Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2011+2) //The "#els
#else

#include "pnl/pnl_complex.h"
#include "pnl/pnl_root.h"
#include "pnl/pnl_integration.h"

#include "libor_affine_framework.h"
#include "libor_affine_pricing.h"

///********* 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 R;
static double Y;
static double ScalingFactor;

double SwapValue(double T_start, double T_end, double period, double strike, dou
{
  int k, nb_payement = pnl_iround((T_end - T_start) / period);
  double Tk, swap_value;

  Tk = T_start;
  swap_value = 0.;
  for (k = 1; k < nb_payement; k++)
    {
      Tk += period;
      swap_value += BondPrice(Tk, ZCMarket);
    }

  swap_value *= period * strike;

  swap_value = BondPrice(T_start, ZCMarket) - swap_value - (1 + period * strike)

  return nominal * swap_value;
```

```
}

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;
}

static double func_intg_swaption(double v, void *LiborAffine)
{
  double phi_i_k, psi_i_k, term_k, tmp_real, tmp_imag;
  int i, m, k;
  double x0 = GET(((StructLiborAffine *)LiborAffine)->ModelParams, 0);
  dcomplex phi_z, psi_z, z0, z1, z2, z3, z4, sum = CZERO;

  v /= ScalingFactor;

  i = indiceTimeLiborAffine(LiborAffine, Ti);
```

```
  m = indiceTimeLiborAffine(LiborAffine, Tm);

  z0 = Complex(R, -v);
  phi_psi_t_v(Ti, z0, LiborAffine, &phi_z, &psi_z);

  z1 = Complex(0, v * Y);
  z2 = Cadd(phi_z, CRmul(psi_z, x0));
  z3 = Cadd(z1, z2);

  tmp_real = Creal(z3);
  tmp_imag = Cimag(z3);

  z4 = Complex(cos(tmp_imag), sin(tmp_imag));

  for (k = i; k <= m; k++)
    {
      phi_i_k = GET(Phi_i_k, k - i);
      psi_i_k = GET(Psi_i_k, k - i);

      term_k = GET(c_k, k - i) * exp(tmp_real + phi_i_k + (psi_i_k - R) * Y);

      z1 = Complex(psi_i_k - R, v);
      z2 = RCdiv(term_k, z1);

      sum = Cadd(sum, z2);
    }

  z4 = Cmul(z4, sum);

  return Creal(z4) / ScalingFactor;
}

double cf_swaption_fourier_libaff(StructLiborAffine *LiborAffine, double swaptio
{
  double Tk, swap_value, R_min, R_max, v_inf, v_sup, swaption_price_fr = 0., coe
  int i, m, k, neval;
  dcomplex uk, phi_k, psi_k;
  PnlFunc func;

  // Static Variables
  Ti = swaption_start;
```

```
Tm = swaption_end;
TN = GET(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.;
LET(c_k, m - i) += 1.;

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

    LET(Phi_i_k, k - i) = Creal(phi_k);
    LET(Psi_i_k, k - i) = Creal(psi_k);

    swap_value += -GET(c_k, k - i) * BondPrice(Tk, LiborAffine->ZCMarket);
    Tk += swaption_period;
  }
swap_value *= swaption_nominal;

R_max = (LiborAffine->MaxMgfArg)(LiborAffine->ModelParams, Ti);
R_min = GET(Psi_i_k, 0);
if (R_min > R_max)
  {
    printf(" Warning: Fourier method can't be used in this case!\ n");
    abort();
  }

coeff_damping = MAX(1e-7, MIN(0.5, R_min / R_max));
R = (1 - coeff_damping) * R_min + coeff_damping * R_max;

if (m == i + 1) Y = find_Y_caplet(LiborAffine);
else Y = find_Y_swaption(LiborAffine);
```

```
  ScalingFactor = MAX(100., fabs(Y));

  func.F = func_intg_swaption;
  func.params = LiborAffine;

  v_inf = 0.;
  v_sup = PNL_POSINF;
  pnl_integration_qag(&func, v_inf, v_sup, 1e-8, 1e-8, 1500, &swaption_price_fr,
  swaption_price_fr *= 2 * swaption_nominal * BondPrice(TN, LiborAffine->ZCMarke

  // Receiver case.
  if (swaption_payer_receiver == 1) swaption_price_fr = swaption_price_fr - swap

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

  return swaption_price_fr;
}

#endif
```