```
   Help
#include "hullwhite1dgeneralized_stdi.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 aft
    er the (year of creation of this file + 2)
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2010+2)
static int CHK_OPT(TR_SwaptionHW1dG)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_SwaptionHW1dG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else


/// Computation of the payoff at the final time of the tre
    e (ie the option maturity)
void Swaption_InitialPayoffHW1dG(TreeHW1dG* Meth, ModelHW1
    dG* HW1dG_Parameters, ZCMarketData* ZCMarket, PnlVect*
    OptionPriceVect2, NumFunc_1 *p, double periodicity,double
    option_maturity,double contract_maturity, double SwaptionFixedRa
    te)
{
    double sigma;

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

    double delta_x1; // delta_x1 = space step of the proces
    s x at time i
```

```
double delta_t1; // time step

double ZCPrice, SumZC;
double current_rate;

int NumberOfPayments;
double Ti;

ZCPrice = .0;
///******************Parameters of the process r *****
****************////
//mean_reversion = (HW1dG_Parameters->MeanReversion);

///** 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[i] et t[i-1]
sigma = Current_VolatilityHW1dG(HW1dG_Parameters, GET(
Meth->t, Meth->Ngrid)); // sigma(ti)
delta_x1 = SpaceStepHW1dG(delta_t1, sigma);//SpaceStep    HW1dG(delta_t1, a,

NumberOfPayments = (int) floor((contract_maturity-
option_maturity )/periodicity + 0.2);

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

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

    SumZC = 0;
    for(i=1; i<=NumberOfPayments; i++)
    {
```

```
            Ti = option_maturity + i*periodicity;
            ZCPrice = DiscountFactor(ZCMarket, HW1dG_Para
    meters, option_maturity, Ti, current_rate);

            SumZC += ZCPrice;
        }

        LET(OptionPriceVect2, j-jminprev) = ((p->Compute)(
    p->Par, periodicity * SwaptionFixedRate * SumZC + ZCPrice))
    ;
    }

}

/// Price of a swaption using a trinomial tree
double tr_hw1dg_swaption(TreeHW1dG* Meth, ModelHW1dG* HW1dG
    _Parameters, ZCMarketData* ZCMarket,int NumberOfTimeStep,
    NumFunc_1 *p, double periodicity,double option_maturity,double
    contract_maturity, double SwaptionFixedRate)
{
    double delta_t1; // time step
    double Pup, Pmiddle, Pdown;

    double current_rate;
    double OptionPrice;

    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);

    //mean_reversion = (HW1dG_Parameters->MeanReversion);

    ///****************** Computation of the vector of payo
    ff at the maturity of the option ********************///
    Swaption_InitialPayoffHW1dG(Meth, HW1dG_Parameters, ZCM
    arket, OptionPriceVect2, p, periodicity, option_maturity,
    contract_maturity, SwaptionFixedRate);
```

```
    ///****************** Backward computation of the
    option price until initial time s ********************///

    BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionP
    riceVect1, OptionPriceVect2, Meth->Ngrid, 1);

    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);
    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;

}


static int tr_swaption1d(int flat_flag, double r0, int     CapletCurve, double a
    maturity, double periodicity,double Nominal, double SwaptionF
    ixedRate, NumFunc_1 *p, int NumberOfTimeStep, double *
    price)
{
    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;
        ReadMarketData(&ZCMarket);
    }

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

    hw1dg_calibrate_volatility(&HW1dG_Parameters, &ZCMarke
    t, &MktATMCapletVol, a);



    // Construction of the Time Grid
    SetTimeGridHW1dG(&Tr, NumberOfTimeStep, 0, option_matu
    rity);

    // Construction of the tree, calibrated to the initial
    yield curve
    SetTreeHW1dG(&Tr, &HW1dG_Parameters, &ZCMarket);


    *price =  Nominal * tr_hw1dg_swaption(&Tr, &HW1dG_Para
    meters, &ZCMarket, NumberOfTimeStep,  p, periodicity, option_
    maturity, contract_maturity, SwaptionFixedRate);

    DeleteTreeHW1dG(&Tr);
    DeleteZCMarketData(&ZCMarket);
    DeleteMktATMCapletVolData(&MktATMCapletVol);
    DeletModelHW1dG(&HW1dG_Parameters);

    return OK;
}



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

```
int CALC(TR_SwaptionHW1dG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return  tr_swaption1d(    ptMod->flat_flag.Val.V_INT,
                            MOD(GetYield)(ptMod),
                            ptMod->CapletCurve.Val.V_ENUM.
    value,
                            ptMod->a.Val.V_DOUBLE,
                            ptOpt->BMaturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                            ptOpt->OMaturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                            ptOpt->ResetPeriod.Val.V_DATE,
                            ptOpt->Nominal.Val.V_PDOUBLE,
                            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_SwaptionHW1dG)(void *Opt, void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"PayerSwaption")==0) ||
    (strcmp(((Option*)Opt)->Name,"ReceiverSwaption")==0))
    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_swaption";
```

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


PricingMethod MET(TR_SwaptionHW1dG)=
{
  "TR_HullWhite1dG_Swaption",
  {{"TimeStepNumber",INT,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_SwaptionHW1dG),
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(TR_SwaptionHW1dG),
  CHK_ok,
  MET(Init)
} ;
```

# References