```
   Help
#include "hullwhite2d_stdi.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "math/InterestRateModelTree/TreeHW2D/TreeHW2D.h"
#include "hullwhite2d_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 <
     (2009+2)
int CALC(TR_SWAPTIONHW2D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(TR_SWAPTIONHW2D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
#else

/// TreeHW2D     : structure that contains components of th
    e tree (see ModelHW2D.h)
/// ModelHW2D    : structure that contains  the parameters
    of the Hull&White one factor model (see ModelHW2D.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)
static void Swaption_InitialPayoff(TreeHW2D* Meth, ModelHW2
    D* ModelParam, ZCMarketData* ZCMarket,PnlMat* OptionPriceM
    at2, NumFunc_1 *p, double periodicity,double option_maturit
    y,double contract_maturity, double SwaptionFixedRate)
{
  double a ,sigma1, b, sigma2, rho,sigma3;

    int jminprev, jmaxprev, kminprev, kmaxprev; // jmin[i],
 jmax [i]
int i, j, k, NumberOfPayments; // i = represents the
time index. j, k represents the nodes index

double delta_y2; // delta_y1 = space step of the proces
s y at time i ; delta_y2 same at time i+1.
double delta_u2; // delta_u1 = space step of the proces
s u at time i ; delta_u2 same at time i+1.
double delta_t1; // time step

double ZCPrice,SumZC; //ZC price
double current_rate, current_u;
double Ti;


ZCPrice = 0.;
// Parameters of the processes r, u and y
a = (ModelParam->rMeanReversion);
sigma1 = (ModelParam->rVolatility);

b = (ModelParam->uMeanReversion);
sigma2 = (ModelParam->uVolatility);

rho = (ModelParam->correlation);

sigma3 = sqrt(sigma1*sigma1 + sigma2*sigma2/((b-a)*(b-
a)) + 2*rho*sigma1*sigma2 / (b-a) );


// Computation of the vector of payoff at the maturity
of the option
jminprev = pnl_vect_int_get(Meth->yIndexMin, Meth->Ng
rid);  // jmin(Ngrid)
jmaxprev = pnl_vect_int_get(Meth->yIndexMax, Meth->Ng
rid);  // jmax(Ngrid)
kminprev = pnl_vect_int_get(Meth->uIndexMin, Meth->Ng
rid);  // kmin(Ngrid)
kmaxprev = pnl_vect_int_get(Meth->uIndexMax, Meth->Ng
rid);  // kmax(Ngrid)

pnl_mat_resize(OptionPriceMat2, jmaxprev-jminprev+1, km
    axprev-kminprev+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_y2 = delta_xHW2D(delta_t1, a, sigma3); // delta_
    y (Ngrid)
    delta_u2 = delta_xHW2D(delta_t1, b, sigma2); // delta_
    u (Ngrid)

    NumberOfPayments = (int) ((contract_maturity-option_
    maturity)/periodicity);
    p->Par[0].Val.V_DOUBLE = 1.0;

    for( j = jminprev ; j<=jmaxprev ; j++)
    {
        for( k = kminprev ; k<=kmaxprev ; k++)
        {
            current_u = k * delta_u2;
            current_rate = j * delta_y2 - current_u/(b-a) +
     GET(Meth->alpha, Meth->Ngrid); // rate(Ngrid,j, k)

            SumZC = 0;
            for(i=1; i<=NumberOfPayments; i++)
            {
                Ti = option_maturity + i*periodicity;
                ZCPrice = cf_hw2d_zcb(ZCMarket, a, sigma1,
    b, sigma2, rho, option_maturity, current_rate, current_u,
    Ti); // P(option_maturity, Ti)
                SumZC += ZCPrice;
            }
            //SwapRate = (1-ZCPrice) / (periodicity*SumZC);

            MLET(OptionPriceMat2, j-jminprev, k-kminprev) =
     ((p->Compute)(p->Par, periodicity * SwaptionFixedRate *
    SumZC + ZCPrice));
        }
    }
}


/// Prix of a swaption using a trinomial tree.
```

```
static double tr_hw2d_swaption(TreeHW2D* Meth, ModelHW2D*
    ModelParam, ZCMarketData* ZCMarket, int NumberOfTimeStep,
    NumFunc_1 *p, double r, double u, double periodicity,double
    option_maturity,double contract_maturity, double SwaptionFixedRa
    te)
{
    double a ,sigma1, b, sigma2, rho, sigma3,OptionPrice;

    PnlMat* OptionPriceMat1; // Matrix of prices of the
    option at i
    PnlMat* OptionPriceMat2; // Matrix of prices of the
    option at i+1

    OptionPriceMat1 = pnl_mat_create(1,1);
    OptionPriceMat2 = pnl_mat_create(1,1);

    ///******************Parameters of the processes r, u
    and y *********************////
    a = (ModelParam->rMeanReversion);
    sigma1 = (ModelParam->rVolatility);

    b = (ModelParam->uMeanReversion);
    sigma2 = (ModelParam->uVolatility);

    rho = (ModelParam->correlation);

    sigma3 = sqrt(sigma1*sigma1 + sigma2*sigma2/((b-a)*(b-
    a)) + 2*rho*sigma1*sigma2 / (b-a) );


    ///**************** PAYOFF at the MATURITY of the
    OPTION ***************///
    Swaption_InitialPayoff(Meth, ModelParam, ZCMarket,
    OptionPriceMat2, p, periodicity, option_maturity, contract_matu
    rity, SwaptionFixedRate);

    ///**************** Backward computation of the option
    price ***************///
    BackwardIterationHW2D(Meth, ModelParam, ZCMarket,
    OptionPriceMat1, OptionPriceMat2, Meth->Ngrid, 0);
    ///****************** Price of the option at time 0 ***
    ****************///
    OptionPrice = MGET(OptionPriceMat2, 0, 0);

    pnl_mat_free(& OptionPriceMat1);
    pnl_mat_free(& OptionPriceMat2);

    return OptionPrice;

}


static int tr_swaption2d(int flat_flag,double r0,double u0,
    double a,double sigma1,double b,double sigma2,double rho,
    double contract_maturity,double option_maturity, double periodic
    ity,double Nominal, double SwaptionFixedRate, NumFunc_1 *p,
    int N_steps, double *price)
{
  TreeHW2D Tr;
  ModelHW2D 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(contract_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.rMeanReversion  = a;
  ModelParams.rVolatility     = sigma1;
  ModelParams.uMeanReversion  = b;
  ModelParams.uVolatility     = sigma2;
  ModelParams.correlation     = rho;

  if(a-b==0)
  {
      printf("{nError : {"Speed of Mean Reversion Interest
    Rate{" and {"Speed of Mean Reversion of u{" must be diffe
    rents! {n");
      exit(EXIT_FAILURE);
  }

  // Construction of the Time Grid
  SetTimegridHW2D(&Tr, N_steps, option_maturity);

  // Construction of the tree, calibrated to the initial yi
    eld curve
  SetTreeHW2D(&Tr, &ModelParams, &ZCMarket);

  //Price of an option on a ZC
  *price =  Nominal * tr_hw2d_swaption(&Tr, &ModelParams, &
    ZCMarket, N_steps, p, r0, u0, periodicity, option_maturity,
    contract_maturity, SwaptionFixedRate);

  DeleteTreeHW2D(&Tr);
  DeleteZCMarketData(&ZCMarket);

  return OK;
}
```

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

int CALC(TR_SWAPTIONHW2D)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return  tr_swaption2d(    ptMod->flat_flag.Val.V_INT,
                            MOD(GetYield)(ptMod),
                            ptMod->InitialYieldsu.Val.V_PDO
    UBLE,
                            ptMod->aR.Val.V_DOUBLE,
                            ptMod->SigmaR.Val.V_PDOUBLE,
                            ptMod->bu.Val.V_DOUBLE,
                            ptMod->Sigmau.Val.V_PDOUBLE,
                            ptMod->Rho.Val.V_PDOUBLE,
                            ptOpt->BMaturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                            ptOpt->OMaturity.Val.V_DATE-pt
    Mod->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_INT,
                            &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(TR_SWAPTIONHW2D)(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_INT2=100;
    }

  return OK;
}

PricingMethod MET(TR_SWAPTIONHW2D)=
{
  "TR_SWAPTIONHW2D",
  {{"TimeStepNumber",LONG,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_SWAPTIONHW2D),
  {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},FO
    RBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_SWAPTIONHW2D),
  CHK_ok,
  MET(Init)
} ;
```

References