Help

```
#include <stdlib.h>
#include "hullwhite1d_stdi.h"
#include "math/InterestRateModelTree/TreeShortRate/TreeShortRate.h"
#include "pnl/pnl_vector.h"
#include "hullwhite1d_includes.h"

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2)
static int CHK_OPT(TR_BermudianSwaptionHW1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_BermudianSwaptionHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


/// TreeShortRate     : structure that contains components of the tree (see Tree
/// ModelParameters    : structure that contains  the parameters of the Hull&Whi
/// 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
void BermudianSwaption_InitialPayoffHW1D(int swaption_start, TreeShortRate *Meth
{
  double a , sigma;

  int jminprev, jmaxprev; // jmin[i], jmax [i]
  int i, j;

  double delta_x1; // delta_x1 = space step of the process x at time i
  double delta_t1; // time step

  double ZCPrice, SwapRate, SumZC;
  double current_rate;

  int NumberOfPayments;
```

```
double Ti;

ZCPrice = 0.; /* to avoid warning */
///*******************Parameters of the process r *********************////
a = ModelParam->MeanReversion;
sigma = ModelParam->RateVolatility;

///** Calcul du vecteur des payoffs a l'instant de maturite de l'option
jminprev = pnl_vect_int_get(Meth->Jminimum, swaption_start);   // jmin(swaptio
jmaxprev = pnl_vect_int_get(Meth->Jmaximum, swaption_start);  // jmax(swaption

pnl_vect_resize(OptionPriceVect2, jmaxprev - jminprev + 1);

delta_t1 = GET(Meth->t, swaption_start) - GET(Meth->t, swaption_start - 1); //
delta_x1 = SpaceStep(delta_t1, a, sigma);   // SpaceStep(swaption_start)

NumberOfPayments = (int) floor((contract_maturity - GET(Meth->t, swaption_star


p->Par[0].Val.V_DOUBLE = 1.0;

for (j = jminprev ; j <= jmaxprev ; j++)
  {
    current_rate = func_model_hw1d(j * delta_x1 + GET(Meth->alpha, swaption_st

    SumZC = 0;
    for (i = 1; i <= NumberOfPayments; i++)
      {
        Ti = GET(Meth->t, swaption_start) + i * periodicity;
        ZCPrice = cf_hw1d_zcb(ZCMarket, a, sigma, GET(Meth->t, swaption_start)

        SumZC += ZCPrice;
      }

    SwapRate = (1 - ZCPrice) / (periodicity * SumZC);

    LET(OptionPriceVect2, j - jminprev) = ((p->Compute)(p->Par, periodicity *

    //LET(OptionPriceVect2, j-jminprev) = SumZC* periodicity*(p->Compute)(p->P
  }
```

```
}

/// Price of a bermudianswaption using a trinomial tree
double tr_hw1d_bermudianswaption(TreeShortRate *Meth, ModelParameters *ModelPara
{
  double a , sigma;
  double delta_t1; // time step
  double Pup, Pmiddle, Pdown;
  int i, j;
  double Ti2, Ti1;
  int i_Ti2, i_Ti1;
  double current_rate, NumberOfPayments;
  double OptionPrice;

  PnlVect *PayoffVect;
  PnlVect *OptionPriceVect1; // Vector of prices of the option at i
  PnlVect *OptionPriceVect2; // Vector of prices of the option at i+1
  OptionPriceVect1 = pnl_vect_create(1);
  OptionPriceVect2 = pnl_vect_create(1);
  PayoffVect = pnl_vect_create(1);


  ///*******************Parameters of the processes r, u and y ****************
  a = ModelParam->MeanReversion;
  sigma = ModelParam->RateVolatility;

  ///****************** Computation of the vector of payoff at the maturity of t
  Ti1 = contract_maturity - periodicity;
  i_Ti1 = IndexTime(Meth, Ti1);
  BermudianSwaption_InitialPayoffHW1D(i_Ti1, Meth, ModelParam, ZCMarket, OptionP

  ///****************** Backward computation of the option price until initial t

  NumberOfPayments = (int) floor((contract_maturity - option_maturity) / periodi

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

```
      BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, i_

      BermudianSwaption_InitialPayoffHW1D(i_Ti1, Meth, ModelParam, ZCMarket, Pay

      for (j = 0; j < PayoffVect->size; j++)
        {
          if (GET(PayoffVect, j) > GET(OptionPriceVect2, j))
            {
              LET(OptionPriceVect2, j) = GET(PayoffVect, j);
            }
        }

    }

  BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, i_Ti1,

  Pup = 1.0 / 6.0;
  Pmiddle = 2.0 / 3.0 ;
  Pdown = 1.0 / 6.0;

  delta_t1 = GET(Meth->t, 1) - GET(Meth->t, 0);
  current_rate = func_model_hw1d(GET(Meth->alpha, 0)); // r(0,j)
  OptionPrice = exp(-current_rate * delta_t1) * (Pup * GET(OptionPriceVect2, 2)

  pnl_vect_free(& OptionPriceVect1);
  pnl_vect_free(& OptionPriceVect2);
  pnl_vect_free(& PayoffVect);

  return OptionPrice;

}


static int tr_bermudianswaption1d(int flat_flag, double r0, char *curve, double
{
  TreeShortRate Tr;
  ModelParameters 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 (option_maturity > GET(ZCMarket.tm, ZCMarket.Nvalue - 1))
        {
          printf("\ nError : time bigger than the last time value entered in ini
          exit(EXIT_FAILURE);
        }
    }

  ModelParams.MeanReversion = a;
  ModelParams.RateVolatility = sigma;

  // Construction of the Time Grid
  SetTimeGrid_Tenor(&Tr, N_steps, option_maturity, contract_maturity, periodicit

  // Construction of the tree, calibrated to the initial yield curve
  SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_model_hw1d, &func_model_d

  *price = Nominal * tr_hw1d_bermudianswaption(&Tr, &ModelParams, &ZCMarket, N_s

  DeleteTreeShortRate(&Tr);
  DeleteZCMarketData(&ZCMarket);

  return OK;
}


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

int CALC(TR_BermudianSwaptionHW1D)(void *Opt, void *Mod, PricingMethod *Met)
```

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

  return  tr_bermudianswaption1d(ptMod->flat_flag.Val.V_INT,
                                 MOD(GetYield)(ptMod),
                                 MOD(GetCurve)(ptMod),
                                 ptMod->a.Val.V_DOUBLE,
                                 ptMod->Sigma.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_LONG,
                                 &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(TR_BermudianSwaptionHW1D)(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->HelpFilenameHint = "tr_hullwhite1d_bermudianswaption";
      Met->Par[0].Val.V_INT = 50;
    }
  return OK;
}

PricingMethod MET(TR_BermudianSwaptionHW1D) =
{
```

```
  "TR_HullWhite1d_BermudianSwaption",
  { {"TimeStepNumber per Period", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_BermudianSwaptionHW1D),
  {{"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID}*/ , {" ", PR
  CHK_OPT(TR_BermudianSwaptionHW1D),
  CHK_ok,
  MET(Init)
} ;
```