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

//The "#else" part of the code will be freely available aft
    er the (year of creation of this file + 2)
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+2)
static int CHK_OPT(TR_BermudianSwaptionHW1D)(void *Opt, voi
    d *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 TreeShortRate.h)
/// ModelParameters    : structure that contains  the para
    meters of the Hull&White one factor model (see TreeShortRate.
    h)
/// ZCMarketData : structure that contains the Zero Coupon
    Bond prices of the market, or given by a constant yield-to-
    maturity (see InitialYieldCurve.h)

/// Computation of the payoff at the final time of the tre
    e (ie the option maturity)
void BermudianSwaption_InitialPayoffHW1D(int swaption_star
    t, TreeShortRate* Meth, ModelParameters* ModelParam, ZCMar
    ketData* ZCMarket, PnlVect* OptionPriceVect2, NumFunc_1 *p,
     double periodicity,double contract_maturity, double Swapt
    ionFixedRate)
```

```
{
  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, 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 matu
    rite de l'option
  jminprev = pnl_vect_int_get(Meth->Jminimum, swaption_star
    t);   // jmin(swaption_start)
  jmaxprev = pnl_vect_int_get(Meth->Jmaximum, swaption_star
    t);  // jmax(swaption_start)

  pnl_vect_resize(OptionPriceVect2, jmaxprev-jminprev+1);

  delta_t1 = GET(Meth->t, swaption_start) - GET(Meth->t,swa
    ption_start-1); // Pas de temps entre t[swaption_start-1]
    et t[swaption_start]
  delta_x1 = SpaceStep(delta_t1, a, sigma);   // SpaceStep(
    swaption_start)

  NumberOfPayments = intapprox((contract_maturity-GET(Meth-
    >t, swaption_start) )/periodicity);

  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_start)); // rate(Ngrid, j )

      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), current_rate, Ti); // P(option_matu
    rity, Ti)

          SumZC += ZCPrice;
        }

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

      LET(OptionPriceVect2, j-jminprev) = ((p->Compute)(p->
    Par, periodicity * SwaptionFixedRate * SumZC + ZCPrice));

      //LET(OptionPriceVect2, j-jminprev) = SumZC* periodic
    ity*(p->Compute)(p->Par, -SwapRate);
    }

}

/// Price of a bermudianswaption using a trinomial tree
double tr_hw1d_bermudianswaption(TreeShortRate* Meth,
    ModelParameters* ModelParam, ZCMarketData* ZCMarket,int NumberO
    fTimeStep, NumFunc_1 *p, double periodicity,double option_
    maturity,double contract_maturity, double SwaptionFixedRate)
{
  double Ti2, Ti1, OptionPrice;
  int i,j, i_Ti2, i_Ti1;
  int NumberOfPayments;

  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);

///****************** Computation of the vector of payo
  ff at the maturity of the option ********************///
Ti1 = contract_maturity-periodicity;
i_Ti1 = IndexTime(Meth, Ti1);
BermudianSwaption_InitialPayoffHW1D(i_Ti1, Meth, ModelPar
  am, ZCMarket, OptionPriceVect2, p, periodicity, contract_
  maturity, SwaptionFixedRate);

///****************** Backward computation of the option
  price until initial time s ********************///

NumberOfPayments = intapprox((contract_maturity-option_
  maturity )/periodicity);

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_Ti2, i_Ti1, &func_model_hw1d);

    BermudianSwaption_InitialPayoffHW1D(i_Ti1, Meth,
  ModelParam, ZCMarket, PayoffVect, p, periodicity, contract_matu
  rity, SwaptionFixedRate);

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

```
        }

    }

  ///****************** Backward computation of the option
    price from  first_reset_date to 0 ********************///
  i_Ti2 = IndexTime(Meth, option_maturity);
  i_Ti1 = 0;
  BackwardIteration(Meth, ModelParam, OptionPriceVect1,
    OptionPriceVect2, i_Ti2, i_Ti1, &func_model_hw1d);

  OptionPrice = GET(OptionPriceVect1, 0);

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

  return OptionPrice;

}


static int tr_bermudianswaption1d(int flat_flag,double r0,
    double a,double sigma, double contract_maturity, double option_
    maturity, double periodicity,double Nominal, double SwaptionF
    ixedRate, NumFunc_1 *p, int N_steps, double *price)
{
  TreeShortRate Tr;
  ModelParameters ModelParams;
  ZCMarketData ZCMarket;

  /* Flag to decide to read or not ZC bond datas in "initia
    lyields.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;
      ReadMarketData(&ZCMarket);

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

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

  // Construction of the Time Grid
  SetTimeGrid_Tenor(&Tr, N_steps, option_maturity, contrac
    t_maturity, periodicity);

  // Construction of the tree, calibrated to the initial yi
    eld curve
  SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_
    model_hw1d, &func_model_der_hw1d, &func_model_inv_hw1d);

  *price = Nominal * tr_hw1d_bermudianswaption(&Tr, &ModelP
    arams, &ZCMarket, N_steps, p, periodicity, option_maturity,
     contract_maturity, SwaptionFixedRate);

  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),
                                 ptMod->a.Val.V_DOUBLE,
                                 ptMod->Sigma.Val.V_PDOUB
    LE,
                                 ptOpt->BMaturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                 ptOpt->OMaturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,
                                 ptOpt->ResetPeriod.Val.V_
    DATE,
                                 ptOpt->Nominal.Val.V_PDOUB
    LE,
                                 ptOpt->FixedRate.Val.V_PDO
    UBLE,
                                 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, voi
    d *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"PayerBermudanSwaption")
    ==0) || (strcmp(((Option*)Opt)->Name,"
    ReceiverBermudanSwaption")==0))
    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=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},FO
    RBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_BermudianSwaptionHW1D),
  CHK_ok,
  MET(Init)
} ;
```

References