

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/hullwhite1d/hullwhite1d_stdi/hullwhite1d_stdi_h_src.pdfhullwhit
#include "
href../../common/math/InterestRateModelTree/TreeShortRate/TreeShortRate_h_src
#include "pnl/pnl_vector.h"
#include "
href../../mod/hullwhite1d/hullwhite1d_stdi/hullwhite1d_includes_h_src.pdfhull

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2)
static int CHK_OPT(TR_ZBOHW1D)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_ZBOHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/// TreeShortRate      : structure that contains components of the tree (see Tree
/// ModelParameters    : structure that contains the parameters of the Hull&Whi
/// 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
void ZCOption_InitialPayoffHW1D(TreeShortRate *Meth, ModelParameters *ModelParam
{
    double a , sigma;

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

    double delta_x1; // delta_x1 = space step of the process u at time i
    double delta_t1; // time step

    double current_rate;
    double A_tT, B_tT, ZCPrice; // A_tT, B_tT scalars used in the ZC price : P(t,T
```

```

A_tT = 0;
B_tT = 0;
///<*****Parameters of the process r *****/
a = ModelParam->MeanReversion;
sigma = ModelParam->RateVolatility;

///<** Calcul du vecteur des payoffs a l'instant de maturite 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); // Time
delta_x1 = SpaceStep(delta_t1, a, sigma); // delta_x1 = space step of the pr

ZCPrice_CoefficientHW1D(ZCMarket, a, sigma, T, S, &A_tT, &B_tT); // Computati

for (j = jminprev ; j <= jmaxprev ; j++)
{
    current_rate = func_model_hw1d(j * delta_x1 + GET(Meth->alpha, Meth->Ngrid)

    ZCPrice = ZCPrice_Using_CoefficientHW1D(current_rate, A_tT, B_tT); // Comp

    LET(OptionPriceVect2, j - jminprev) = (p->Compute)(p->Par, ZCPrice); // Pa
}

}

///< Prix at time s of an option, maturing at T, on a ZC, with maturity S, using
double tr_hw1d_zcoption(TreeShortRate *Meth, ModelParameters *ModelParam, ZCMark
{
    int i, j, jmin, jmax;
    double a, sigma, delta_t1, delta_x1, A_tT, B_tT, current_rate, ZCPrice, Option

    PnlVect *OptionPriceVect1; // Vector of prices of the option at time i
    PnlVect *OptionPriceVect2; // Vector of prices of the option at time 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 payoff at the maturity of t
ZCOption_InitialPayoffHW1D(Meth, ModelParam, ZCMarket, OptionPriceVect2, p, T,

//***** Backward computation of the option price until time 0 **
for (i = Meth->Ngrid - 1; i >= 0; i--)
{
    BackwardIteration(Meth, ModelParam, OptionPriceVect1, OptionPriceVect2, i

    if (Eur_Or_Am != 0)
    {
        jmin = pnl_vect_int_get(Meth->Jminimum, i); // jminprev := jmin(i)
        jmax = pnl_vect_int_get(Meth->Jmaximum, i); // jmaxprev := jmax(i)

        delta_t1 = GET(Meth->t, i) - GET(Meth->t, MAX(i - 1, 0)); // TimeStep
        delta_x1 = SpaceStep(delta_t1, a, sigma); // SpaceStep (i)

        ZCPrice_CoefficientHW1D(ZCMarket, a, sigma, GET(Meth->t, i), S, &A_tT

        for (j = jmin ; j <= jmax ; j++)
        {
            current_rate = func_model_hw1d(j * delta_x1 + GET(Meth->alpha, i))

            ZCPrice = ZCPrice_Using_CoefficientHW1D(current_rate, A_tT, B_tT);
            // In the case of american option, decide wether to exerice the op
            if (GET(OptionPriceVect2, j - jmin) < (p->Compute)(p->Par, ZCPrice
                {
                    LET(OptionPriceVect2, j - jmin) = (p->Compute)(p->Par, ZCPrice
                }
            }
        }
    }

OptionPrice = GET(OptionPriceVect2, 0);

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

return OptionPrice;

```

```

} // FIN de la fonction ZCOption

static int tr_zbold(int flat_flag, double r0, char *curve, double a, double sigma)
{
    TreeShortRate Tr;
    ModelParameters 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 (T > GET(ZCMarket.tm, ZCMarket.Nvalue - 1))
        {
            printf("\nError : time bigger than the last time value entered in ini
            exit(EXIT_FAILURE);
        }
    }

    ModelParams.MeanReversion = a;
    ModelParams.RateVolatility = sigma;

    SetTimeGrid(&Tr, N_steps, T);

    SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_model_hw1d, &func_model_d

    //Price of an option on a ZC
    *price = tr_hw1d_zcoption(&Tr, &ModelParams, &ZCMarket, T, S, p, am);

```

```

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

return OK;
}

///  

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

int CALC(TR_ZBOHW1D)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return tr_zbo1d(ptMod->flat_flag.Val.V_INT,
                    MOD(GetYield)(ptMod),
                    MOD(GetCurve)(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->PayOff.Val.V_NUMFUNC_1,
                    ptOpt->EuOrAm.Val.V_BOOL,
                    Met->Par[0].Val.V_LONG,
                    &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(TR_ZBOHW1D)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "ZeroCouponCallBondEuro") == 0) || (strcmp(
        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_ZBOHW1D) =
{
    "TR_HullWhite1d_ZB0",
    { {"TimeStepNumber", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_ZBOHW1D),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(TR_ZBOHW1D),
    CHK_ok,
    MET(Init)
} ;

```