Help

```
#include  "
href../../../mod/libor_affine_cir1d/libor_affine_cir1d_stdi/libor_affine_cir1d_s
#include "
href../../../common/math/libor_affine_model/libor_affine_framework_h_src.pdfmath
#include "
href../../../common/math/libor_affine_model/libor_affine_pricing_h_src.pdfmath/l
#include "
href../../../common/math/libor_affine_model/libor_affine_models_h_src.pdfmath/li
#include "pnl/pnl_root.h"
#include "pnl/pnl_cdf.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2011+2) //The "#els
static int CHK_OPT(CF_LibAffCir1d_Direct_Swaption)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(CF_LibAffCir1d_Direct_Swaption)(void *Opt, void *Mod, PricingMethod *Me
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


///********* Static Variables ***********///
static double Ti;
static double Tm;
static double TN;
static PnlVect *c_k;
static PnlVect *Phi_i_k;
static PnlVect *Psi_i_k;

static double find_Y_caplet(StructLiborAffine *LiborAffine)
{
  double phi_1, psi_1, phi_2, psi_2;

  phi_1 = GET(Phi_i_k, 0);
  psi_1 = GET(Psi_i_k, 0);

  phi_2 = GET(Phi_i_k, 1);
```

```
  psi_2 = GET(Psi_i_k, 1);

  return -(phi_2 - phi_1 + log(GET(c_k, 1))) / (psi_2 - psi_1);

}

static double func_payoff(double x, void *LiborAffine)
{
  int i, m, k;
  double term_k, sum = 0., sum_der = 0.;
  double phi_i, psi_i, phi_k, psi_k;

  i = indiceTimeLiborAffine((StructLiborAffine *)LiborAffine, Ti);
  m = indiceTimeLiborAffine((StructLiborAffine *)LiborAffine, Tm);

  phi_i = GET(Phi_i_k, 0);
  psi_i = GET(Psi_i_k, 0);

  for (k = i + 1; k <= m; k++)
    {
      phi_k = GET(Phi_i_k, k - i);
      psi_k = GET(Psi_i_k, k - i);

      term_k = GET(c_k, k - i) * exp((phi_k - phi_i) + (psi_k - psi_i) * x);
      sum += term_k;
      sum_der += psi_k * term_k;
    }

  return 1. - sum;
}

static double find_Y_swaption(StructLiborAffine *LiborAffine)
{
  double tol = 1e-9;
  double x_inf, x_sup = PNL_NEGINF, root;
  PnlFunc func;
  double phi_1, psi_1, phi_2, psi_2;
  int k, i = indiceTimeLiborAffine(LiborAffine, Ti);
  int m = indiceTimeLiborAffine(LiborAffine, Tm);

  phi_1 = GET(Phi_i_k, 0);
```

```
  psi_1 = GET(Psi_i_k, 0);
  phi_2 = GET(Phi_i_k, m - i);
  psi_2 = GET(Psi_i_k, m - i);
  x_inf = -(phi_2 - phi_1) / (psi_2 - psi_1);

  for (k = i + 1; k <= m; k++)
    {
      phi_2 = GET(Phi_i_k, k - i);
      psi_2 = GET(Psi_i_k, k - i);

      x_sup = MAX(x_sup, (-log((m - i) * GET(c_k, k - i)) - (phi_2 - phi_1)) / (
    }

  func.F = func_payoff;
  func.params = LiborAffine;

  root = pnl_root_brent(&func, x_inf, x_sup, &tol);

  return root;
}

//swaption_payer_receiver=0 : Payer
//swaption_payer_receiver=1 : Receiver
static double cf_swaption_direct(StructLiborAffine *LiborAffine, double swaption
{
  double x0, lambda, theta, eta, Sqr_eta, Y;
  double P = 0., Q = 0., x, deg_freedom, n_centrality_param, bound, price, trm_k
  double Tk, a_Ti, b_Ti, dzeta_k, sigma_k, sum = 0.;
  int i, m, k, which = 1, status;
  double psi_d;
  dcomplex uk, phi, psi;

  x0     = GET(LiborAffine->ModelParams, 0);
  lambda = GET(LiborAffine->ModelParams, 1);
  theta  = GET(LiborAffine->ModelParams, 2);
  eta    = GET(LiborAffine->ModelParams, 3);
  Sqr_eta = SQR(eta);

  // Static variables
  Ti = swaption_start;
  Tm = swaption_end;
```

```
TN = LET(LiborAffine->TimeDates, (LiborAffine->TimeDates)->size - 1);

i = indiceTimeLiborAffine(LiborAffine, Ti);
m = indiceTimeLiborAffine(LiborAffine, Tm);

c_k = pnl_vect_create_from_double(m - i + 1, swaption_period * swaption_strike
Phi_i_k = pnl_vect_create(m - i + 1);
Psi_i_k = pnl_vect_create(m - i + 1);

LET(c_k, 0) = -1.0;
LET(c_k, m - i) += 1.;

for (k = i; k <= m; k++)
  {
    uk = Complex(GET(LiborAffine->MartingaleParams, k), 0.);
    phi_psi_t_v(TN - Ti, uk, LiborAffine, &phi, &psi);

    LET(Phi_i_k, k - i) = Creal(phi);
    LET(Psi_i_k, k - i) = Creal(psi);
  }
// Zero of the function f
if (m == i + 1) Y = find_Y_caplet(LiborAffine);
else Y = find_Y_swaption(LiborAffine);

a_Ti = exp(-lambda * Ti);
if (lambda == 0.) b_Ti = Ti;
else b_Ti = (1. - a_Ti) / lambda;

deg_freedom = lambda * theta / Sqr_eta;

sum = 0.;
Tk = Ti;
for (k = i; k <= m; k++)
  {
    psi_d = GET(Psi_i_k, k - i);

    dzeta_k = 1 - 2 * Sqr_eta * b_Ti * psi_d;
    sigma_k = Sqr_eta * b_Ti / dzeta_k;

    x = Y / sigma_k;
```

```
      n_centrality_param = x0 * a_Ti / (Sqr_eta * b_Ti * dzeta_k);

      if (x < 0)
        {
          P = 0.;
          Q = 1.;
        }
      else
        {
          pnl_cdf_chn(&which, &P, &Q, &x, &deg_freedom, &n_centrality_param, &st
        }

      if (swaption_payer_receiver == 0)
        trm_k = -GET(c_k, k - i) * BondPrice(Tk, LiborAffine->ZCMarket) * Q;
      else
        trm_k = GET(c_k, k - i) * BondPrice(Tk, LiborAffine->ZCMarket) * P;

      sum += trm_k;
      Tk += swaption_period;
    }

  price = swaption_nominal * sum;

  pnl_vect_free(&c_k);
  pnl_vect_free(&Phi_i_k);
  pnl_vect_free(&Psi_i_k);

  return price;
}


static int cf_swaption_direct_libaff_cir1d(int InitYieldCurve_flag, double R_fla
{
  StructLiborAffine LiborAffine;
  ZCMarketData ZCMarket;
  PnlVect *ModelParams = pnl_vect_create(4);

  ZCMarket.filename = curve;
  SetInitYieldCurve(InitYieldCurve_flag, R_flat, &ZCMarket);

  LET(ModelParams, 0) = x0;
```

```
  LET(ModelParams, 1) = lambda;
  LET(ModelParams, 2) = theta;
  LET(ModelParams, 3) = eta;

  CreateStructLiborAffine(&LiborAffine, &ZCMarket, swaption_start, swaption_end,

  *swaption_price = cf_swaption_direct(&LiborAffine, swaption_start, swaption_en

  FreeStructLiborAffine(&LiborAffine);

  return OK;
}


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

int CALC(CF_LibAffCir1d_Direct_Swaption)(void *Opt, void *Mod, PricingMethod *Me
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  int swaption_payer_receiver = (((ptOpt->PayOff.Val.V_NUMFUNC_1)->Compute) == &

  return  cf_swaption_direct_libaff_cir1d(ptMod->flat_flag.Val.V_INT,
                                          MOD(GetYield)(ptMod),
                                          MOD(GetCurve)(ptMod),
                                          ptMod->x0.Val.V_DOUBLE,
                                          ptMod->lambda.Val.V_PDOUBLE,
                                          ptMod->theta.Val.V_DOUBLE,
                                          ptMod->eta.Val.V_PDOUBLE,
                                          ptOpt->OMaturity.Val.V_DATE - ptMod->T
                                          ptOpt->BMaturity.Val.V_DATE - ptMod->T
                                          ptOpt->ResetPeriod.Val.V_DATE,
                                          ptOpt->FixedRate.Val.V_PDOUBLE,
                                          ptOpt->Nominal.Val.V_PDOUBLE,
                                          swaption_payer_receiver,
                                          &(Met->Res[0].Val.V_DOUBLE));
}
static int CHK_OPT(CF_LibAffCir1d_Direct_Swaption)(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->init = 1;
      Met->HelpFilenameHint = "cf_libor_affine_cir1d_swaption_direct";
    }
  return OK;
}


PricingMethod MET(CF_LibAffCir1d_Direct_Swaption) =
{
  "CF_LibAffCir1d_Direct_Swaption",
  {{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(CF_LibAffCir1d_Direct_Swaption),
  {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CHK_OPT(CF_LibAffCir1d_Direct_Swaption),
  CHK_ok,
  MET(Init)
} ;
```