_enum_par (&(ptMod->t_intensity),
  1);
    intensity = Par[0].Val.V_PDOUBLE;
    v_intensity = pnl_vect_create_from_double (n,
  intensity);
  }
else
  {
    is_homo = 0;
    Par = lookup_premia_enum_par (&(ptMod->t_intensity),
  0);
    v_intensity = pnl_vect_create_from_file (Par[0].Val.
  V_FILENAME);
  }

/* Check if recovery is constant. If not, force the inhom
  ogeneous case. */
switch (ptOpt->t_recovery.Val.V_ENUM.value)
  {
  case T_RECOVERY_CONSTANT:
      {
        VAR *Par = lookup_premia_enum_par(&(ptOpt->t_reco
  very), T_RECOVERY_CONSTANT);
```

```
          R = Par[0].Val.V_DOUBLE;
          /* Constant Recovery but we are going to use the
    code for the inhomogeneous case */
          if ( is_homo == 0 ) recovery = pnl_vect_create_
    from_double (n, R);
        }
      break;
    case T_RECOVERY_UNIFORM:
        {
          VAR *Par = lookup_premia_enum_par(&(ptOpt->t_reco
    very), 2);
          PnlVect *params = Par[0].Val.V_PNLVECT;
          is_homo = 0;
          recovery = pnl_vect_create (0);
          pnl_vect_rand_uni (recovery, n, GET (params, 0),
    GET (params, 1), generator);
        }
      break;
    }


  if (is_homo)
    price_cdo_homo (cop, R, N_time, sub, maturity, ptOpt->
    tranch.Val.V_PNLVECT,
                    Met->Res[0].Val.V_PNLVECT,
                    Met->Res[1].Val.V_PNLVECT, Met->Res[2].
    Val.V_PNLVECT);
  else
    price_cdo_inhomo (cop, N_time, sub, r, maturity, n,
                      ptOpt->tranch.Val.V_PNLVECT,
                      nominal, v_intensity, recovery,
                      Met->Res[0].Val.V_PNLVECT,
                      Met->Res[1].Val.V_PNLVECT, Met->Res[2
    ].Val.V_PNLVECT);

  free_copula (&cop);
  pnl_vect_free (&nominal);
  pnl_vect_free (&v_intensity);
  pnl_vect_free (&recovery);
  return OK;
}
```

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

#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->Par[0].Val.V_INT = 4;
      Met->Par[1].Val.V_ENUM.value = 0;
      Met->Par[1].Val.V_ENUM.members = &PremiaEnumRNGs;

      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 (Stein) =
{
  "Stein",
  { {"N subdvisions", INT, {4}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC (Stein),
  { {"Price(bp)", PNLVECT, {100}, FORBID},
      {"D_leg", PNLVECT, {100}, FORBID},
```

```
      {"P_leg", PNLVECT, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT (Stein),
  CHK_ok,
  MET (Init)
};
```

References