Help

```
#include <stdlib.h>
#include "
href../../../mod/hullwhite1d/hullwhite1d_stdi/hullwhite1d_stdi_h_src.pdfhullwhit
#include "
href../../../common/math/InterestRateModelTree/TreeShortRate/TreeShortRate_h_src
#include "pnl/pnl_vector.h"
#include "
href../../../mod/hullwhite1d/hullwhite1d_stdi/hullwhite1d_includes_h_src.pdfhull



#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(TR_CapFloorHW1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_CapFloorHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

/// Computation of the payoff at the final time of the tree (ie the option matur
static void CapFloor_InitialPayoffHW1D(TreeShortRate *Meth, ModelParameters *Mod
{
  double a, sigma;
  int jminprev, jmaxprev; // jmin[i], jmax [i]
  int j; //  j represents the nodes index

  double delta_x2; // delta_x1 = space step of the process x at time i ; delta_x
  double delta_t1; // time step

  double ZCPrice;
  double current_rate;
  int  i_T1;
  double periodicity;


  /// Parameters of the process r
```

```
  a = (ModelParam->MeanReversion);
  sigma = (ModelParam->RateVolatility);

  /// Computation of the vector of payoff at the maturity of the option
  periodicity = T2 - T1;
  i_T1 = IndexTime(Meth, T1);

  jminprev = pnl_vect_int_get(Meth->Jminimum, i_T1);  // jmin(i_T1)
  jmaxprev = pnl_vect_int_get(Meth->Jmaximum, i_T1);  // jmax(i_T1)

  pnl_vect_resize(OptionPriceVect2, jmaxprev - jminprev + 1);

  delta_t1 = GET(Meth->t, i_T1) - GET(Meth->t, i_T1 - 1);
  delta_x2 = SpaceStep(delta_t1, a, sigma); // delta_x (i_T1)

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

  for (j = jminprev ; j <= jmaxprev ; j++)
    {
      current_rate = func_model_hw1d(j * delta_x2  + GET(Meth->alpha, i_T1)); //

      ZCPrice = cf_hw1d_zcb(ZCMarket, a, sigma, T1, current_rate, T2);

      LET(OptionPriceVect2, j - jminprev) = (p->Compute)(p->Par, (1 + periodicit
    }

}

/// Price of a Cap/Floor using a trinomial tree
static double tr_hw1d_capfloor(TreeShortRate *Meth, ModelParameters *ModelParam,
{
  double OptionPrice, Ti2, Ti1;
  int i, i_Ti2, i_Ti1, n;

  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);


  ///*******************Parameters of the process r **********************////
```

```
  //a = ModelParam->MeanReversion;
  //sigma = ModelParam->RateVolatility;

  ///**************** PAYOFF at the MATURITY of the OPTION : T(n-1)*************
  Ti2 = contract_maturity;
  Ti1 = Ti2 - periodicity;

  CapFloor_InitialPayoffHW1D(Meth, ModelParam, ZCMarket, OptionPriceVect2, p, Ti

  ///**************** Backward computation of the option price ***************/
  n = (int)((contract_maturity - first_reset_date) / periodicity + 0.1);

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

      BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, i_

      CapFloor_InitialPayoffHW1D(Meth, ModelParam, ZCMarket, OptionPriceVect1, p

      pnl_vect_plus_vect(OptionPriceVect2, OptionPriceVect1);
    }

  ///****************** Backward computation of the option price from  first_res
  i_Ti2 = IndexTime(Meth, first_reset_date);
  i_Ti1 = 0;
  BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, i_Ti2,

  OptionPrice = GET(OptionPriceVect1, 0);

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

  return OptionPrice;

}
```

```
static int tr_capfloor1d(int flat_flag, double r0, char *curve, double a, 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 (contract_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, first_reset_date, contract_maturity - periodic

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

  *price =  Nominal * tr_hw1d_capfloor(&Tr, &ModelParams, &ZCMarket, N_steps, p,

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

```
  return OK;
}



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

int CALC(TR_CapFloorHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return  tr_capfloor1d(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_DATE,
                        ptOpt->FirstResetDate.Val.V_DATE - ptMod->T.Val.V_DATE,
                        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_CapFloorHW1D)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "Cap") == 0) || (strcmp(((Option *)Opt)->Na
    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_INT = 10;
    }
  return OK;
}


PricingMethod MET(TR_CapFloorHW1D) =
{
  "TR_HullWhite1d_CapFloor",
  { {"TimeStepNumber per Period", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_CapFloorHW1D),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(TR_CapFloorHW1D),
  CHK_ok,
  MET(Init)
} ;
```