Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
#else

#include <stdio.h>
#include <stdlib.h>

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

#include "
href../../../common/math/read_market_zc/InitialYieldCurve_h_src.pdfmath/read_mar
#include "
href../../../mod/hullwhite1dgeneralized/hullwhite1dgeneralized_stdi/hullwhite1dg

// Caplet price in the Black model.
double black_caplet_price(ZCMarketData *ZCMarket, double vol_impli, double caple
{
  double d1, d2, LiborRate, DiscountFactor1, DiscountFactor2, caplet_price;

  DiscountFactor1 = BondPrice(caplet_reset_date, ZCMarket);
  DiscountFactor2 = BondPrice(caplet_reset_date + periodicity, ZCMarket);

  LiborRate = (DiscountFactor1 / DiscountFactor2 - 1) / periodicity;


  d1 = (log(LiborRate / caplet_strike) + 0.5 * SQR(vol_impli) * caplet_reset_dat
  d2 = d1 - vol_impli * sqrt(caplet_reset_date);

  caplet_price = DiscountFactor2 * periodicity * (LiborRate * cdf_nor(d1) - capl

  return caplet_price;
}

// Implied volatility of a caplet (Black model)
double bk_caplet_vol_implied_newton(ZCMarketData *ZCMarket, double caplet_price,
{
  int i, MAX_ITERATIONS;
```

```
  double ACCURACY;
  double T_sqrt, vol_avg, price, diff, d1, vega, DiscountFactor1, DiscountFactor

  MAX_ITERATIONS = 50;
  ACCURACY    = 1.0e-10;

  T_sqrt = sqrt(caplet_reset_date);
  vol_avg = 0.1;

  DiscountFactor1 = BondPrice(caplet_reset_date, ZCMarket);
  DiscountFactor2 = BondPrice(caplet_reset_date + periodicity, ZCMarket);

  for (i = 0; i < MAX_ITERATIONS; i++)
    {
      price = black_caplet_price(ZCMarket, vol_avg, caplet_strike, periodicity,
      diff = caplet_price - price;

      if (fabs(diff) < ACCURACY) return vol_avg;

      d1 = (log((DiscountFactor1 - DiscountFactor2) / (periodicity * DiscountFac

      vega = (DiscountFactor1 - DiscountFactor2) * cdf_nor(d1) * T_sqrt;
      vol_avg = vol_avg + diff / vega;
    }

  return -99e10;  // something screwy happened, should throw exception
}

// Caplet price in the HW1dGeneralized as a function of the average volatility o
double hw1dg_caplet_price(ZCMarketData *ZCMarket, double vol_avg, double caplet_
{
  double d1, d2, DiscountFactor1, DiscountFactor2, caplet_price;

  DiscountFactor1 = BondPrice(caplet_reset_date, ZCMarket);
  DiscountFactor2 = BondPrice(caplet_reset_date + periodicity, ZCMarket);


  d1 = (log((1 + caplet_strike * periodicity) * DiscountFactor2) - log(DiscountF
  d2 = d1 - vol_avg * sqrt(caplet_reset_date);

  caplet_price = DiscountFactor1 * cdf_nor(-d2) - (1 + caplet_strike * periodici
```

```
  return caplet_price;
}

double hw1dg_floorlet_price(ZCMarketData *ZCMarket, double vol_avg, double caple
{
  double d1, d2, DiscountFactor1, DiscountFactor2, caplet_price;

  DiscountFactor1 = BondPrice(caplet_reset_date, ZCMarket);
  DiscountFactor2 = BondPrice(caplet_reset_date + periodicity, ZCMarket);


  d1 = (log((1 + caplet_strike * periodicity) * DiscountFactor2) - log(DiscountF
  d2 = d1 - vol_avg * sqrt(caplet_reset_date);

  caplet_price = (1 + caplet_strike * periodicity) * DiscountFactor2 * cdf_nor(d

  return caplet_price;
}

// Compute the average volatility of the forward price of discount bond
// Forward price of discount bond at time t is : P(t, S)/P(t, T) with t<T<S
double hw1dg_fwd_zc_vol_implied_newton(ZCMarketData *ZCMarket, double caplet_pri
{
  int i, MAX_ITERATIONS;
  double ACCURACY;
  double T_sqrt, vol_avg, price, diff, d2, vega, DiscountFactor1, DiscountFactor

  MAX_ITERATIONS = 50;
  ACCURACY    = 1.0e-10;

  T_sqrt = sqrt(caplet_reset_date);
  vol_avg = 0.5;

  DiscountFactor1 = BondPrice(caplet_reset_date, ZCMarket);
  DiscountFactor2 = BondPrice(caplet_reset_date + periodicity, ZCMarket);

  for (i = 0; i < MAX_ITERATIONS; i++)
    {
      price = hw1dg_caplet_price(ZCMarket, vol_avg, caplet_strike, periodicity,
      diff = caplet_price - price;
```

```
      if (fabs(diff) < ACCURACY) return vol_avg;

      d2 = (log((1 + caplet_strike * periodicity) * DiscountFactor2) - log(Disco

      vega = DiscountFactor1 * cdf_nor(d2) * T_sqrt;
      vol_avg = vol_avg + diff / vega;
    }

  return -99e10;  // something screwy happened, should throw exception
}

// Compute the average volatility of the forward discount factor from the ATM ca
// To do so we just calculate the price of caplets using theirs implied volatili
void From_Black_To_HW1dG_volatility(ZCMarketData *ZCMarket, MktATMCapletVolData
{
  double caplet_price, caplet_reset_date, caplet_payment_date, black_caplet_vola
  int i, N;

  N = MktATMCapletVol->NbrData;
  periodicity = MktATMCapletVol->Periodicity;

  pnl_vect_clone(mkt_fwd_zc_mat, MktATMCapletVol->CapletMaturity);
  pnl_vect_resize(mkt_fwd_zc_vol, N);

  for (i = 0; i < N; i++)
    {
      caplet_reset_date = GET(mkt_fwd_zc_mat, i);
      caplet_payment_date = caplet_reset_date + periodicity;

      // Strike for a caplet At-The-Money = Libor Rate(0, T, S)
      atm_caplet_strike = (BondPrice(caplet_reset_date, ZCMarket) / BondPrice(ca
      black_caplet_volatiltiy = GET(MktATMCapletVol->CapletVolatility, i);

      // Compute the price of the i'th caplet using teh Black formula and its im
      caplet_price = black_caplet_price(ZCMarket, black_caplet_volatiltiy, atm_c

      LET(mkt_fwd_zc_vol, i) = hw1dg_fwd_zc_vol_implied_newton(ZCMarket, caplet_
    }

}
```

```
// Compute the parameters of the volatility function of HW1d model (supposed to
// The computation is done in a way to match the forward discount factor volatil
int hw1dg_calibrate_volatility(ModelHW1dG *HW1dG_Parameters, ZCMarketData *ZCMar
{
  int i, N;
  double kappa, T1, T2, alpha1, alpha2, sigma_avg1, sigma_avg2, periodicity;
  PnlVect *mkt_fwd_zc_mat, *mkt_fwd_zc_vol;


  HW1dG_Parameters->MeanReversion = hw1dg_mean_reversion;
  kappa = HW1dG_Parameters->MeanReversion;
  periodicity = MktATMCapletVol->Periodicity;

  mkt_fwd_zc_mat = pnl_vect_create(0);
  mkt_fwd_zc_vol = pnl_vect_create(0);

  // Compute the average volatility of the forward discount factor from the ATM
  From_Black_To_HW1dG_volatility(ZCMarket, MktATMCapletVol, mkt_fwd_zc_mat , mkt

  N = mkt_fwd_zc_vol->size;

  HW1dG_Parameters->ShortRateVolGrid = pnl_vect_create(N);
  HW1dG_Parameters->TimeGrid = pnl_vect_create(0);
  pnl_vect_clone(HW1dG_Parameters->TimeGrid, mkt_fwd_zc_mat);

  T1 = GET(mkt_fwd_zc_mat, 0);

  LET(HW1dG_Parameters->ShortRateVolGrid, 0) = sqrt(T1 * 2 * kappa / (exp(2 * ka
      (kappa * GET(mkt_fwd_zc_vol, 0) / (exp(-kappa * T1) * (1 - exp(-kappa * pe

  for (i = 1; i < N; i++)
    {
      T1 = GET(mkt_fwd_zc_mat, i - 1);
      T2 = GET(mkt_fwd_zc_mat, i);

      alpha1 = T1 * kappa * kappa / (exp(-2 * kappa * T1) * SQR(1 - exp(-kappa *
      alpha2 = T2 * kappa * kappa / (exp(-2 * kappa * T2) * SQR(1 - exp(-kappa *

      sigma_avg1 = GET(mkt_fwd_zc_vol, i - 1);
```

```
      sigma_avg2 = GET(mkt_fwd_zc_vol, i);

      LET(HW1dG_Parameters->ShortRateVolGrid, i) = sqrt((SQR(sigma_avg2) * alpha
          2 * kappa / (exp(2 * kappa * T2) - exp(2 * kappa * T1)));
    }


  pnl_vect_free(&mkt_fwd_zc_mat);
  pnl_vect_free(&mkt_fwd_zc_vol);
  return 1;
}


static double Integrale(ModelHW1dG *HW1dG_Parameters, double t)
{
  int i, j, N;
  double integral, a, sigma_j, T_j1, T_j2;

  i = 0;
  N = (HW1dG_Parameters->TimeGrid)->size;
  a = HW1dG_Parameters->MeanReversion;

  if (HW1dG_Parameters->TimeGrid == NULL)
    {
      printf("FATALE ERREUR, PAS DE GRILLE DE TEMPS !");
    }

  else
    {
      while (GET(HW1dG_Parameters->TimeGrid, i) < t && i < N - 1)
        {
          i++;
        }
    }

  integral = .0;

  // if t<=T[0]
  if (i == 0)
    {
      T_j2 = t;
```

```
      sigma_j = GET(HW1dG_Parameters->ShortRateVolGrid, i);
      integral = exp(-2 * a * t) * SQR(sigma_j) * (exp(2 * a * T_j2) - 1.0) / (4

      return integral;
    }

  // if t>T[0]
  T_j2 = GET(HW1dG_Parameters->TimeGrid, 0);
  sigma_j = GET(HW1dG_Parameters->ShortRateVolGrid, 0);
  integral += SQR(sigma_j) * (exp(2 * a * T_j2) - 1.) / (4.*a);

  for (j = 0; j < i - 1; j++)
    {
      T_j1 = GET(HW1dG_Parameters->TimeGrid, j);
      T_j2 = GET(HW1dG_Parameters->TimeGrid, j + 1);
      sigma_j = GET(HW1dG_Parameters->ShortRateVolGrid, j + 1);
      integral += SQR(sigma_j) * (exp(2 * a * T_j2) - exp(2 * a * T_j1)) / (4 *
    }

  T_j1 = GET(HW1dG_Parameters->TimeGrid, i - 1);
  T_j2 = t;
  sigma_j = GET(HW1dG_Parameters->ShortRateVolGrid, i);
  integral += SQR(sigma_j) * (exp(2 * a * T_j2) - exp(2 * a * T_j1)) / (4 * a);

  integral *= exp(-2 * a * t);

  return integral;
}

double DiscountFactor(ZCMarketData *ZCMarket, ModelHW1dG *HW1dG_Parameters, doub
{
  double a, P_0t, P_0T, integral, B_tT, f_0t;
  double P_tT;

  a = HW1dG_Parameters->MeanReversion;
  B_tT = (1 - exp(-a * (T - t))) / a;
  P_0t = BondPrice(t, ZCMarket);
  P_0T = BondPrice(T, ZCMarket);
  f_0t = ForwardRate(t, ZCMarket);

  integral = Integrale(HW1dG_Parameters, t);
```

```
  P_tT = (P_0T / P_0t) * exp(B_tT * f_0t - SQR(B_tT) * integral - B_tT * r_t);
  return P_tT;
}


double hw1dg_fwd_zc_average_vol(ModelHW1dG *HW1dG_Parameters, double T, double S
{
  double integral, a;

  a = HW1dG_Parameters->MeanReversion;
  integral = Integrale(HW1dG_Parameters, T);

  return (1 - exp(-a * (S - T))) * sqrt(2 * integral / T) / a;
}

double hw1dg_zc_put_price(ZCMarketData *ZCMarket, ModelHW1dG *HW1dG_Parameters,
{
  double sigma_avg, caplet_strike, periodicity, zc_put_price;

  periodicity = S - T;
  caplet_strike = (1 - strike) / (periodicity * strike);
  sigma_avg = hw1dg_fwd_zc_average_vol(HW1dG_Parameters, T, S);

  zc_put_price = strike * hw1dg_caplet_price(ZCMarket, sigma_avg, caplet_strike,

  return zc_put_price;

}

double hw1dg_zc_call_price(ZCMarketData *ZCMarket, ModelHW1dG *HW1dG_Parameters,
{
  double sigma_avg, caplet_strike, periodicity, zc_call_price;

  periodicity = S - T;
  caplet_strike = (1 - strike) / (periodicity * strike);
  sigma_avg = hw1dg_fwd_zc_average_vol(HW1dG_Parameters, T, S);

  zc_call_price = strike * hw1dg_floorlet_price(ZCMarket, sigma_avg, caplet_stri

  return zc_call_price;
```

```
}
///********************************** Read the caplet volatilities from file ***



// Read the caplet volatilities from the file "impliedcapletvol.dat" and put it
void ReadCapletMarketData(MktATMCapletVolData *MktATMCapletVol, int CapletCurve)
{
  FILE *Entrees;                    /*File variable of the code*/

  int i;
  char ligne[20];
  char *pligne;
  double p, tt;
  char data[MAX_PATH_LEN];
  char *init; // Name of the file where to read caplet volatilities of the marke

  if (CapletCurve == 1) init = "impliedcapletvol_1.dat";
  if (CapletCurve == 2) init = "impliedcapletvol_2.dat";

  sprintf(data, "%s%s%s", premia_data_dir, path_sep, init);
  Entrees = fopen(data, "r");

  if (Entrees == NULL)
    {
      printf("Le FICHIER N'A PU ETRE OUVERT. VERIFIER LE CHEMIN\ n");
      abort();
    }

  i = 0; // i represents the number of value read in the file
  pligne = ligne;

  MktATMCapletVol->CapletVolatility = pnl_vect_create(100);
  MktATMCapletVol->CapletMaturity = pnl_vect_create(100);

  pligne = fgets(ligne, sizeof(ligne), Entrees);
  sscanf(ligne, "periodicity=%lf", &tt);
  MktATMCapletVol->Periodicity = tt;

  while (1)
    {
```

```
      pligne = fgets(ligne, sizeof(ligne), Entrees);
      if (pligne == NULL)
        {
          break;
        }
      else
        {
          sscanf(ligne, "%lf t=%lf", &p, &tt);
          LET(MktATMCapletVol->CapletVolatility, i) = p; // Store the caplet vol
          LET(MktATMCapletVol->CapletMaturity, i) = tt; // Store the caplet matu
          i++;
        }
    }

  fclose(Entrees);

  MktATMCapletVol->NbrData = i;

  pnl_vect_resize(MktATMCapletVol->CapletVolatility, i);
  pnl_vect_resize(MktATMCapletVol->CapletMaturity, i);
}


// Delete caplets data
int DeleteMktATMCapletVolData(MktATMCapletVolData *MktATMCapletVol)
{
  pnl_vect_free(&(MktATMCapletVol->CapletMaturity));
  pnl_vect_free(&(MktATMCapletVol->CapletVolatility));

  return 1;
}

#endif //PremiaCurrentVersion
```