```
   Help
#include <stdlib.h>
#include "hullwhite1d_stdi.h"
#include "math/InterestRateModelTree/TreeShortRate/TreeSho
    rtRate.h"
#include "pnl/pnl_vector.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_SwaptionHW1D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_SwaptionHW1D)(void *Opt,void *Mod,Pricing
    Method *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 Swaption_InitialPayoffHW1D(TreeShortRate* Meth, ModelP
    arameters* ModelParam, ZCMarketData* ZCMarket, PnlVect*
    OptionPriceVect2, NumFunc_1 *p, double periodicity,double
    option_maturity,double contract_maturity, double SwaptionFixedRa
    te)
{
```

```
double a ,sigma;

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

double delta_x1; // delta_x1 = space step of the proces
s 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, Meth->Ngrid
);   // jmin(Ngrid)
jmaxprev = pnl_vect_int_get(Meth->Jmaximum, Meth->Ngrid
);  // jmax(Ngrid)

pnl_vect_resize(OptionPriceVect2, jmaxprev-jminprev+1);

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

NumberOfPayments = (int) floor((contract_maturity-
option_maturity )/periodicity + 0.2);

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, Meth->Ngrid)); // rate(Ngrid, j )

        SumZC = 0;
        for(i=1; i<=NumberOfPayments; i++)
        {
            Ti = option_maturity + i*periodicity;
            ZCPrice = cf_hw1d_zcb(ZCMarket, a, sigma,
    option_maturity, current_rate, Ti); // P(option_maturity, 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* perio
    dicity*(p->Compute)(p->Par, -SwapRate);
    }

}

/// Price of a swaption using a trinomial tree
double tr_hw1d_swaption(TreeShortRate* Meth, ModelParamet
    ers* ModelParam, ZCMarketData* ZCMarket,int NumberOfTimeStep
    , NumFunc_1 *p, double r, double periodicity,double
    option_maturity,double contract_maturity, double SwaptionFixedRa
    te)
{
    int index_last, index_first;
    double OptionPrice;

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

    ///****************** Computation of the vector of payo
    ff at the maturity of the option ********************///
    Swaption_InitialPayoffHW1D(Meth, ModelParam, ZCMarket,
    OptionPriceVect2, p, periodicity, option_maturity, contract_matu
    rity, SwaptionFixedRate);

    ///****************** Backward computation of the
    option price until initial time s ********************///
    index_last = Meth->Ngrid;
    index_first = 0;

    BackwardIteration(Meth, ModelParam, OptionPriceVect1,
    OptionPriceVect2, index_last, index_first, &func_model_hw1d);

    OptionPrice = GET(OptionPriceVect1, 0);

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

    return OptionPrice;

}


static int tr_swaption1d(int flat_flag,double r0,double a,
    double sigma, double contract_maturity, double option_maturity,
    double periodicity,double Nominal, double SwaptionFixedRate,
    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 "ini
```

```
    tialyields.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.Nvalu
    e-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(&Tr, N_steps, option_maturity);

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

    *price = Nominal * tr_hw1d_swaption(&Tr, &ModelParams,
    &ZCMarket, N_steps, p, r0, periodicity, option_maturity,
    contract_maturity, SwaptionFixedRate);

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

  return OK;
```

```
}


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

int CALC(TR_SwaptionHW1D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return  tr_swaption1d(ptMod->flat_flag.Val.V_INT,
                        MOD(GetYield)(ptMod),
                        ptMod->a.Val.V_DOUBLE,
                        ptMod->Sigma.Val.V_PDOUBLE,
                        ptOpt->BMaturity.Val.V_DATE-ptMod->
    T.Val.V_DATE,
                        ptOpt->OMaturity.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_SwaptionHW1D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option*)Opt)->Name,"PayerSwaption")==0) |
    | (strcmp(((Option*)Opt)->Name,"ReceiverSwaption")==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_LONG=200;
    }

  return OK;
}

PricingMethod MET(TR_SwaptionHW1D)=
{
  "TR_HullWhite1d_Swaption",
  {{"TimeStepNumber",LONG,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_SwaptionHW1D),
  {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},FO
    RBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_SwaptionHW1D),
  CHK_ok,
  MET(Init)
} ;
```

References