Help

```
#include "
href../../../mod/hullwhite1dgeneralized/hullwhite1dgeneralized_stdi/hullwhite1dg

#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_vector.h"

#include "
href../../../common/math/InterestRateModelTree/TreeHW1dGeneralized/TreeHW1dGener
#include "
href../../../common/math/read_market_zc/InitialYieldCurve_h_src.pdfmath/read_mar

//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_CapFloorHW1dG)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_CapFloorHW1dG)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


static void CapFloor_InitialPayoffHW1dG(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Param
{
  int jminprev, jmaxprev;
  int j;

  double ZCPrice;

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

  pnl_vect_resize(ZCbondPriceVect, jmaxprev - jminprev + 1);

  pnl_vect_set_double(ZCbondPriceVect, 1.0); // Payoff = 1 for a ZC bond

  BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionPriceVect, ZCbondPriceVec
```

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

  for (j = 0 ; j < ZCbondPriceVect->size ; j++)
    {
      ZCPrice = GET(ZCbondPriceVect, j);

      LET(OptionPriceVect, j) = (p->Compute)(p->Par, (1 + periodicity * CapFloor
    }
}


static void CapFloor_BackwardIteration(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Parame
{
  double mean_reversion, sigma, ZCPrice;

  int jmin; // jmin[i+1], jmax[i+1]
  int jminprev, jmaxprev; // jmin[i], jmax [i]
  int i, j, k;
  double Pup, Pdown, Pmiddle;
  double delta_x1, delta_x2, beta_x;
  double delta_t1, delta_t2;
  double current_rate;


  ///*******************Parameters of the process r **********************////
  mean_reversion = (HW1dG_Parameters->MeanReversion);

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

  pnl_vect_resize(ZCbondPriceVect2, OptionPriceVect2->size);

  pnl_vect_set_double(ZCbondPriceVect2, 1.0); // Payoff = 1 for a ZC bond

  for (i = index_last - 1; i >= index_first; i--)
    {
      jmin = jminprev; // jmin := jmin(i+1)
      //jmax = jmaxprev; // jmax := jmax(i+1)

      jminprev = pnl_vect_int_get(Meth->Jminimum, i);  // jmin(i)
```

```
  jmaxprev = pnl_vect_int_get(Meth->Jmaximum, i);  // jmax(i)

  pnl_vect_resize(OptionPriceVect1, jmaxprev - jminprev + 1); // OptionPrice
  pnl_vect_resize(ZCbondPriceVect1, jmaxprev - jminprev + 1);

  delta_t1 = GET(Meth->t, i) - GET(Meth->t, i - 1); // Time step between t[i
  delta_t2 = GET(Meth->t, i + 1) - GET(Meth->t, i); // Time step between t[i

  sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(Meth->t, i)); // sig
  delta_x1 = SpaceStepHW1dG(delta_t1, sigma);//SpaceStepHW1dG(delta_t1, a, s

  sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(Meth->t, i + 1)); //
  delta_x2 = SpaceStepHW1dG(delta_t2, sigma);//SpaceStepHW1dG(delta_t2, a, s

  beta_x = delta_x1 / delta_x2;

  // Loop over the node at the time i
  for (j = jminprev ; j <= jmaxprev ; j++)
    {
      k = pnl_iround(j * beta_x * exp(-delta_t2 * mean_reversion)); //h inde

      // Probability to go from (i,j) to (i+1,k+1) with an UP movement
      Pup = ProbaUpHW1dG(j, k, delta_t2, beta_x, mean_reversion);
      // Probability to go from (i,j) to (i+1,k) with a Middle movement
      Pmiddle = ProbaMiddleHW1dG(j, k, delta_t2, beta_x, mean_reversion);
      // Probability to go from (i,j) to (i+1,k-1) with a Down movement
      Pdown = 1 - Pup - Pmiddle;

      current_rate = j * delta_x1 + GET(Meth->alpha, i); // r(i,j)

      LET(OptionPriceVect1, j - jminprev) = exp(-current_rate * delta_t2) *

      LET(ZCbondPriceVect1, j - jminprev) = exp(-current_rate * delta_t2) *

    }
  // Copy OptionPrice1 in OptionPrice2
  pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);
  pnl_vect_clone(ZCbondPriceVect2, ZCbondPriceVect1);

} // END of the loop on i
```

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

  for (j = 0 ; j < ZCbondPriceVect2->size ; j++)
    {
      ZCPrice = GET(ZCbondPriceVect2, j);

      LET(OptionPriceVect2, j) += (p->Compute)(p->Par, (1 + periodicity * CapFlo
    }

}

/// Price of a Cap/Floor using a trinomial tree

static double tr_hw1dg_capfloor(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Parameters, Z
{

  int i, i_Ti2, i_Ti1, n;
  double Ti2, Ti1, delta_t1, current_rate, OptionPrice, Pup, Pdown, Pmiddle;

  PnlVect *OptionPriceVect1; // Vector of prices of the option at i
  PnlVect *OptionPriceVect2; // Vector of prices of the option at i+1
  PnlVect *ZCbondPriceVect1; // Vector of prices of the option at i+1
  PnlVect *ZCbondPriceVect2; // Vector of prices of the option at i+1
  OptionPriceVect1 = pnl_vect_create(1);
  OptionPriceVect2 = pnl_vect_create(1);
  ZCbondPriceVect1 = pnl_vect_create(1);
  ZCbondPriceVect2 = pnl_vect_create(1);

  // Mmean reversion parameter
  //a = HW1dG_Parameters->MeanReversion;

  ///**************** PAYOFF at the MATURITY of the OPTION : T(n-1)************
  Ti2 = contract_maturity;
  Ti1 = Ti2 - periodicity;
  i_Ti1 = IndexTimeHW1dG(Meth, Ti1);

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

  CapFloor_InitialPayoffHW1dG(Meth, HW1dG_Parameters, ZCbondPriceVect2, OptionPr
```

```
  ///**************** 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; // Ti1 = T(i)
      Ti2 = Ti1 + periodicity;                  // Ti2 = T(i+1)

      i_Ti2 = IndexTimeHW1dG(Meth, Ti2);
      i_Ti1 = IndexTimeHW1dG(Meth, Ti1);

      CapFloor_BackwardIteration(Meth, HW1dG_Parameters, p, ZCbondPriceVect1, ZC
    }

  ///****************** Price of the option at initial time s *****************

  BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionPriceVect1, OptionPriceVe

  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); // Pas de temps entre t[1] et t[
  current_rate = GET(Meth->alpha, 0); // r(i,j)
  OptionPrice = exp(-current_rate * delta_t1) * (Pup * GET(OptionPriceVect1, 2)

  pnl_vect_free(& OptionPriceVect1);
  pnl_vect_free(& OptionPriceVect2);
  pnl_vect_free(& ZCbondPriceVect1);
  pnl_vect_free(& ZCbondPriceVect2);

  return OptionPrice;

}

static int tr_capfloor1d(int flat_flag, double r0, char *curve, int CapletCurve,
{
  TreeHW1dG Tr;
  ModelHW1dG HW1dG_Parameters;
  ZCMarketData ZCMarket;
```

```
  MktATMCapletVolData MktATMCapletVol;

  // Read the interest rate term structure from file, or set it flat
  if (flat_flag == 0)
    {
      ZCMarket.FlatOrMarket = 0;
      ZCMarket.Rate = r0;
    }

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

  // Read the caplet volatilities from file "impliedcapletvol.dat".
  ReadCapletMarketData(&MktATMCapletVol, CapletCurve);

  hw1dg_calibrate_volatility(&HW1dG_Parameters, &ZCMarket, &MktATMCapletVol, a);

  // Construction of the Time Grid
  SetTimeGrid_TenorHW1dG(&Tr, N_steps, first_reset_date, contract_maturity, peri

  // Construction of the tree, calibrated to the initial yield curve
  SetTreeHW1dG(&Tr, &HW1dG_Parameters, &ZCMarket);

  *price =  Nominal * tr_hw1dg_capfloor(&Tr, &HW1dG_Parameters, &ZCMarket, N_ste

  DeleteTreeHW1dG(&Tr);
  DeleteZCMarketData(&ZCMarket);
  DeleteMktATMCapletVolData(&MktATMCapletVol);
  DeletModelHW1dG(&HW1dG_Parameters);

  return OK;
}



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

```
int CALC(TR_CapFloorHW1dG)(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->CapletCurve.Val.V_ENUM.value,
                        ptMod->a.Val.V_DOUBLE,
                        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_CapFloorHW1dG)(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->HelpFilenameHint = "tr_hullwhite1dgeneralized_capfloor";
      Met->Par[0].Val.V_INT = 20;
    }
  return OK;
}
```

```
PricingMethod MET(TR_CapFloorHW1dG) =
{
  "TR_HullWhite1dG_CapFloor",
  { {"TimeStepNumber per Period", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(TR_CapFloorHW1dG),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(TR_CapFloorHW1dG),
  CHK_ok,
  MET(Init)
} ;
```