

[Help](#)

```
#include "
href../../mod/hullwhite2d/hullwhite2d_std/hullwhite2d_std_h_src.pdfhullwhit
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "
href../../common/math/InterestRateModelTree/TreeHW2D/TreeHW2D_h_src.pdfmath/I
#include "
href../../mod/hullwhite2d/hullwhite2d_std/hullwhite2d_includes_h_src.pdfhull

//The "#else" part of the code will be freely available after the (year of creat
#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 the tree (see TreeHW2D.
/// ModelHW2D     : structure that contains the parameters of the Hull&White one
/// ZCMarketData : structure that contains the Zero Coupon Bond prices of the ma

/// Computation of the payoff at the final time of the tree (ie the option matur
static void ZCOption_InitialPayoff(TreeHW2D *Meth, ModelHW2D *ModelParam, ZCMark
{
    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 represents the nodes index

    double delta_y2; // delta_y1 = space step of the process y at time i ; delta_y
    double delta_u2; // delta_u1 = space step of the process u at time i ; delta_u
    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 * rh

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

pnl_mat_resize(OptionPriceMat2, jmaxprev - jminprev + 1, kmaxprev - kminprev +

delta_t1 = GET(Meth->t, Meth->Ngrid) - GET(Meth->t, Meth->Ngrid - 1); // Pas d
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, rho, T, S, &A_tT, &B_tT,

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, M

        ZCPrice = ZCPrice_Using_Coefficient(current_rate, current_u, A_tT, B_

```

```

        MLET(OptionPriceMat2, j - jminprev, k - kminprev) = (p->Compute)(p->Pa
    }
}

}

/// Price of an option on a ZC using a trinomial tree.
static double tr_hw2d_zcoption(TreeHW2D *Meth, ModelHW2D *ModelParam, ZCMarketDa
{
    double a , sigma1, b, sigma2, rho, sigma3;
    double delta_t1, current_rate, current_u, OptionPrice, delta_y1, delta_u1, A_t
    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 * rh

    ///***** Computation of the vector of payoff at the maturity of t
    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, OptionP

        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
    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, sigma2, rho, GET(Meth->t,

    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

            ZCPrice = ZCPrice_Using_Coefficient(current_rate, current_u, A

            // Decide whether to exercise the option or not
            if (MGET(OptionPriceMat2, h - jmin, l - kmin) < (p->Compute)(p
                {
                    MLET(OptionPriceMat2, h - jmin, l - kmin) = (p->Compute)(p
                }
            }
        }
    }

    }

    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, char *curve, double u0, double a,
                    double S, double T, NumFunc_1 *p, int am, int N_steps, double)
{
    TreeHW2D Tr;
    ModelHW2D ModelParams;
    ZCMarketData ZCMarket;

    /* Flag to decide to read or not ZC bond datas in "initialyields.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;
        ZCMarket.filename = curve;
        ReadMarketData(&ZCMarket);

        if (S > GET(ZCMarket.tm, ZCMarket.Nvalue - 1))
        {
            printf("\ nError : time bigger than the last time value entered in ini
            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 \ "Spee
        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,

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),
                    MOD(GetCurve)(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, "ZeroCouponCallBondEuro") == 0) || (strcmp(
        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)
} ;

```