Help

```
#include  "
href../../../mod/bergomi2d/bergomi2d_std/bergomi2d_std_h_src.pdfbergomi2d_std.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_list.h"
#include "pnl/pnl_integration.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_basis.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2012+2) //The "#els
static int CHK_OPT(AP_EXPANSION_OA)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_EXPANSION_OA)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

//-----------------------------------Hermite polynomials-----------------------
static double HermitePoly(int n, double x)
{
  switch (n)
    {
    case 0:
      return 1;
      break;
    case 1:
      return x;
      break;

    case 2 :
      return SQR(x) - 1;
      break;

    case 3:
```

```
      return POW(x, 3) - 3 * x;
      break;

    case 4:
      return POW(x, 4) - 6 * POW(x, 2) + 3;
      break;
    case 5:

      return POW(x, 5) - 10 * POW(x, 3) + 15 * x;
      break;
    case 6:
      return POW(x, 6) - 15 * POW(x, 4) + 45 * POW(x, 2) - 15 ;
      break;
    default:
      return 0;
      break;
    }
}

//-------------- supplementary functions to calculate the nu functions----------
static double funct_supMul(int i, double t, double m, double sigma, double k, do
{
  switch (i)
    {
    case 1://A
      return m * SQR(sigma) * (t / (2.0 * k) - (1 - exp(-2 * k * t)) / SQR(2.0 *
      break;
    case 2://B
      return m * sigma * m * sigma * (2.0 * k * t - exp(-2.0 * k * t)  + 4.0 * e
      break;
    case 3://C
      return rho * POW(sqrt(m), 3) * sigma * (t / (k) - (1 - exp(-k * t)) / SQR(
      break;
    case 4://D
      return SQR(rho * m * sigma / k) * (t * (1 + exp(-k * t)) + 2 * (exp(-k * t
      break;
    case 5://E
      return rho * sigma * sqrt(m) * (1 - exp(-k * t)) / k;
      break;
    default:
      return 0;
```

```
      break;
    }
}

double funct_GMul(double t, double rho1, double rho2, double k1, double k2, doub
{
  return sigma1 * sigma2 * rho1 * rho2 * ((k1 * (k1 + k2) * exp(k1 * t) + k2 * (
}
//------------------------------ Nu functions ------------------------------
double funct_nuMul(double omega, int n, double t, double m, PnlVect *sigma, PnlV
{
  double A = 0, B = 0, C = 0, D = 0, E = 0, I = 1;
  int i, j;
  int l = sigma->size;
  for (i = 0; i < l; i++)
    {
      A += funct_supMul(1, t, m, pnl_vect_get(sigma, i), pnl_vect_get(k, i), pnl
      B += funct_supMul(2, t, m, pnl_vect_get(sigma, i), pnl_vect_get(k, i), pnl
      C += funct_supMul(3, t, m, pnl_vect_get(sigma, i), pnl_vect_get(k, i), pnl
      D += funct_supMul(4, t, m, pnl_vect_get(sigma, i), pnl_vect_get(k, i), pnl
    }
  for (i = 0; i < l; i++)
    {
      for (j = 0; j < l; j++)
        {
          E += funct_GMul(t, pnl_vect_get(rho, i), pnl_vect_get(rho, j), pnl_vec
        }
    }
  I = E * SQR(m);
  switch (n)
    {
    case 0:
      return m * t / 2 + SQR(omega / 2) * A;
      break;
    case 1:
      return 0;
      break;
    case 2:
      return SQR(omega / 2) * B - (omega / 2) * C;
      break;
    case 3:
```

```
      return SQR(omega) * B / 2 - omega * C / 2 + SQR(omega / 2) * D + SQR(omega
      break;
    case 4:
      return SQR(omega / 2) * B + SQR(omega * C / 2) / 2 + SQR(omega / 2) * D +
      break;
    case 5:
      return SQR(omega * C / 2);
      break;
    case 6:
      return SQR(omega * C / 2) / 2;
      break;
    default:
      return 0;
      break;
    }
}
//--------------------------------------Call option pricing --------------------
static double CallPriceMul(double omega, double K, double S, double t, double m,
{
  double z0, z1, z2, z3, z4;
  double d1, d2, v, st, CB;

  z4 = funct_nuMul(omega, 6, t, m, sigma, k, rho);
  z3 = -funct_nuMul(omega, 5, t, m, sigma, k, rho) + z4;
  z2 = funct_nuMul(omega, 4, t, m, sigma, k, rho) + z3;
  z1 = -funct_nuMul(omega, 3, t, m, sigma, k, rho) + z2;
  z0 = funct_nuMul(omega, 2, t, m, sigma, k, rho) + z1;
  //----------------- We distinguish two cases, K=0.0 and K> 0.0:
  //-----------------case 1, K=0.0:
  if (K == 0.0)
    {
      return S * (1 + z0);
    }
  //-----------------case 2, K>0.0:
  else
    {
      v = 2 * funct_nuMul(omega, 0, t, m, sigma, k, rho);
      d1 = (log(S / K) + v / 2) / sqrt(v);
      d2 = (log(S / K) - v / 2) / sqrt(v);
      st = z0 * HermitePoly(0, -d2) +  z1 * HermitePoly(1, -d2) / pow(sqrt(v), 1
      CB = S * cdf_nor(d1) - K * cdf_nor(d2);
```

```
      //------------------------------------------------------------------------
      return CB + S * cdf_nor(d1) * z0 + K * pnl_normal_density(d2) * st / sqrt(
    }
}
//--------------------------------------the Greek: Delta dCallprice/dS----------
static double DeltaMul(double omega, double K, double S, double t, double m, Pnl
{
  double z0, z1, z2, z3, z4;
  double d1, d2;
  double v, st, std;

  z4 = funct_nuMul(omega, 6, t, m, sigma, k, rho);
  z3 = -funct_nuMul(omega, 5, t, m, sigma, k, rho) + z4;
  z2 = funct_nuMul(omega, 4, t, m, sigma, k, rho) + z3;
  z1 = -funct_nuMul(omega, 3, t, m, sigma, k, rho) + z2;
  z0 = funct_nuMul(omega, 2, t, m, sigma, k, rho) + z1;
  //----------------- We distinguish two cases, K=0.0 and K> 0.0:
  //-----------------case 1, K=0.0:
  if (K == 0.0)
    {
      return (1 + z0);
    }
  //-----------------case 2, K>0.0:
  else
    {

      v = 2 * funct_nuMul(omega, 0, t, m, sigma, k, rho);
      d1 = (log(S / K) + v / 2) / sqrt(v);
      d2 = (log(S / K) - v / 2) / sqrt(v);
      st = z0 * HermitePoly(0, -d2) +  z1 * HermitePoly(1, -d2) / pow(sqrt(v), 1
      //----------------------- derivative of st with respect to S is std
      std = -(z0 * 0.0 +  z1 * HermitePoly(0, -d2) / pow(sqrt(v), 1) +  2 * z2 *

      return cdf_nor(d1) * (1 + z0) + pnl_normal_density(d1) * (z0 + 1) / (sqrt(
    }
}

int ApExpansionOA(double S0, NumFunc_1  *p, double t, double r, double q, double
{
  double K;
  int flag_call;
```

```
  double K0;
  PnlVect *kappa, *rho, *sigma;


  K = p->Par[0].Val.V_PDOUBLE;
  if ((p->Compute) == &Call)
    flag_call = 1;
  else
    flag_call = 0;;

  kappa = pnl_vect_create(2);
  pnl_vect_set(kappa, 0, kappa1);
  pnl_vect_set(kappa, 1, kappa2);

  rho = pnl_vect_create(2);
  pnl_vect_set(rho, 0, rhoSx);
  pnl_vect_set(rho, 1, rhoSy);

  sigma = pnl_vect_create_from_double(2, 1.0);
  pnl_vect_set(sigma, 0, theta);
  pnl_vect_set(sigma, 1, 1 - theta);


//v= 2*funct_nuMul(omega,0,t,csi0,sigma,kappa,rho);
  K0 = K * exp(-(r - q) * t); // discounted K

//Call case
  *ptprice = exp(-q * t) * CallPriceMul(omega, K0, S0, t, csi0, sigma, kappa, rh
  *ptdelta = exp(-q * t) * DeltaMul(omega, K0, S0, t, csi0, sigma, kappa, rho);

//Put Case
  if (flag_call == 0)
    {
      *ptprice = *ptprice - S0 * exp(-q * t) + K * exp(-r * t);
      *ptdelta = *ptdelta - exp(-q * t);
    }

  pnl_vect_free(&kappa);
  pnl_vect_free(&rho);
  pnl_vect_free(&sigma);
```

```
  return OK;
}

int CALC(AP_EXPANSION_OA)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

  return ApExpansionOA(ptMod->S0.Val.V_PDOUBLE,
                       ptOpt->PayOff.Val.V_NUMFUNC_1,
                       ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                       r
                       , divid,
                       ptMod->csi0.Val.V_PDOUBLE,
                       ptMod->omega.Val.V_PDOUBLE,
                       ptMod->theta.Val.V_PDOUBLE
                       , ptMod->k1.Val.V_PDOUBLE,
                       ptMod->k2.Val.V_PDOUBLE,
                       //ptMod->rhoxy.Val.V_RGDOUBLE,
                       ptMod->rhoSx.Val.V_RGDOUBLE,
                       ptMod->rhoSy.Val.V_RGDOUBLE,
                       &(Met->Res[0].Val.V_DOUBLE),
                       &(Met->Res[1].Val.V_DOUBLE));
}

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

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
```

```
      Met->HelpFilenameHint = "AP_EXPANSION_Bergomi";
      Met->init = 1;
    }

  return OK;
}

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