Help

```
#include "hullwhite2d_stdi.h"
#include "math/read_market_zc/InitialYieldCurve.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_integration.h"

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2009+2)
int CALC(CF_EuropeanSwaption_HW2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
static int CHK_OPT(CF_EuropeanSwaption_HW2D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
#else

static double xt, yt, a, b, sigma, eta, rho;
static double tau, Nominal, K;
static double omega;
static double option_mat, swap_mat;
static double mu_x, mu_y, sigma_x, sigma_y, rho_xy;
static double critical_y;
static int  nb_payement;

static PnlVect *Ci;

static void HW2dparams_to_G2dparams(double a, double b, double *sigma, double *e
{
  double sigma4, sigma3;

  sigma4 = (*eta) / fabs(a - b);
  sigma3 = sqrt(SQR((*sigma)) + SQR((*eta)) / SQR(a - b) - 2 * (*rho) * (*sigma)

  (*rho) = ((*sigma) * (*rho) - sigma4) / sigma3;
  (*eta) = sigma4;
  (*sigma) = sigma3;
}
```

```
static double V_func(double t, double T)
{
  return SQR(sigma) / SQR(a) * (T - t + 2. / a * exp(-a * (T - t)) - 1. / (2.*a)
         SQR(eta) / SQR(b) * (T - t + 2. / b * exp(-b * (T - t)) - 1. / (2.*b) *
         2.*rho * sigma * eta / (a * b) * (T - t + (exp(-a * (T - t)) - 1.) / a
}

static double log_A_func(ZCMarketData *ZCMarket, double t, double T)
{
  double VtT, V0T, V0t;
  double P0_t, P0_T;

  VtT = V_func(t, T);
  V0T = V_func(0, T);
  V0t = V_func(0, t);

  P0_T = BondPrice(T, ZCMarket);
  P0_t = BondPrice(t, ZCMarket);

  return log(P0_T / P0_t) + 0.5 * (VtT - V0T + V0t);
}

static double B_func(double z, double t, double T)
{
  return (1. / z) * (1. - exp(-z * (T - t)));
}

static double MxT(double s, double t, double T)
{
  return (SQR(sigma) / SQR(a) + rho * sigma * eta / (a * b)) * (1. - exp(-a * (t
         - (SQR(sigma) / (2 * SQR(a))) * (exp(-a * (T - t)) - exp(-a * (T + t -
         - (rho * eta * sigma / (b * (a + b))) * (exp(-b * (T - t)) - exp(-b * T
}

static double MyT(double s, double t, double T)
{
  return (SQR(eta) / SQR(b) + rho * sigma * eta / (a * b)) * (1. - exp(-b * (t -
         - (SQR(eta) / (2.*SQR(b))) * (exp(-b * (T - t)) - exp(-b * (T + t - 2 *
         - (rho * eta * sigma / (a * (a + b))) * (exp(-a * (T - t)) - exp(-a * T
}
```

```
static double h1(double x)
{
  return (critical_y - mu_y) / (sigma_y * sqrt(1. - SQR(rho_xy))) - rho_xy * (x
}

static double h2(double x, double ti)
{
  return h1(x) + B_func(b, option_mat, ti) * sigma_y * sqrt(1. - SQR(rho_xy));
}

static double log_lamda_func(ZCMarketData *ZCMarket, double x, double ti, int i)
{
  return  log(GET(Ci, i)) + log_A_func(ZCMarket, option_mat, ti) - B_func(a, opt
}

static double ki_func(double x, double ti)
{
  return -B_func(b, option_mat, ti) * (mu_y - 0.5 * (1. - SQR(rho_xy)) * SQR(sig
}

/*Computation of Critical Y*/
static double phiY(ZCMarketData *ZCMarket, double current, double x)
{
  int i;
  double sum, ti;

  sum = 0.;
  ti = option_mat;
  for (i = 0; i < nb_payement; i++)
    {
      ti += tau;
      sum += GET(Ci, i) * exp(log_A_func(ZCMarket, option_mat, ti) - B_func(a, o
    }

  return sum;
}

static double Critical_Y_Bisection(ZCMarketData *ZCMarket, double x)
{
  double y_high, y_low, y, y_step, phi_value, diff;
```

```
int i, nbr_iter;
double precision_Critical_Y_Bisection = 0.0001;

y_low = -B_func(a, option_mat, swap_mat) / B_func(b, option_mat, swap_mat) * x
phi_value = phiY(ZCMarket, y_low, x);

y_step = 2;

nbr_iter = 0;
while (phi_value > 1.)
  {
    nbr_iter++;
    y_low += y_step;
    phi_value = phiY(ZCMarket, y_low, x);
  }

y_high = y_low - y_step;
phi_value = phiY(ZCMarket, y_high, x);

if (phi_value < 1e-10)
  {
    y_step = 10;
  }

nbr_iter = 0;
while (phi_value < 1.)
  {
    nbr_iter++;
    y_low = y_high;
    y_high -= y_step;

    phi_value = phiY(ZCMarket, y_high, x);
  }

for (i = 0; i < 50; i++)
  {
    y = 0.5 * (y_high + y_low);

    phi_value = phiY(ZCMarket, y, x);

    diff = phi_value - 1.;
```

```
      if (fabs(diff) < precision_Critical_Y_Bisection)
        return y;

      if (diff > 0) y_high = y;
      else y_low = y;
    }

  return y;
}

// Function to integrate from ]-inf, inf[
static double integrand_fun(double x, void *ZCMarket)
{
  double d1, d2, sum, func_value;
  double ti;
  int i;

  critical_y = Critical_Y_Bisection((ZCMarketData *)ZCMarket, x);
  d1 = -omega * h1(x);
  sum = 0.;
  ti = option_mat;
  for (i = 0; i < nb_payement; i++)
    {
      ti += tau;
      d2 = -omega * h2(x, ti);
      sum += exp(ki_func(x, ti) + log_lamda_func((ZCMarketData *)ZCMarket, x, ti
    }

  func_value = exp(-0.5 * (x - mu_x) * (x - mu_x) / SQR(sigma_x)) / (sigma_x * s

  return func_value;
}

/*Payer Swaption momega=1, Receiver momega=-1*/
static double SWAPTION_g2(ZCMarketData *ZCMarket, double momega, double moption_
{
  double sum;
  double Tim, P0_Ti, P0_opmat;
  int i, neval, N;
  double integral;
```

```
double lim_sup, lim_inf, x, f_x, err;
PnlFunc func;

omega = momega;
option_mat = moption_mat;
tau = mtau;
swap_mat = mswap_mat;
Nominal = mNominal;
K = mK;
xt = mxt;
yt = myt;
a = ma;
b = mb;
sigma = msigma;
eta = meta;
rho = mrho;

nb_payement = (int)((swap_mat - option_mat) / tau);

P0_opmat = BondPrice(option_mat, ZCMarket);
//P0_swapmat = BondPrice(swap_mat, ZCMarket);

sum = 0.;
err = 0.;
Tim = option_mat;
for (i = 0; i < nb_payement; i++)
  {
    Tim += tau;
    P0_Ti = BondPrice(Tim, ZCMarket);
    sum += tau * P0_Ti;
  }
/*Compute Swap Rate*/
Ci =  pnl_vect_create_from_double(nb_payement, K * tau);
LET(Ci, nb_payement - 1) = 1 + K * tau;

/*Integral computation*/
mu_x = -MxT(0, option_mat, option_mat);
mu_y = -MyT(0, option_mat, option_mat);

sigma_x = sigma * sqrt((1. - exp(-2.*a * (option_mat))) / (2.*a));
sigma_y = eta * sqrt((1. - exp(-2.*b * (option_mat))) / (2.*b));
```

```
  rho_xy = rho * eta * sigma / ((a + b) * sigma_x * sigma_y) * (1. - exp(-(a + b

  // Truncate the domain of integration on the interval [lim_inf=mu_x - N1*sigma
  // such that |f(lim_inf)| and |f(lim_sup)| are small enough
  N = 0;
  do
    {
      N++;
      x = mu_x - N * sigma_x;
      f_x = integrand_fun(x, ZCMarket);
    }
  while (fabs(f_x) > 1e-8);
  lim_inf = x;

  N = 0;
  do
    {
      N++;
      x = mu_x + N * sigma_x;
      f_x = integrand_fun(x, ZCMarket);
      lim_sup = 0.0;
    }
  while (fabs(f_x) > 1e-8);
  lim_sup = x;

  neval = 0;

  func.F = integrand_fun;
  func.params = ZCMarket;

  pnl_integration_GK(&func, lim_inf, lim_sup, 0.0001, 0.00001, &integral, &err,

  pnl_vect_free(&Ci);

  return Nominal * omega * P0_opmat * integral;

}

/*Payer Swaption payer_receiver=1, Receiver payer_receiver=-1*/
int cf_ps_hw2d(int flat_flag, double flat_yield, char *curve, double Nominal, do
{
```

```
  double xt, yt;
  double payer_receiver;

  ZCMarketData ZCMarket;

  xt = 0.0;
  yt = 0.0;

  if (flat_flag == 0)
    {
      ZCMarket.FlatOrMarket = 0;
      ZCMarket.Rate = flat_yield;
    }

  else
    {
      ZCMarket.FlatOrMarket = 1;
      ZCMarket.filename = curve;
      ReadMarketData(&ZCMarket);
    }

  if ((p->Compute) == &Call)
    payer_receiver = -1.;
  else
    payer_receiver = 1.;

  // Transform the Hull/White model parameters to G2++ parameters.
  HW2dparams_to_G2dparams(a, b, &sigma, &eta, &rho);

  *price = SWAPTION_g2(&ZCMarket, payer_receiver, option_maturity, periodicity,

  DeleteZCMarketData(&ZCMarket);
  return OK;
}


int CALC(CF_EuropeanSwaption_HW2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
```

```
  return  cf_ps_hw2d(ptMod->flat_flag.Val.V_INT,
                     MOD(GetYield)(ptMod),
                     MOD(GetCurve)(ptMod),
                     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->aR.Val.V_DOUBLE,
                     ptMod->bu.Val.V_DOUBLE,
                     ptMod->SigmaR.Val.V_PDOUBLE,
                     ptMod->Sigmau.Val.V_PDOUBLE,
                     ptMod->Rho.Val.V_PDOUBLE,
                     ptOpt->PayOff.Val.V_NUMFUNC_1,
                     &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(CF_EuropeanSwaption_HW2D)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "PayerSwaption") == 0) || (strcmp(((Option
    return OK;
  else
    return WRONG;
}

#endif //PremiaCurrentVersion



static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->HelpFilenameHint = "CF_SwaptionHW2D";
      Met->init = 1;
    }

  return OK;
}

PricingMethod MET(CF_EuropeanSwaption_HW2D) =
```

```
{
  "CF_SwaptionHW2D",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(CF_EuropeanSwaption_HW2D),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(CF_EuropeanSwaption_HW2D),
  CHK_ok,
  MET(Init)
} ;
```