Help

```
#include "
href../../../mod/hullwhite2d/hullwhite2d_stdi/hullwhite2d_stdi_h_src.pdfhullwhit
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "
href../../../common/math/InterestRateModelTree/TreeHW2D/TreeHW2D_h_src.pdfmath/I
#include "
href../../../mod/hullwhite2d/hullwhite2d_stdi/hullwhite2d_includes_h_src.pdfhull

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2009+2)
int CALC(TR_BERMUDIANSWAPTIONHW2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(TR_BERMUDIANSWAPTIONHW2D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
#else

/// TreeHW2D     : structure that contains components of the tree (see ModelHW2D
/// ModelHW2D    : structure that contains  the parameters of the Hull&White one
/// ZCMarketData : structure that contains the Zero Coupon Bond prices of the ma


/// Computation of the payoff at the final time of the tree (ie the option matur
static void BermudianSwaption_InitialPayoffHW2D(int swaption_start, TreeHW2D *Me
{
  double a , sigma1, b, sigma2, rho, sigma3;

  int jminprev, jmaxprev, kminprev, kmaxprev; // jmin[i], jmax [i]
  int i, j, k, NumberOfPayments; // i = represents the time index. j, k represen

  double delta_y2; // delta_y1 = space step of the process y at time i ; delta_y
  double delta_u2; // delta_u1 = space step of the process u at time i ; delta_u
  double delta_t1; // time step

  double ZCPrice, SumZC; //ZC price
```

```
double current_rate, current_u;
double Ti;

ZCPrice = 0;
// Parameters of the processes r, u and y
a = (ModelParam->rMeanReversion);
sigma1 = (ModelParam->rVolatility);

b = (ModelParam->uMeanReversion);
sigma2 = (ModelParam->uVolatility);

rho = (ModelParam->correlation);

sigma3 = sqrt(sigma1 * sigma1 + sigma2 * sigma2 / ((b - a) * (b - a)) + 2 * rh

// Computation of the vector of payoff at the maturity of the option
jminprev = pnl_vect_int_get(Meth->yIndexMin, swaption_start);  // jmin(swaptio
jmaxprev = pnl_vect_int_get(Meth->yIndexMax, swaption_start);  // jmax(swaptio
kminprev = pnl_vect_int_get(Meth->uIndexMin, swaption_start);  // kmin(swaptio
kmaxprev = pnl_vect_int_get(Meth->uIndexMax, swaption_start);  // kmax(swaptio

pnl_mat_resize(OptionPriceMat2, jmaxprev - jminprev + 1, kmaxprev - kminprev +

delta_t1 = GET(Meth->t, swaption_start) - GET(Meth->t, swaption_start - 1); //
delta_y2 = delta_xHW2D(delta_t1, a, sigma3); // delta_y (swaption_start)
delta_u2 = delta_xHW2D(delta_t1, b, sigma2); // delta_u (swaption_start)

NumberOfPayments = (int)((contract_maturity - GET(Meth->t, swaption_start)) /
p->Par[0].Val.V_DOUBLE = 1.0;

for (j = jminprev ; j <= jmaxprev ; j++)
  {
    for (k = kminprev ; k <= kmaxprev ; k++)
      {
        current_u = k * delta_u2;
        current_rate = j * delta_y2 - current_u / (b - a) + GET(Meth->alpha, s

        SumZC = 0;
        for (i = 1; i <= NumberOfPayments; i++)
          {
            Ti = GET(Meth->t, swaption_start) + i * periodicity;
```

```
            ZCPrice = cf_hw2d_zcb(ZCMarket, a, sigma1, b, sigma2, rho, GET(Met
            SumZC += ZCPrice;
          }
        //SwapRate = (1-ZCPrice) / (periodicity*SumZC);

        MLET(OptionPriceMat2, j - jminprev, k - kminprev) = ((p->Compute)(p->P
      }
    }
}


/// Prix of a swaption using a trinomial tree.
static double tr_hw2d_bermudianswaption(TreeHW2D *Meth, ModelHW2D *ModelParam, Z
{
  double Ti1, Ti2, OptionPrice;
  int i, j, k, i_Ti1, i_Ti2, NumberOfPayments;

  PnlMat *OptionPriceMat1; // Matrix of prices of the option at i
  PnlMat *OptionPriceMat2; // Matrix of prices of the option at i+1
  PnlMat *PayoffMat; // Matrix of prices of the option at i

  OptionPriceMat1 = pnl_mat_create(1, 1);
  OptionPriceMat2 = pnl_mat_create(1, 1);
  PayoffMat = pnl_mat_create(1, 1);

  ///*******************Parameters of the processes r, u and y *****************
  /* a = (ModelParam->rMeanReversion); */
  /* sigma1 = (ModelParam->rVolatility); */
  /* b = (ModelParam->uMeanReversion); */
  /* sigma2 = (ModelParam->uVolatility); */
  /* rho = (ModelParam->correlation); */
  //sigma3 = sqrt(sigma1*sigma1 + sigma2*sigma2/((b-a)*(b-a)) + 2*rho*sigma1*sig

  ///**************** PAYOFF at the MATURITY of the OPTION ***************///
  Ti1 = contract_maturity - periodicity;
  i_Ti1 = indiceTimeHW2D(Meth, Ti1);

  BermudianSwaption_InitialPayoffHW2D(i_Ti1, Meth, ModelParam, ZCMarket, OptionP

  ///**************** Backward computation of the option price ***************/
  NumberOfPayments = pnl_iround((contract_maturity - option_maturity) / periodic
```

```
  for (i = NumberOfPayments - 2 ; i >= 0 ; i--)
    {
      Ti1 = option_maturity + i * periodicity;
      Ti2 = Ti1 + periodicity;
      i_Ti2 = indiceTimeHW2D(Meth, Ti2);
      i_Ti1 = indiceTimeHW2D(Meth, Ti1);

      BackwardIterationHW2D(Meth, ModelParam, ZCMarket, OptionPriceMat1, OptionP

      BermudianSwaption_InitialPayoffHW2D(i_Ti1, Meth, ModelParam, ZCMarket, Pay

      for (j = 0; j < PayoffMat->m; j++)
        {
          for (k = 0; k < PayoffMat->n; k++)
            {
              if (MGET(PayoffMat, j, k) > MGET(OptionPriceMat2, j, k))
                {
                  MLET(OptionPriceMat2, j, k) = MGET(PayoffMat, j, k);
                }
            }
        }

    }

  ///****************** Backward computation of the option price from  first_res
  i_Ti2 = indiceTimeHW2D(Meth, option_maturity);
  i_Ti1 = 0;
  BackwardIterationHW2D(Meth, ModelParam, ZCMarket, OptionPriceMat1, OptionPrice

  OptionPrice = MGET(OptionPriceMat1, 0, 0);

  pnl_mat_free(& OptionPriceMat1);
  pnl_mat_free(& OptionPriceMat2);
  pnl_mat_free(& PayoffMat);

  return OptionPrice;

}
```

```
static int tr_bermudianswaption2d(int flat_flag, double r0, char *curve, double
{
  TreeHW2D Tr;
  ModelHW2D ModelParams;
  ZCMarketData ZCMarket;

  /* Flag to decide to read or not ZC bond datas in "initialyields.dat" */
  /* If P(0,T) not read then P(0,T)=exp(-r0*T) */
  if (flat_flag == 0)
    {
      ZCMarket.FlatOrMarket = 0;
      ZCMarket.Rate = r0;
    }

  else
    {
      ZCMarket.FlatOrMarket = 1;
      ZCMarket.filename = curve;
      ReadMarketData(&ZCMarket);

      if (contract_maturity > GET(ZCMarket.tm, ZCMarket.Nvalue - 1))
        {
          printf("\ nError : time bigger than the last time value entered in ini
          exit(EXIT_FAILURE);
        }
    }

  ModelParams.rMeanReversion  = a;
  ModelParams.rVolatility     = sigma1;
  ModelParams.uMeanReversion  = b;
  ModelParams.uVolatility     = sigma2;
  ModelParams.correlation     = rho;

  if (a - b == 0)
    {
      printf("\ nError : \ "Speed of Mean Reversion Interest Rate\ " and \ "Spee
      exit(EXIT_FAILURE);
    }

  // Construction of the Time Grid
  SetTimegridHW2D(&Tr, N_steps, contract_maturity);
```

```
  // Construction of the tree, calibrated to the initial yield curve
  SetTreeHW2D(&Tr, &ModelParams, &ZCMarket);

  //Price of an option on a ZC
  *price =  Nominal * tr_hw2d_bermudianswaption(&Tr, &ModelParams, &ZCMarket, N_

  DeleteTreeHW2D(&Tr);
  DeleteZCMarketData(&ZCMarket);

  return OK;
}




///********************************************** PREMIA FUNCTIONS ***********

int CALC(TR_BERMUDIANSWAPTIONHW2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return  tr_bermudianswaption2d(ptMod->flat_flag.Val.V_INT,
                                 MOD(GetYield)(ptMod),
                                 MOD(GetCurve)(ptMod),
                                 ptMod->InitialYieldsu.Val.V_PDOUBLE,
                                 ptMod->aR.Val.V_DOUBLE,
                                 ptMod->SigmaR.Val.V_PDOUBLE,
                                 ptMod->bu.Val.V_DOUBLE,
                                 ptMod->Sigmau.Val.V_PDOUBLE,
                                 ptMod->Rho.Val.V_PDOUBLE,
                                 ptOpt->BMaturity.Val.V_DATE - ptMod->T.Val.V_DA
                                 ptOpt->OMaturity.Val.V_DATE - ptMod->T.Val.V_DA
                                 ptOpt->ResetPeriod.Val.V_DATE,
                                 ptOpt->Nominal.Val.V_PDOUBLE,
                                 ptOpt->FixedRate.Val.V_PDOUBLE,
                                 ptOpt->PayOff.Val.V_NUMFUNC_1,
                                 Met->Par[0].Val.V_INT,
                                 &(Met->Res[0].Val.V_DOUBLE));
}
```

```
static int CHK_OPT(TR_BERMUDIANSWAPTIONHW2D)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "PayerBermudanSwaption") == 0) || (strcmp((
    return OK;
  else
    return WRONG;
}
#endif //PremiaCurrentVersion


static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->Par[0].Val.V_INT2 = 100;
    }

  return OK;
}

PricingMethod MET(TR_BERMUDIANSWAPTIONHW2D) =
{
  "TR_BERMUDIANSWAPTIONHW2D",
  { {"TimeStepNumber", LONG, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_BERMUDIANSWAPTIONHW2D),
  {{"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID}*/ , {" ", PR
  CHK_OPT(TR_BERMUDIANSWAPTIONHW2D),
  CHK_ok,
  MET(Init)
} ;
```