```
   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_BERMUDIANSWAPTIONHW2D)(void *Opt,void *Mod,
    PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(TR_BERMUDIANSWAPTIONHW2D)(void *Opt, voi
    d *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 BermudianSwaption_InitialPayoffHW2D(int swaptio
    n_start, TreeHW2D* Meth, ModelHW2D* ModelParam, ZCMarketD
    ata* ZCMarket,PnlMat* OptionPriceMat2, NumFunc_1 *p, double
    periodicity,double contract_maturity, double SwaptionFixed
    Rate)
{
    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, swaption_
start);  // jmin(swaption_start)
jmaxprev = pnl_vect_int_get(Meth->yIndexMax, swaption_
start);  // jmax(swaption_start)
kminprev = pnl_vect_int_get(Meth->uIndexMin, swaption_
start);  // kmin(swaption_start)
kmaxprev = pnl_vect_int_get(Meth->uIndexMax, swaption_
start);  // kmax(swaption_start)

pnl_mat_resize(OptionPriceMat2, jmaxprev-jminprev+1, km
```

```
    axprev-kminprev+1);

    delta_t1 = GET(Meth->t, swaption_start) - GET(Meth->t,
    swaption_start-1); // Pas de temps entre t[swaption_start-1]
     et tswaption_start]
    delta_y2 = delta_xHW2D(delta_t1, a, sigma3); // delta_
    y (swaption_start)
    delta_u2 = delta_xHW2D(delta_t1, b, sigma2); // delta_
    u (swaption_start)

    NumberOfPayments = (int) ((contract_maturity-GET(Meth->
    t, swaption_start))/periodicity + 0.2);
    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, swaption_start); // rate(Ngrid,j, k)

            SumZC = 0;
            for (i=1; i<=NumberOfPayments; i++)
            {
                Ti = GET(Meth->t, swaption_start) + i*perio
    dicity;
                ZCPrice = cf_hw2d_zcb(ZCMarket, a, sigma1,
    b, sigma2, rho, GET(Meth->t, swaption_start), 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_bermudianswaption(TreeHW2D* Meth,
    ModelHW2D* ModelParam, ZCMarketData* ZCMarket, int NumberOfTi
    meStep, NumFunc_1 *p, double r, double u, double periodicity
    ,double option_maturity,double contract_maturity, double
    SwaptionFixedRate)
{
    double a , sigma1, b, sigma2, rho;
    double Ti1, Ti2, OptionPrice;
    int i, j, k, i_Ti1, i_Ti2, NumberOfPayments;

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

    OptionPriceMat1 = pnl_mat_create(1,1);
    OptionPriceMat2 = pnl_mat_create(1,1);
    PayoffMat = 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 ****************///
    Ti1 = contract_maturity-periodicity;
    i_Ti1 = indiceTimeHW2D(Meth, Ti1);
```

```
BermudianSwaption_InitialPayoffHW2D(i_Ti1, Meth, ModelP
aram, ZCMarket, OptionPriceMat2, p, periodicity, contract_
maturity, SwaptionFixedRate);

///**************** Backward computation of the option
price ****************///
NumberOfPayments = intapprox((contract_maturity-option_
maturity )/periodicity);

for (i=NumberOfPayments-2 ; i>=0 ; i--)
{
    Ti1 = option_maturity + i * periodicity;
    Ti2 = Ti1 + periodicity;
    i_Ti2 = indiceTimeHW2D(Meth, Ti2);
    i_Ti1 = indiceTimeHW2D(Meth, Ti1);

    BackwardIterationHW2D(Meth, ModelParam, ZCMarket,
OptionPriceMat1, OptionPriceMat2, i_Ti2, i_Ti1);

    BermudianSwaption_InitialPayoffHW2D(i_Ti1, Meth,
ModelParam, ZCMarket, PayoffMat, p, periodicity, contract_matu
rity, SwaptionFixedRate);

    for (j=0;j<PayoffMat->m;j++)
    {
        for (k=0;k<PayoffMat->n;k++)
        {
            if (MGET(PayoffMat,j,k)>MGET(OptionPriceM
at2,j,k))
            {
                MLET(OptionPriceMat2,j,k) = MGET(Payo
ffMat,j,k);
            }
        }
    }

}

///****************** Backward computation of the
option price from  first_reset_date to 0 ******************///
i_Ti2 = indiceTimeHW2D(Meth, option_maturity);
```

```
    i_Ti1 = 0;
    BackwardIterationHW2D(Meth, ModelParam, ZCMarket,
    OptionPriceMat1, OptionPriceMat2, i_Ti2, i_Ti1);

    OptionPrice = MGET(OptionPriceMat1, 0, 0);

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

    return OptionPrice;

}


static int tr_bermudianswaption2d(int flat_flag,double r0,
    double u0,double a,double sigma1,double b,double sigma2,double
    rho, double contract_maturity,double option_maturity,
    double periodicity,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 "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 (contract_maturity > GET(ZCMarket.tm,ZCMarket.Nv
    alue-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 Intere
    st 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, contract_maturity);

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

    //Price of an option on a ZC
    *price =  Nominal * tr_hw2d_bermudianswaption(&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_BERMUDIANSWAPTIONHW2D)(void *Opt,void *Mod,
    PricingMethod *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;

    return  tr_bermudianswaption2d(    ptMod->flat_flag.Val
    .V_INT,
                                       MOD(GetYield)(ptMod)
    ,
                                       ptMod->InitialYield
    su.Val.V_PDOUBLE,
                                       ptMod->aR.Val.V_
    DOUBLE,
                                       ptMod->SigmaR.Val.V_
    PDOUBLE,
                                       ptMod->bu.Val.V_
    DOUBLE,
                                       ptMod->Sigmau.Val.V_
    PDOUBLE,
                                       ptMod->Rho.Val.V_PDO
    UBLE,
                                       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_
    INT,
                                       &(Met->Res[0].Val.V_
```

```
    DOUBLE));
}

static int CHK_OPT(TR_BERMUDIANSWAPTIONHW2D)(void *Opt, voi
    d *Mod)
{
    if ((strcmp(((Option*)Opt)->Name,"PayerBermudanSwaptio
    n")==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_INT2=100;
    }

    return OK;
}

PricingMethod MET(TR_BERMUDIANSWAPTIONHW2D)=
{
    "TR_BERMUDIANSWAPTIONHW2D",
    {{"TimeStepNumber",LONG,{100},ALLOW},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(TR_BERMUDIANSWAPTIONHW2D),
    {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},
    FORBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(TR_BERMUDIANSWAPTIONHW2D),
    CHK_ok,
    MET(Init)
} ;
```

References