```
   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_CapFloorHW1dG)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TR_CapFloorHW1dG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else


static void CapFloor_InitialPayoffHW1dG(TreeHW1dG* Meth,
    ModelHW1dG* HW1dG_Parameters, PnlVect* ZCbondPriceVect, PnlVec
    t* OptionPriceVect, int i_T,  NumFunc_1 *p, double periodic
    ity, double CapFloorFixedRate)
{
    int jminprev, jmaxprev;
    int j;

    double 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(ZCbondPriceVect, jmaxprev-jminprev+1);

    pnl_vect_set_double(ZCbondPriceVect, 1.0); // Payoff =
    1 for a ZC bond

    BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionP
    riceVect, ZCbondPriceVect, Meth->Ngrid, i_T);

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

    for( j = 0 ; j<ZCbondPriceVect->size ; j++)
    {
        ZCPrice = GET(ZCbondPriceVect, j);

        LET(OptionPriceVect, j) = (p->Compute)(p->Par, (1+
    periodicity*CapFloorFixedRate)*ZCPrice);
    }
}


static void CapFloor_BackwardIteration(TreeHW1dG* Meth,
    ModelHW1dG* HW1dG_Parameters, NumFunc_1 *p, PnlVect* ZCbondPric
    eVect1, PnlVect* ZCbondPriceVect2, PnlVect* OptionPriceVec
    t1, PnlVect* OptionPriceVect2, int index_last, int index_
    first, double periodicity, double CapFloorFixedRate)
{
    double mean_reversion, sigma, ZCPrice;

    int jmin; // jmin[i+1], jmax[i+1]
    int jminprev, jmaxprev; // jmin[i], jmax [i]
    int i, j, k;
    double Pup, Pdown, Pmiddle;
    double delta_x1, delta_x2, beta_x;
    double delta_t1, delta_t2;
    double current_rate;


    ///******************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)

pnl_vect_resize(ZCbondPriceVect2, OptionPriceVect2->si
ze);

pnl_vect_set_double(ZCbondPriceVect2, 1.0); // Payoff =
 1 for a ZC bond

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);  // OptionPrice1 := Prix a l'instant i,
    pnl_vect_resize(ZCbondPriceVect1, jmaxprev-jminprev
 +1);

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

    sigma = Current_VolatilityHW1dG(HW1dG_Parameters,
GET(Meth->t, i)); // sigma(ti)
    delta_x1 = SpaceStepHW1dG(delta_t1, sigma);//SpaceS
tepHW1dG(delta_t1, a, sigma);

    sigma = Current_VolatilityHW1dG(HW1dG_Parameters,
GET(Meth->t, i+1)); // sigma(ti+1)
    delta_x2 = SpaceStepHW1dG(delta_t2, sigma);//SpaceS
tepHW1dG(delta_t2, a, sigma);
```

```
    beta_x = delta_x1 / delta_x2;

    // Loop over the node at the time i
    for(j = jminprev ; j<= jmaxprev ; j++)
    {
        k = intapprox(j*beta_x*exp(-delta_t2 * mean_rev
ersion)); //h index of the middle node emanating from (i,j)

        // Probability to go from (i,j) to (i+1,k+1)
with an UP movement
        Pup = ProbaUpHW1dG(j, k, delta_t2, beta_x, mea
n_reversion);
        // Probability to go from (i,j) to (i+1,k) wit
h a Middle movement
        Pmiddle = ProbaMiddleHW1dG(j, k, delta_t2, bet
a_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) * ( Pup * GET(OptionPriceVect2, k+1-jmin)
+ Pmiddle * GET(OptionPriceVect2, k-jmin) + Pdown * GET(
OptionPriceVect2, k-1-jmin)); // Backward computation of the bo
nd price

        LET(ZCbondPriceVect1,j-jminprev) = exp(-
current_rate*delta_t2) * ( Pup * GET(ZCbondPriceVect2, k+1-jmin)
+ Pmiddle * GET(ZCbondPriceVect2, k-jmin) + Pdown * GET(    ZCbondPriceVect2

    }
    // Copy OptionPrice1 in OptionPrice2
    pnl_vect_clone(OptionPriceVect2, OptionPriceVect1);
    pnl_vect_clone(ZCbondPriceVect2, ZCbondPriceVect1);

} // END of the loop on i

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

```
    for( j = 0 ; j<ZCbondPriceVect2->size ; j++)
    {
        ZCPrice = GET(ZCbondPriceVect2, j);

        LET(OptionPriceVect2, j) += (p->Compute)(p->Par, (1
    +periodicity*CapFloorFixedRate)*ZCPrice);
    }

}

/// Price of a Cap/Floor using a trinomial tree

static double tr_hw1dg_capfloor(TreeHW1dG* Meth, ModelHW1dG
    * HW1dG_Parameters, ZCMarketData* ZCMarket, int NumberOfTi
    meStep, NumFunc_1 *p, double periodicity,double first_reset_
    date,double contract_maturity, double CapFloorFixedRate)
{

    int i, i_Ti2, i_Ti1, n;
    double Ti2, Ti1, delta_t1, current_rate, OptionPrice,
    Pup, Pdown, Pmiddle;

    PnlVect* OptionPriceVect1; // Vector of prices of the
    option at i
    PnlVect* OptionPriceVect2; // Vector of prices of the
    option at i+1
    PnlVect* ZCbondPriceVect1; // Vector of prices of the
    option at i+1
    PnlVect* ZCbondPriceVect2; // Vector of prices of the
    option at i+1
    OptionPriceVect1 = pnl_vect_create(1);
    OptionPriceVect2 = pnl_vect_create(1);
    ZCbondPriceVect1 = pnl_vect_create(1);
    ZCbondPriceVect2 = pnl_vect_create(1);

    // Mmean reversion parameter
    //a = HW1dG_Parameters->MeanReversion;

    ///*************** PAYOFF at the MATURITY of the
    OPTION : T(n-1)***************///
```

```
Ti2 = contract_maturity;
Ti1 = Ti2 - periodicity;
i_Ti1 = IndexTimeHW1dG(Meth, Ti1);

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

CapFloor_InitialPayoffHW1dG(Meth, HW1dG_Parameters,     ZCbondPriceVect2, Op
loorFixedRate);

///**************** Backward computation of the option
price ****************///

n = (int) ((contract_maturity-first_reset_date)/perio
dicity + 0.1);

for(i = n-2; i>=0; i--)
{
    Ti1 = first_reset_date + i * periodicity; // Ti1 =
T(i)
    Ti2 = Ti1 + periodicity;                  // Ti2 =
T(i+1)

    i_Ti2 = IndexTimeHW1dG(Meth, Ti2);
    i_Ti1 = IndexTimeHW1dG(Meth, Ti1);

    CapFloor_BackwardIteration(Meth, HW1dG_Parameters,
p, ZCbondPriceVect1, ZCbondPriceVect2, OptionPriceVect1,
OptionPriceVect2,  i_Ti2, i_Ti1, periodicity, CapFloorFixedRate)
;
}

///****************** Price of the option at initial
time s ******************///

BackwardIterationHW1dG(Meth, HW1dG_Parameters, OptionP
riceVect1, OptionPriceVect2, i_Ti1, 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); // 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);
    pnl_vect_free(& ZCbondPriceVect1);
    pnl_vect_free(& ZCbondPriceVect2);

    return OptionPrice;

}

static int tr_capfloor1d(int flat_flag, double r0, int     CapletCurve, double a
    et_date, double periodicity,double Nominal, double CapFloor
    FixedRate, NumFunc_1 *p, long N_steps, 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
    SetTimeGrid_TenorHW1dG(&Tr, N_steps, first_reset_date,
    contract_maturity, periodicity);

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

    *price =  Nominal * tr_hw1dg_capfloor(&Tr, &HW1dG_Para
    meters, &ZCMarket, N_steps, p, periodicity, first_reset_date,
     contract_maturity, CapFloorFixedRate);

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

    return OK;
}



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

int CALC(TR_CapFloorHW1dG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return  tr_capfloor1d(    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->FirstResetDate.Val.V_DA
    TE-ptMod->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_CapFloorHW1dG)(void *Opt, void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"Cap")==0) || (strcmp(((
    Option*)Opt)->Name,"Floor")==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_capfloor";
      Met->Par[0].Val.V_INT=20;
    }
  return OK;
}


PricingMethod MET(TR_CapFloorHW1dG)=
{
  "TR_HullWhite1dG_CapFloor",
```

```
  {{"TimeStepNumber per Period",INT,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(TR_CapFloorHW1dG),
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(TR_CapFloorHW1dG),
  CHK_ok,
  MET(Init)
} ;
```

References