```
   Help
#include <stdlib.h>
#include "hullwhite1d_stdi.h"
#include "math/InterestRateModelTree/TreeShortRate/TreeSho
    rtRate.h"
#include "pnl/pnl_vector.h"
#include "hullwhite1d_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 <
     (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 TreeShortRate.h)
/// ModelParameters    : structure that contains  the para
    meters of the Hull&White one factor model (see TreeShortRate.
    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)
void ZCOption_InitialPayoffHW1D(TreeShortRate* Meth, ModelP
    arameters* ModelParam, ZCMarketData* ZCMarket, PnlVect*
    OptionPriceVect2, NumFunc_1 *p, double T, double S)
{
    double a ,sigma;
```

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

double delta_x1; // delta_x1 = space step of the proces
s 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 * exp( -B_tT * current_rate)

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 matu
rite 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 step between t[Ngrid-1] et t[Ngrid]
delta_x1 = SpaceStep(delta_t1, a, sigma);   // delta_x1
 = space step of the process x at time t[Ngrid]

ZCPrice_CoefficientHW1D(ZCMarket, a, sigma, T, S, &A_tT
 , &B_tT); // Computation of the two scalars A_tT and B_tT

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

    ZCPrice = ZCPrice_Using_CoefficientHW1D(current_ra
```

```
    te, A_tT, B_tT); // Computation of the ZC price : P(T,S)

        LET(OptionPriceVect2, j-jminprev) = (p->Compute)(p-
    >Par, ZCPrice); // Payoff of the option
    }

}

/// Prix at time s of an option, maturing at T, on a ZC,
    with maturity S, using a trinomial tree.
double tr_hw1d_zcoption(TreeShortRate* Meth, ModelParamet
    ers* ModelParam, ZCMarketData* ZCMarket, double T, double S,
     NumFunc_1 *p, int Eur_Or_Am)
{
    int i, j, jmin, jmax;
    double a, sigma, delta_t1, delta_x1, A_tT, B_tT,
    current_rate, ZCPrice, 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 = ModelParam->MeanReversion;
    sigma = ModelParam->RateVolatility;

    ///****************** Computation of the vector of payo
    ff at the maturity of the option *******************///
    ZCOption_InitialPayoffHW1D(Meth, ModelParam, ZCMarket,
    OptionPriceVect2, p, T, S);

    ///****************** Backward computation of the
    option price until time 0 *******************///
    for (i = Meth->Ngrid-1; i>=0; i--)
    {
        BackwardIteration(Meth, ModelParam, OptionPriceVec
    t1, OptionPriceVect2, i+1, i, &func_model_hw1d);
```

```
    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 (i)
        delta_x1 = SpaceStep(delta_t1, a, sigma);  //
SpaceStep (i)

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

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

            ZCPrice = ZCPrice_Using_CoefficientHW1D(
current_rate, A_tT, B_tT); // ZC price P(ti, S, r_ti=current ra
te)
            // In the case of american option, decide
wether to exerice the option or not
            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_zbo1d(int flat_flag,double r0,double a,
    double sigma, double S,double T, NumFunc_1 *p,int am,int N_step
    s,double *price)
{
    TreeShortRate Tr;
    ModelParameters 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 (T > GET(ZCMarket.tm,ZCMarket.Nvalue-1))
        {
            printf("{nError : time bigger than the last
    time value entered in initialyield.dat{n");
            exit(EXIT_FAILURE);
        }
    }

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

    SetTimeGrid(&Tr, N_steps, T);

    SetTreeShortRate(&Tr, &ModelParams, &ZCMarket, &func_
```

```
    model_hw1d, &func_model_der_hw1d, &func_model_inv_hw1d);

    //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),
                    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,"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;
    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_ZBO",
    {   {"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)
} ;
```

References