

## Help

```
#include "hullwhite1dgeneralized_std.h"

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

#include "math/InterestRateModelTree/TreeHW1dGeneralized/TreeHW1dGeneralized.h"
#include "math/read_market_zc/InitialYieldCurve.h"

//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_BermudianSwaptionHW1DG)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(TR_BermudianSwaptionHW1DG)(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 BermudianSwaption_InitialPayoffHW1D(int swaption_start, TreeHW1dG *M
{
    double sigma;

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

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

    double ZCPrice, SumZC;
    double current_rate;

    int NumberOfPayments;
    double Ti;
```

```

ZCPrice = 0.0;

/** Calcul du vecteur des payoffs a l'instant de maturite de l'option
jminprev = pnl_vect_int_get(Meth->Jminimum, swaption_start); // jmin(swaption_start)
jmaxprev = pnl_vect_int_get(Meth->Jmaximum, swaption_start); // jmax(swaption_start)

pnl_vect_resize(OptionPriceVect2, jmaxprev - jminprev + 1);

delta_t1 = GET(Meth->t, swaption_start) - GET(Meth->t, swaption_start - 1);
sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(Meth->t, swaption_start));
delta_x1 = SpaceStepHW1dG(delta_t1, sigma); // SpaceStepHW1dG(delta_t1, a, sigma)

NumberOfPayments = (int) floor((contract_maturity - GET(Meth->t, swaption_start)) / periodicity);

p->Par[0].Val.V_DOUBLE = 1.0;

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

    SumZC = 0;
    for (i = 1; i <= NumberOfPayments; i++)
    {
        Ti = GET(Meth->t, swaption_start) + i * periodicity;
        ZCPrice = DiscountFactor(ZCMarket, HW1dG_Parameters, GET(Meth->t, swaption_start), Ti);

        SumZC += ZCPrice;
    }

    LET(OptionPriceVect2, j - jminprev) = ((p->Compute)(p->Par, periodicity * (j - jminprev)))
}

}

/// Price of a bermudianswaption using a trinomial tree
static double tr_hw1dg_bermudianswaption(TreeHW1dG *Meth, ModelHW1dG *HW1dG_Parameters)
{
    double delta_t1; // time step
    double Pup, Pmiddle, Pdown;

```

```

int i, j;
double Ti2, Ti1;
int i_Ti2, i_Ti1;
double current_rate, NumberOfPayments;
double OptionPrice;

PnlVect *PayoffVect;
PnlVect *OptionPriceVect1; // Vector of prices of the option at i
PnlVect *OptionPriceVect2; // Vector of prices of the option at i+1
OptionPriceVect1 = pnl_vect_create(1);
OptionPriceVect2 = pnl_vect_create(1);
PayoffVect = pnl_vect_create(1);

//mean_reversion = (HW1dG_Parameters->MeanReversion);

//***** Computation of the vector of payoff at the maturity of t
Ti1 = contract_maturity - periodicity;
i_Ti1 = IndexTimeHW1dG(Meth, Ti1);
BermudianSwaption_InitialPayoffHW1D(i_Ti1, Meth, HW1dG_Parameters, ZCMarket, 0

//***** Backward computation of the option price until initial t

NumberOfPayments = (int) floor((contract_maturity - option_maturity) / periodi

for (i = NumberOfPayments - 2 ; i >= 0 ; i--)
{
    Ti1 = option_maturity + i * periodicity;
    Ti2 = Ti1 + periodicity;
    i_Ti2 = IndexTimeHW1dG(Meth, Ti2);
    i_Ti1 = IndexTimeHW1dG(Meth, Ti1);

    BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionPriceVect1, OptionPri

    BermudianSwaption_InitialPayoffHW1D(i_Ti1, Meth, HW1dG_Parameters, ZCMarke

    for (j = 0; j < PayoffVect->size; j++)
    {
        if (GET(PayoffVect, j) > GET(OptionPriceVect2, j))
        {
            LET(OptionPriceVect2, j) = GET(PayoffVect, j);
        }
    }
}

```

```

    }

}

BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionPriceVect1, OptionPriceVect2)

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);
current_rate = GET(Meth->alpha, 0); // r(0,j)
OptionPrice = exp(-current_rate * delta_t1) * (Pup * GET(OptionPriceVect2, 2)

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

return OptionPrice;

}

static int tr_bermudianswaption1d(int flat_flag, double r0, char *curve, int Cap
{
    TreeHW1dG Tr;
    ModelHW1dG HW1dG_Parameters;
    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;
    }
}

```

[illegible]

```

        ptOpt->FixedRate.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_1,
        Met->Par[0].Val.V_LONG,
        &(Met->Res[0].Val.V_DOUBLE));
    }

static int CHK_OPT(TR_BermudianSwaptionHW1DG)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PayerBermudanSwaption") == 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_bermudianswaption";
        Met->Par[0].Val.V_INT = 50;
    }
    return OK;
}

PricingMethod MET(TR_BermudianSwaptionHW1DG) =
{
    "TR_HullWhite1dG_BermudianSwaption",
    { {"TimeStepNumber per Period", INT, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(TR_BermudianSwaptionHW1DG),
    {{"Price", DOUBLE, {100}, FORBID}/*,{"Delta",DOUBLE,{100},FORBID}*/ , {" ", PR
    CHK_OPT(TR_BermudianSwaptionHW1DG),
    CHK_ok,
    MET(Init)
} ;

```