

[Help](#)

```
#include "
href../../../../mod/hullwhite1dgeneralized/hullwhite1dgeneralized_std/hullwhite1dg

#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_vector.h"

#include "
href../../../../common/math/InterestRateModelTree/TreeHW1dGeneralized/TreeHW1dGener
#include "
href../../../../common/math/read_market_zc/InitialYieldCurve_h_src.pdfmath/read_mar

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2)
static int CHK_OPT(TR_ZBOHW1DG)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_ZBOHW1DG)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/// Computation of the payoff at the final time of the tree (ie the option matur
static void ZCOption_InitialPayoffHW1DG(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Param
{
    int j, jminprev, jmaxprev; // jmin[i], jmax [i]
    double delta_x1, delta_t1, sigma, current_rate, ZCPrice;

    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
    sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(Meth->t, Meth->Ngrid));
    delta_x1 = SpaceStepHW1dG(delta_t1, sigma); //SpaceStepHW1dG(delta_t1, a, sigma
```

```

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

    ZCPrice = DiscountFactor(ZCMarket, HW1dG_Parameters, T, S, current_rate);

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

}

/// Backward computation of the price of an option on a Zero Coupon Bond
static void ZCOption_BackwardIteration(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Parameters)
{
    int i, j, k, jminprev, jmaxprev, jmin;
    double mean_reversion, sigma, delta_t1, delta_t2, delta_x1, delta_x2, beta_x,

    ///*****Parameters of the process r *****///
    mean_reversion = (HW1dG_Parameters->MeanReversion);

    jminprev = pnl_vect_int_get(Meth->Jminimum, index_last); // jmin(index_last)
    jmaxprev = pnl_vect_int_get(Meth->Jmaximum, index_last); // jmax(index_last)

    for (i = index_last - 1; i >= index_first; i--)
    {
        jmin = jminprev; // jmin := jmin(i+1)
        //jmax = jmaxprev; // jmax := jmax(i+1)

        jminprev = pnl_vect_int_get(Meth->Jminimum, i); // jmin(i)
        jmaxprev = pnl_vect_int_get(Meth->Jmaximum, i); // jmax(i)

        pnl_vect_resize(OptionPriceVect1, jmaxprev - jminprev + 1); // OptionPrice

        delta_t1 = GET(Meth->t, i) - GET(Meth->t, i - 1); // Time step between t[i]
        delta_t2 = GET(Meth->t, i + 1) - GET(Meth->t, i); // Time step between t[i]

        sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(Meth->t, i)); // sig
        delta_x1 = SpaceStepHW1dG(delta_t1, sigma); //SpaceStepHW1dG(delta_t1, a, s

        sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(Meth->t, i + 1)); //

```

```

delta_x2 = SpaceStepHW1dG(delta_t2, sigma); //SpaceStepHW1dG(delta_t2, a, s

beta_x = delta_x1 / delta_x2;

// Loop over the node at the time i
for (j = jminprev ; j <= jmaxprev ; j++)
{
    k = pnl_iround(j * beta_x * exp(-delta_t2 * mean_reversion)); //h inde

    // Probability to go from (i,j) to (i+1,k+1) with an UP movement
    Pup = ProbaUpHW1dG(j, k, delta_t2, beta_x, mean_reversion);
    // Probability to go from (i,j) to (i+1,k) with a Middle movement
    Pmiddle = ProbaMiddleHW1dG(j, k, delta_t2, beta_x, mean_reversion);
    // Probability to go from (i,j) to (i+1,k-1) with a Down movement
    Pdown = 1 - Pup - Pmiddle;

    current_rate = j * delta_x1 + GET(Meth->alpha, i); // r(i,j)

    LET(OptionPriceVect1, j - jminprev) = exp(-current_rate * delta_t2) *

    if (Eur_Or_Am != 0)
    {
        ZCPrice = DiscountFactor(ZCMarket, HW1dG_Parameters, GET(Meth->t,
        // In the case of american option, decide wether to exerice the op
        if (GET(OptionPriceVect1, j - jminprev) < (p->Compute)(p->Par, ZCP
        {
            LET(OptionPriceVect1, j - jminprev) = (p->Compute)(p->Par, ZCP
        }
    }
}

pnl_vect_clone(OptionPriceVect2, OptionPriceVect1); // Copy OptionPriceVec

} // END of the loop on i (time)
}

/// Price at time s of an option, maturing at T, on a ZC, with maturity S, using
static double tr_hw1dg_zcoption(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Parameters, Z
{

```

```

double delta_t1; // time step.
double Pup, Pmiddle, Pdown;

double current_rate;
double OptionPrice;

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 = HW1dG_Parameters->MeanReversion;

///<***** Computation of the vector of payoff at the maturity of t
ZCOption_InitialPayoffHW1DG(Meth, HW1dG_Parameters, ZCMarket, OptionPriceVect2);

///<***** Backward computation of the option price until initial t
ZCOption_BackwardIteration(Meth, HW1dG_Parameters, ZCMarket, S, OptionPriceVect2);

// First node of the tree
Pup = 1.0 / 6.0;
Pmiddle = 2.0 / 3.0 ;
Pdown = 1.0 / 6.0;

delta_t1 = GET(Meth->t, 1) - GET(Meth->t, 0); // Pas de temps entre t[1] et t[0]
current_rate = GET(Meth->alpha, 0); // r(i,j)
OptionPrice = exp(-current_rate * delta_t1) * (Pup * GET(OptionPriceVect1, 2) +
Pmiddle * GET(OptionPriceVect1, 1) + Pdown * GET(OptionPriceVect1, 0));

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

return OptionPrice;

} // FIN de la fonction ZCOption

static int tr_zbo_hw1dg(int flat_flag, double r0, char *curve, int CapletCurve,

```

```

{
    TreeHW1dG Tr;
    ModelHW1dG ModelParams;
    ZCMarketData ZCMarket;
    MktATMCapletVolData MktATMCapletVol;

    // Read the interest rate term structure from file, or set it flat
    if (flat_flag == 0)
    {
        ZCMarket.FlatOrMarket = 0;
        ZCMarket.Rate = r0;
    }

    else
    {
        ZCMarket.FlatOrMarket = 1;
        ZCMarket.filename = curve;
        ReadMarketData(&ZCMarket);
    }

    // Read the caplet volatilities from file "impliedcapletvol.dat".
    ReadCapletMarketData(&MktATMCapletVol, CapletCurve);

    hw1dg_calibrate_volatility(&ModelParams, &ZCMarket, &MktATMCapletVol, a);

    // Construction of the Time Grid
    SetTimeGridHW1dG(&Tr, N_steps, 0, T);

    // Construction of the tree, calibrated to the initial yield curve
    SetTreeHW1dG(&Tr, &ModelParams, &ZCMarket);

    //Price of an option on a ZC
    *price = tr_hw1dg_zcoption(&Tr, &ModelParams, &ZCMarket, T, S, p, r0, am);

    DeleteTreeHW1dG(&Tr);
    DeleteZCMarketData(&ZCMarket);
    DeleteMktATMCapletVolData(&MktATMCapletVol);
    DeletModelHW1dG(&ModelParams);

    return OK;
}

```

```
}
```

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

```
int CALC(TR_ZBOHW1DG)(void *Opt, void *Mod, PricingMethod *Met)
```

```
{
```

```
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
```

```
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
```

```
    return tr_zbo_hw1dg(ptMod->flat_flag.Val.V_INT,  
                        MOD(GetYield)(ptMod),  
                        MOD(GetCurve)(ptMod),  
                        ptMod->CapletCurve.Val.V_ENUM.value,  
                        ptMod->a.Val.V_DOUBLE,  
                        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_ZBOHW1DG)(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->HelpFilenameHint = "tr_hullwhite1dgeneralized_zbo";
```

```

        Met->Par[0].Val.V_LONG = 200;
    }
    return OK;
}

PricingMethod MET(TR_ZBOHW1DG) =
{
    "TR_HullWhite1dG_ZBO",
    { {"TimeStepNumber", LONG, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_ZBOHW1DG),
    {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(TR_ZBOHW1DG),
    CHK_ok,
    MET(Init)
} ;

```