Help

```
#include <stdio.h>
#include <stdlib.h>
#include <
href../../../common/math/cdo/cdo_math_h_src.pdfmath.h>

#include "
href../../../mod/dynamic/dynamic_stdndc/dynamic_stdndc_h_src.pdfdynamic_stdndc.h
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_integration.h"


#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
static int CHK_OPT(Herbertsson)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(Herbertsson)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double herbertsson_cdo(double T, int n, int M, double r, double R,
                              const PnlVect *a1, double att, double det
                             )
{
  /** T maturite du derive de credit**/
  /** n nombre de subdivision de l'intervalle [0,T]**/
  /** r taux d'interet**/
  /** R recovery en cas de defaut**/
  /** a vecteur decrivant le saut de l'intensite il est de taille M
      nombre de firmes soumis au risque de defaut**/
  /** att et det representent les points d'attachement et de
      detachement de la  tranche de CDO dont on calculera le prix**/


  PnlMat *Q;  /** matrice de transition*/
  PnlVect *l; /**vecteur de perte**/
  double DL = 0; /** default leg selon les tranches**/
```

```
double PL = 0; /** payment leg selon les tranches**/
PnlMat *G, *G1;
PnlVect *y;
PnlVect *a;
PnlMat *D;
PnlMat *F;
int k, i;
int n_l, n_u;
double s = 0;
double step = T * 1. / n;
double spread;
int n1, n2, n3, n4, n5;

/** nbre de defauts affectant les tranches standards**/
n1 = 1 + trunc(0.03 * M / (1 - R));
n2 = 1 + trunc(0.06 * M / (1 - R));
n3 = 1 + trunc(0.09 * M / (1 - R));
n4 = 1 + trunc(0.12 * M / (1 - R));
n5 = 1 + trunc(0.22 * M / (1 - R));

a = pnl_vect_create_from_double(M + 1, 0);
pnl_vect_set(a, 0, pnl_vect_get(a1, 0));

for (i = 1; i < n1; i++)
  {
    pnl_vect_set(a, i, pnl_vect_get(a1, 1));
  }
for (i = n1; i < n2; i++)
  {
    pnl_vect_set(a, i, pnl_vect_get(a1, 2));
  }
for (i = n2; i < n3; i++)
  {
    pnl_vect_set(a, i, pnl_vect_get(a1, 3));
  }
for (i = n3; i < n4; i++)
  {
    pnl_vect_set(a, i, pnl_vect_get(a1, 4));
  }
for (i = n4; i < n5; i++)
  {
```

```
    pnl_vect_set(a, i, pnl_vect_get(a1, 5));
  }
for (i = n5; i < M + 1; i++)
  {
    pnl_vect_set(a, i, pnl_vect_get(a1, 6));
  }
/**Definition de la matrice de transition**/
Q = pnl_mat_create_from_double(M + 1, M + 1, 0);
D = pnl_mat_create_from_double(M + 1, M + 1, 0);
F = pnl_mat_create_from_double(M + 1, M + 1, 0);
G = pnl_mat_create_from_double(M + 1, M + 1, 0);
G1 = pnl_mat_create_from_double(M + 1, M + 1, 0);

for (k = 0; k < M; k++)
  {
    s = s + pnl_vect_get(a, k);
    pnl_mat_set(Q, k, k, -(M - k)*s);
    pnl_mat_set(Q, k, k + 1, (M - k)*s);
  }
/** Iteration pour le calcul du prix de chaque tranche CDO**/
l = pnl_vect_create_from_double(M + 1, 0);
y = pnl_vect_create_from_double(M + 1, 0);
pnl_mat_clone(D, Q);

for (i = 0; i < M + 1; i++)
  {
    pnl_mat_set(D, i, i, pnl_mat_get(D, i, i) - r);
  }
/**calcul de l'inverse de Q-rI**/
pnl_mat_upper_inverse(G, D);
n_l = 1 + trunc(att * M / (1 - R)); /** nbre de defaut à partir duquel la tran
n_u = 1 + trunc(det * M / (1 - R)); /** nbre maximum de defaut qui affecte la

/**Definition du vecteur l pour la p-ieme tranche CDO**/
for (i = 0; i < M + 1; i++)
  {
    if (i < n_l)  pnl_vect_set(l, i, 0);
    else if (i >= n_l)
      {
        if (i < n_u) pnl_vect_set(l, i, i * (1 - R) * 1. / M - att);
        else  pnl_vect_set(l, i, det - att);
```

```
      }
  }
pnl_mat_clone(D, Q);
pnl_mat_mult_double(D, step);

if (det > 0.03)
  {
    for (k = 1; k < n + 1; k++)
      {
        /**Calcul  du  DL**/
        pnl_mat_exp(F, D);
        pnl_mat_mult_vect_inplace(y, F, l);
        DL += exp(-r * step * k) * step * (det - att - pnl_vect_get(y, 0));
        pnl_mat_mult_double(D, (k + 1) * 1. / k);
      }
    pnl_mat_mult_double(F, exp(-r * T));
    pnl_mat_clone(D, F);

    for (i = 0; i < M + 1; i++)
      {
        pnl_mat_set(F, i, i, pnl_mat_get(F, i, i) - 1);
      }
    pnl_mat_mult_mat_inplace(G1, F, G);
    pnl_mat_clone(G, G1);
    pnl_mat_mult_double(G, r);
    pnl_mat_plus_mat(G, D);
    pnl_mat_mult_vect_inplace(y, G, l);
    PL = pnl_vect_get(y, 0);

    /** Calcul du spread exprime en bps**/;
    spread = 10000 * PL / DL;
  }
else
  {
    pnl_mat_clone(D, Q);
    pnl_mat_mult_double(D, step);

    for (k = 1; k < n + 1; k++)
      {
        /**Calcul  du  DL**/
        // F=expdm2(D,M+1);
```

```
          pnl_mat_exp(F, D);
          pnl_mat_mult_vect_inplace(y, F, l);
          DL += 0.05 * exp(-r * step * k) * step * pnl_vect_get(y, 0);
          pnl_mat_mult_double(D, (k + 1) * 1. / k);
        }
      pnl_mat_mult_double(F, exp(-r * T));
      pnl_mat_clone(D, F);

      for (i = 0; i < M + 1; i++)
        {
          pnl_mat_set(F, i, i, pnl_mat_get(F, i, i) - 1);
        }
      pnl_mat_mult_mat_inplace(G1, F, G);
      pnl_mat_clone(G, G1);
      pnl_mat_mult_double(G, r);
      pnl_mat_plus_mat(G, D);
      pnl_mat_mult_vect_inplace(y, G, l);
      spread = (DL + pnl_vect_get(y, 0)) / (det - att);

      for (k = 1; k < n + 1; k++)
        {
          spread -= 0.05 * exp(-r * k * step) * step;
        }
      spread = 10000 * spread;
    }

  pnl_mat_free(&G);
  pnl_mat_free(&G1);
  pnl_mat_free(&D);
  pnl_mat_free(&Q);
  pnl_mat_free(&F);
  pnl_vect_free(&y);
  pnl_vect_free(&l);
  pnl_vect_free(&a);

  return (spread); /** spread exprime en pourcentage **/
}


int CALC(Herbertsson)(void *Opt, void *Mod, PricingMethod *Met)
{
```

```
  TYPEOPT      *ptOpt    = (TYPEOPT *)Opt;
  TYPEMOD      *ptMod    = (TYPEMOD *)Mod;
  int           n_tranch = ptOpt->tranch.Val.V_PNLVECT->size - 1;
  int           n, sub, i;
  double        recovery, r, prix, maturity;
  PnlVect *tranch;
  PnlVect *a;   /** vecteur des intensites */

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

  n = ptMod->Ncomp.Val.V_PINT;
  r = ptMod->r.Val.V_DOUBLE;
  maturity = ptOpt->maturity.Val.V_DATE;
  recovery = ptMod->Recovery.Val.V_PDOUBLE;
  tranch = ptOpt->tranch.Val.V_PNLVECT;
  sub = (int)(ptOpt->NbPayment.Val.V_INT * maturity);

  a = pnl_vect_create(7);
  /** Intensites associees***/
  pnl_vect_set(a, 0, 0.0033);
  pnl_vect_set(a, 1, 0.00164);
  pnl_vect_set(a, 2, 0.00845);
  pnl_vect_set(a, 3, 0.0145);
  pnl_vect_set(a, 4, 0.00864);
  pnl_vect_set(a, 5, 0.0124);
  pnl_vect_set(a, 6, 0.0514);

  for (i = 0 ; i < n_tranch ; i++)
    {
      prix = herbertsson_cdo(maturity, sub, n, r, recovery, a, GET(tranch, i), G
      LET(Met->Res[0].Val.V_PNLVECT, i) = prix;
    }
  pnl_vect_free(&a);

  return OK;
}

static int CHK_OPT(Herbertsson)(void *Opt, void *Mod)
{
  Option  *ptOpt   = (Option *)Opt;
```

```
  TYPEOPT *TypeOpt = (TYPEOPT *)ptOpt->TypeOpt;
  int      status  = 0;

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

```
};
```