```
   Help
#include "hullwhite1dgeneralized_stdi.h"

#include "math/read_market_zc/InitialYieldCurve.h"
#include "hullwhite1dgeneralized_volcalibration.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)
int CALC(CF_PayerSwaptionHW1dG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(CF_PayerSwaptionHW1dG)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
#else

///* Computation the function phi used to find the Criti
    cal Rate in the Jamishidian decomposition
static double phi(ZCMarketData* ZCMarket, ModelHW1dG* HW1dG
    _Parameters, double r, double periodicity, double option_
    maturity, double contract_maturity, double SwaptionFixedRate)
{
    int i, nb_payement;
    double a, ci, sum, sum_der, T, Ti, B_tT, ZCPrice;

    B_tT = 0.;
    ZCPrice = 0.;
    sum = 0.;
    sum_der = 0.;

    ci = periodicity * SwaptionFixedRate;
    T = option_maturity;
    Ti = option_maturity;

    a = HW1dG_Parameters->MeanReversion;
    nb_payement = (int)((contract_maturity-option_maturity)
    /periodicity);

    for(i=1; i<=nb_payement; i++)
    {
        Ti += periodicity;

        B_tT = (1-exp(-a*(Ti-T)))/a;
        ZCPrice = DiscountFactor(ZCMarket, HW1dG_Paramete
    rs, T, Ti, r);

        sum += ci * ZCPrice;
        sum_der += ci * ZCPrice * (-B_tT);
    }

    sum += ZCPrice;

    sum_der += ZCPrice * (-B_tT);

    return (sum-1.)/sum_der;
}


///* Computation of Critical Rate in the Jamishidian de
    composition, with the newton method to find zero of a function
static double Critical_Rate(ZCMarketData* ZCMarket, ModelH
    W1dG* HW1dG_Parameters, double r_initial, double periodic
    ity, double option_maturity, double contract_maturity,
    double SwaptionFixedRate)
{
    double previous, current_rate;
    int nbr_iterations;

    const double precision = 0.0001;

    current_rate = r_initial;
    nbr_iterations = 0;

    do
    {
        nbr_iterations++;
        previous = current_rate;
        current_rate = current_rate - phi(ZCMarket, HW1dG_
    Parameters, current_rate, periodicity, option_maturity, contr
    act_maturity, SwaptionFixedRate);
    } while((fabs(previous-current_rate) > precision) && (
    nbr_iterations <= 10));

    return current_rate;
}

///* Payer Swaption price as a combination of ZC Put
    option prices
static int cf_ps1d(int flat_flag, double r_t, int CapletCu
    rve, double Nominal, double periodicity, double option_matu
    rity, double contract_maturity, double SwaptionFixedRate,
    double a,double *price)
{
    int i, nb_payement;
    double ci, sum ,Ti;

    double critical_r, Strike_i, PutOptionPrice;

    ModelHW1dG HW1dG_Parameters;
    ZCMarketData ZCMarket;
    MktATMCapletVolData MktATMCapletVol;

    PutOptionPrice = 0.; /* to avoid warning */

    /* 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 = r_t;
    }

    else
    {
        ZCMarket.FlatOrMarket = 1;
        ReadMarketData(&ZCMarket);
        r_t = -log(BondPrice(INC, &ZCMarket))/INC;

        if(contract_maturity > GET(ZCMarket.tm,ZCMarket.Nv
    alue-1))
        {
            printf("{nError : time bigger than the last
    time value entered in initialyield.dat{n");
            exit(EXIT_FAILURE);
        }
    }

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

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

    Ti = option_maturity;
    ci = periodicity * SwaptionFixedRate;

    nb_payement = (int)((contract_maturity-option_maturity)
    /periodicity);

    critical_r = Critical_Rate(&ZCMarket, &HW1dG_Paramete
    rs, r_t, periodicity, option_maturity, contract_maturity,
    SwaptionFixedRate);

    sum = 0.;

    for(i=1; i<=nb_payement; i++)
    {
        Ti += periodicity;

        Strike_i = DiscountFactor(&ZCMarket, &HW1dG_Para
    meters, option_maturity, Ti, critical_r);

        PutOptionPrice = hw1dg_zc_put_price(&ZCMarket, &    HW1dG_Parameters, St

        sum += ci * PutOptionPrice;
    }

    sum += PutOptionPrice;

    *price = Nominal * sum;

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

    return OK;
}

int CALC(CF_PayerSwaptionHW1dG)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;


  return cf_ps1d( ptMod->flat_flag.Val.V_INT,
                  MOD(GetYield)(ptMod),
                  ptMod->CapletCurve.Val.V_ENUM.value,
                  ptOpt->Nominal.Val.V_PDOUBLE,
                  ptOpt->ResetPeriod.Val.V_DATE,
                  ptOpt->OMaturity.Val.V_DATE-ptMod->T.Val.
    V_DATE,
                  ptOpt->BMaturity.Val.V_DATE-ptMod->T.Val.
    V_DATE,
                  ptOpt->FixedRate.Val.V_PDOUBLE,
                  ptMod->a.Val.V_DOUBLE,
                  &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(CF_PayerSwaptionHW1dG)(void *Opt, void *
    Mod)
{
  return strcmp( ((Option*)Opt)->Name,"PayerSwaption");
}
#endif //PremiaCurrentVersion
```

```
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;
       Met->HelpFilenameHint = "    cf_hullwhite1dgeneralized_payerswaption";
    }

  return OK;
}

PricingMethod MET(CF_PayerSwaptionHW1dG)=
{
  "CF_HullWhite1dG_PayerSwaption",
  {{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(CF_PayerSwaptionHW1dG),
  {{"Price",DOUBLE,{100},FORBID},{" ",PREMIA_NULLTYPE,{0},
    FORBID}},
  CHK_OPT(CF_PayerSwaptionHW1dG),
  CHK_ok,
  MET(Init)
} ;
```

References