Help

```
#include "
href../../../mod/doublehes1d/doublehes1d_vol/doublehes1d_vol_h_src.pdfhes1d_vol.
#include "pnl/pnl_vector.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_mathtools.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2016+2) //The "#els
  static int CHK_OPT(AP_HESCHIARELLA_VARIANCESWAP)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_HESCHIARELLA_VARIANCESWAP)(void *Opt, void *Mod, PricingMethod *Me
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else

/**
 * Calculate the characteristic function for continuous sampling.
 */
static dcomplex charInfinite(double T, double r, double sigma, double delta, dou
                      double theta, dcomplex phi, double V0)
{
    // To calculate the characteristic function for continuous sampling
    dcomplex gamaPhi = Csqrt(RCsub(k*k, CRmul(phi, 2.*sigma*sigma / T)));

    dcomplex two_x_gamaPhi = RCmul(2., gamaPhi);
    dcomplex gamaPhi_p_k = CRadd(gamaPhi, k);
    dcomplex exp_gamaPhiT_m_1 = CRsub(Cexp(CRmul(gamaPhi, T)), 1.);
    dcomplex aux = Cadd(Cmul(gamaPhi_p_k, exp_gamaPhiT_m_1), two_x_gamaPhi);

    dcomplex A = RCmul(2.*k*theta/(sigma*sigma),
                       Clog( Cdiv(Cmul(two_x_gamaPhi, Cexp(CRmul(gamaPhi_p_k, T
                                  aux) ));

    dcomplex B = Cdiv(Cmul(RCmul(2., phi), exp_gamaPhiT_m_1), RCmul(T, aux));

    dcomplex x = Cexp(Cadd(A, CRmul(B, V0)));
```

```
    return x;
}

/**
 * Calculate the characteristic function for discrete sampling.
 *
 * @param [in] phi
 * @param [in] N
 * @param [out] x
 */
static dcomplex charFinite(double T, double r, double sigma, double delta, doubl
                    double theta, double rho, double V0, dcomplex phi, int N)
{
    double dt = T / N; // time step
    double l = 0.001; // bump method to calculate g (phi)

    double u0 = (1. - exp(-2.*k*T)*V0*V0) / (8.*k*T) + (1. - 2.*exp(-k*T) +
                exp(-2.*k*T))*V0*theta / (4.*k*T) - sigma*(2.*sigma*exp(-k*T) -
                sigma - 4.*rho*k*exp(-k*T) + 4.*rho*k - sigma*exp(-2.*k*T))*V0 /
                (8.*k*k*T) + (2.*k*k*T*theta*theta + theta*sigma*(sigma - 4.*rho
                (2.*theta*sigma*(sigma - 2.*rho*k) + 4.*k*theta*theta)*(exp(-k*T
                (sigma*sigma*theta / 2. + k*theta*theta)*(exp(-2.*k*T) - 1.)) /
                (8.*k*k*T);

    double c2 = (1. - exp(-2.*k*T))*V0*V0 / (2.*k) +
                (sigma*sigma + 2.*theta*k)*(1. - 2.*exp(-k*T) + exp(-2.*k*T))*V0
                theta*(sigma*sigma + 2.*theta*k)*(2.*k*T - 3. - exp(-2.*k*T) +
                4.*exp(-k*T)) / (4.*k*k);

    // May need to change calculation of g
    dcomplex g = CRdiv(Csub(charInfinite(T, r, sigma, delta, k, theta, CRadd(phi
                            charInfinite(T, r, sigma, delta, k, theta, CRsub(phi
                       2.*l);

    // gives the characteristic function of the RV.
    dcomplex x = Cadd(Cmul(charInfinite(T, r, sigma, delta, k, theta, phi, V0),
                           RCadd(1., Cadd(CRmul(RCmul(dt, phi), pow(r - delta, 2
                                                RCmul(dt, Cadd(CRmul(phi, u0),
                                                               CRmul(Cmul(phi, phi), c
                      RCmul(dt, CRmul(Cmul(Cminus(g), phi), (r - delta) / T)));
```

```
    return x;
}

/**
 * Calculate the final price of variance and volatility swaps.
 *
 * @param [in] var
 * @param [in] N
 */

  static int ap_heschiarella_varswap(double V0, double k, double theta, double s
  int N, double *varSwapPrice)
  {
double sum_Aj_I;
int i;
    double a = 0.;
    // Expression to get Expectation(RV)
    double Kf = 1. / (8.*N*pow(k, 3.)*T) * (N*(pow(sigma, 2.)*(theta - 2.*V0) +
                2.*k*pow(V0 - theta, 2.)) * (exp(-2.*k*T) - 1.) * (1. - exp(k*T/
                (1. + exp(k*T/N)) + 2.*k*T*(k*k*T*pow(theta - 2.*r, 2.) +
                N*theta*(4.*k*k - 4.*rho*k*sigma + pow(sigma, 2.))) +
                4.*(V0 - theta)*(N*(2.*k*k + pow(sigma, 2.) - 2.*rho*k*sigma) +
                k*k*T*(theta - 2.*r))*(1. - exp(-k*T)) -
                2.*N*N*theta*sigma*(sigma - 4.*rho*k)*(1. - exp(-k*T/N)) +
                4.*(V0 - theta)*k*T*sigma*(sigma - 2.*rho*k)*(1. - exp(-k*T)) /
                (1. - exp(-k*T/N)));
    double b = Kf*100.;

    int size_j = 299;
    PnlVect* j = pnl_vect_create_from_scalar(size_j, 1.); // Is the limit J = 30
    PnlVect* Aj = pnl_vect_create(size_j + 1);
    PnlVect* I = pnl_vect_create(size_j + 1);
    pnl_vect_cumsum(j);
    LET(Aj, 0) = 1. / (b - a);
    LET(I, 0) = (b*b - a*a) / 2.;

    sum_Aj_I = GET(Aj, 0) * GET(I, 0);

    for (i = 1; i <= size_j; i++)
    {
        LET(Aj, i) = 2. / (b-a) * Creal(Cmul(
```

```
            charFinite(T, r, sigma, delta, k, theta, rho, V0,
                       Complex(0., GET(j, i-1)*M_PI/(b - a)), N),
            Cexp(Complex(0., -GET(j, i-1)*a*M_PI / (b - a)))));

        // analytical expression for Variance swaps
        LET(I, i) = pow(b - a, 2.) * (-1. + cos(M_PI*GET(j, i-1))) /
                    (M_PI*M_PI*pow(GET(j, i-1), 2.)) +
                    (b - a)*(b - 0.05)*sin(GET(j, i-1)*M_PI) / (GET(j, i-1)*M_PI

        sum_Aj_I += GET(Aj, i) * GET(I, i);
    }

    // Variance swap price
    *varSwapPrice = 10000. * ( sum_Aj_I - pow(Strike/100., 2.) ) * exp(-r*T);

    pnl_vect_free(&j);
    pnl_vect_free(&Aj);
    pnl_vect_free(&I);
    return OK;
}


int CALC(AP_HESCHIARELLA_VARIANCESWAP)(void *Opt, void *Mod, PricingMethod *Me
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid, strike;
    NumFunc_1 *p;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
    p = ptOpt->PayOff.Val.V_NUMFUNC_1;
    strike = p->Par[0].Val.V_DOUBLE/100.;

    return ap_heschiarella_varswap(
             ptMod->Sigma0.Val.V_PDOUBLE
             , ptMod->MeanReversion.Val.V_PDOUBLE,
             ptMod->LongRunVariance.Val.V_PDOUBLE,
             ptMod->Sigma.Val.V_PDOUBLE,
             ptMod->Rho.Val.V_PDOUBLE,
             r, divid,
```

```
            ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
            strike,
  Met->Par[0].Val.V_INT,
            &(Met->Res[0].Val.V_DOUBLE));

  }

  static int CHK_OPT(AP_HESCHIARELLA_VARIANCESWAP)(void *Opt, void *Mod)
  {
    if ((strcmp(((Option *)Opt)->Name, "VarianceSwap") == 0))
      return OK;

    return WRONG;
  }

#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
 if (Met->init == 0)
    {
      Met->Par[0].Val.V_INT = 10;
      Met->init = 1;
      Met->HelpFilenameHint = "ap_hes_chiarellavarianceswap";
    }
  return OK;
    return OK;
  }

  PricingMethod MET(AP_HESCHIARELLA_VARIANCESWAP) =
  {
    "AP_HES_CHIARELLAVARIANCESWAP",
    {  {"Numbero of Sampling Steps", INT, {100}, ALLOW},  {" ", PREMIA_NULLTYPE,
    CALC(AP_HESCHIARELLA_VARIANCESWAP),
    { {"Price in 10000 variance points", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_HESCHIARELLA_VARIANCESWAP),
    CHK_ok ,
    MET(Init)
  } ;
```

```
/*////////////////////////////////////////*/
```