```
   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_ZBOHW2D)(void *Opt,void *Mod,PricingMethod *
    Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(TR_ZBOHW2D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
#else

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

    int jminprev, jmaxprev, kminprev, kmaxprev; // jmin[i],
     jmax [i]
```

```
int j, k; // i = represents the time index. j, k repres
ents 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 A_tT, B_tT, C_tT, ZCPrice; //ZC price
double current_rate, current_u;
double T;

A_tT=0;
B_tT=0;
C_tT=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) );

T = (Meth->Tf);
///** Computation of the vector of payoff at the matu
rity 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)

    ZCPrice_Coefficient(ZCMarket, a, sigma1, b, sigma2, rh
    o, T, S, &A_tT , &B_tT, &C_tT); //A_tT, B_tT, C_tT : Coeffi
    cients used in the calculation of the Zero Coupon price

    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)

            ZCPrice = ZCPrice_Using_Coefficient(current_ra
    te, current_u , A_tT, B_tT, C_tT);

            MLET(OptionPriceMat2, j-jminprev, k-kminprev) =
     (p->Compute)(p->Par, ZCPrice); //Payoff(ZCPrice, X, Call_
    Or_Put);
        }
    }

}

/// Price of an option on a ZC using a trinomial tree.
static double tr_hw2d_zcoption(TreeHW2D* Meth, ModelHW2D*
    ModelParam, ZCMarketData* ZCMarket, double T, double S, int
    NumberOfTimeStep, NumFunc_1 *p, double r, double u, int Eur_
    Or_Am)
{
    double a ,sigma1, b, sigma2, rho, sigma3;
    double delta_t1, current_rate, current_u, OptionPrice,
```

```
delta_y1, delta_u1, A_tT, B_tT, C_tT, ZCPrice;
int i, h, l, jmin, jmax, kmin, kmax;

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);

A_tT=0;
B_tT=0;
C_tT=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 payo
ff at the maturity of the option *******************///
ZCOption_InitialPayoff(Meth, ModelParam, ZCMarket,
OptionPriceMat2, p, S);

///****************** Backward computation of the
option price until time 0 *******************///
for (i = Meth->Ngrid-1; i>=0; i--)
{
    BackwardIterationHW2D(Meth, ModelParam, ZCMarket,
OptionPriceMat1, OptionPriceMat2, i+1, i);

    if (Eur_Or_Am != 0)
    {
        jmin = pnl_vect_int_get(Meth->yIndexMin, i);  /
```

```
/ jmin(i)
        jmax = pnl_vect_int_get(Meth->yIndexMax, i);  /
/ jmax(i)
        kmin = pnl_vect_int_get(Meth->uIndexMin, i);  /
/ kmin(i)
        kmax = pnl_vect_int_get(Meth->uIndexMax, i);  /
/ kmax(i)

        delta_t1 = GET(Meth->t, i) - GET(Meth->t,MAX(i-
1,0)); // time step. if i=0, then delta=0 in order to ha
ve delta_x=0.
        delta_y1 = delta_xHW2D(delta_t1, a, sigma3); //
 space step 1
        delta_u1 = delta_xHW2D(delta_t1, b, sigma2); //
 space step 2

        ZCPrice_Coefficient(ZCMarket, a, sigma1, b, si
gma2, rho, GET(Meth->t, i), S, &A_tT , &B_tT, &C_tT);

        for (h= jmin ; h <=jmax ; h++)
        {
            for (l= kmin ; l <=kmax ; l++)
            {
                current_u = l*delta_u1;
                current_rate = GET(Meth->alpha, i) + h*
delta_y1 - current_u/(b-a);

                ZCPrice = ZCPrice_Using_Coefficient(
current_rate, current_u, A_tT, B_tT, C_tT);

                // Decide whether to exercise the
option or not
                if ( MGET(OptionPriceMat2, h-jmin, l-km
in) < (p->Compute)(p->Par, ZCPrice))
                {
                    MLET(OptionPriceMat2, h-jmin, l-km
in) = (p->Compute)(p->Par, ZCPrice);
                }
            }
        }
    }
```

```
    }

    OptionPrice = MGET(OptionPriceMat2, 0, 0);

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

    return OptionPrice;

}// FIN de la fonction ZCOption


static int tr_zbo2d(int flat_flag,double r0,double u0,
    double a,double sigma1,double b,double sigma2,double rho,
                    double S,double T,NumFunc_1 *p,int am,
    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 (S > GET(ZCMarket.tm,ZCMarket.Nvalue-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, T);

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

    //Price of an option on a ZC
    *price = tr_hw2d_zcoption(&Tr, &ModelParams, &ZCMarket,
     T, S, N_steps, p, r0, u0, am);


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

    return OK;
}




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

```
int CALC(TR_ZBOHW2D)(void *Opt,void *Mod,PricingMethod *
    Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;

    return  tr_zbo2d( ptMod->flat_flag.Val.V_INT,
                      MOD(GetYield)(ptMod),
                      ptMod->InitialYieldsu.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_PDOUBLE,
                      ptOpt->BMaturity.Val.V_DATE-ptMod->T.
    Val.V_DATE,
                      ptOpt->OMaturity.Val.V_DATE-ptMod->T.
    Val.V_DATE,
                      ptOpt->PayOff.Val.V_NUMFUNC_1,
                      ptOpt->EuOrAm.Val.V_BOOL,
                      Met->Par[0].Val.V_INT,
                      &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(TR_ZBOHW2D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option*)Opt)->Name,"ZeroCouponCallBondEu
    ro")==0) || (strcmp(((Option*)Opt)->Name,"ZeroCouponCallBo
    ndAmer")==0) || (strcmp(((Option*)Opt)->Name,"ZeroCouponPut
    BondEuro")==0) || (strcmp(((Option*)Opt)->Name,"ZeroCouponP
    utBondAmer")==0) )
        return OK;
    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_ZBOHW2D)=
{
    "TR_ZBOHW2D",
    {   {"StepNumber",INT2,{100},ALLOW},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(TR_ZBOHW2D),
    {
        {"Price",DOUBLE,{100},FORBID},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(TR_ZBOHW2D),
    CHK_ok,
    MET(Init)
} ;
```

References