Help

```
extern "C" {
#include  "hes1d_std.h"
}
#include "math/numerics.h"
#include <complex>
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_cdf.h"
#include "enums.h"

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

  static complex<double> I(0.0, 1.0);

  static double T, sigma, rho, k, v, r, divid, teta, S, strike;
  static long int countfuneval;

//=============================== CHARACT FUNCTION ===================
  static complex<double> charact_exp(double uu, double alpha)
  {
    double a, rs, sig, tau;
    complex<double> g, tpp, tpm, tpf1, DD, CN, ans, d, expo, xi, b;

    tau = T;
    a = k * teta;
    rs = rho * sigma;

    sig = sigma * sigma;
    xi = uu - I * alpha;
    b = k - I * rs * xi;
```

```
    d = sqrt(b * b + sig * xi * (xi + I));
    tpp = b + d;
    tpm = b - d;
    g = tpm / tpp;

    expo = exp(-tau * d);
    DD = tpm / sig * (1.0 - expo) / (1.0 - g * expo);

    CN = (1.0 - g * expo) / (1.0 - g);
    tpf1 = a / sig * (tau * tpm - 2.0 * log(CN)) + I * (r - divid) * xi * tau +

    ans = tpf1 + v * DD;

    return ans;
  }

  /* static complex<double> charact_funct(double uu, double alpha)
   * {
   *
   *   complex<double> ans;
   *
   *   ans=exp(charact_exp(uu, alpha));
   *
   *   return ans;
   * } */


  static double mgf(int ind, double spot, double strk, double ti, double ri, dou
  {
    return real(charact_exp(0., alpha + 1.)) - ind * (log(spot) + ri * ti); //te
  }

//================================ OPTIMAL ALPHA =======================
  static double funtozero_c(int ind, double logs, double logk, double ti, double
  {
    double mu, a, c, p;
    complex<double> ip, g, exp0, exp1, znam, fac;
    complex<double> cder, pder, gder;
    complex<double> fun;
```

```
  mu = ka / sigma2;
  a = (ri - dividi) * ti + logs - logk;
  c = mu - rhow * (alpha + 1.);
  cder = -rhow;
  p = c * c - alpha * (alpha + 1.);
  // p<0!
  p = sqrt(-p);
  ip = I * p;
  pder = I * ((1. - rhow * rhow) * alpha + (0.5 + rhow * (mu - rhow))) / p;
  g = c + ip;
  g = (c - ip) / g;
  gder = 2.*(cder * ip - pder * c) / ((c + ip) * (c + ip));
  fac = ti * sigma2 * pder;
  exp0 = exp(-ti * sigma2 * ip);
  exp1 = 1. - exp0;
  znam = 1. - g * exp0;

  fun = a + (mu * theta * ti + sigma0 / sigma2 * exp1 / znam) * (cder - pder);
  fun += sigma0 / sigma2 * (c - ip) * exp0 / znam * (fac * (1. - g) + exp1 * g
  fun -= 2.*mu * theta / sigma2 * (gder / (1. - g) * exp1 + g * fac * exp0) /

  return real(fun);
}

static double funtozero(int ind, double logs, double logk, double ti, double r
{
  double mu, a, c, p, g, exp0, exp1, znam, fac;
  double cder, pder, gder;
  double fun;

  countfuneval++;

  mu = ka / sigma2;
  a = (ri - dividi) * ti + logs - logk;
  c = mu - rhow * (alpha + 1.);
  cder = -rhow;
  p = c * c - alpha * (alpha + 1.);
  if (p < 0.)
    {
      return funtozero_c(ind, logs, logk, ti, ri, dividi, sigma0, ka, theta, s
    }
```

```
  if (p == 0.)
    {
      g = 2. + ti * sigma2 * c;
      fun = a - mu * theta * ti * rhow * (1. - 2. / g) - sigma0 * rhow * ti *
      return fun;
    }
  p = sqrt(p);
  pder = (-(1. - rhow * rhow) * alpha - (0.5 + rhow * (mu - rhow))) / p;
  g = c + p;
  if (g == 0.)
    {
      printf("\ n OOOPS!! c+p==0\ n");
      return 0.;
    }
  g = (c - p) / g;
  gder = 2.*(cder * p - pder * c) / ((c + p) * (c + p));
  fac = ti * sigma2 * pder;
  exp0 = exp(-ti * sigma2 * p);
  exp1 = 1. - exp0;
  znam = 1. - g * exp0;
  if (znam == 0.)
    {
      printf("\ n OOOPS!! znam==0\ n");
      return 0.;
    }
  if (1. - g == 0.)
    {
      printf("\ n OOOPS!! 1-g==0\ n");
      return 0.;
    }

  fun = a + (mu * theta * ti + sigma0 / sigma2 * exp1 / znam) * (cder - pder);
  fun += sigma0 / sigma2 * (c - p) * exp0 / znam * (fac * (1. - g) + exp1 * gd
  fun -= 2.*mu * theta / sigma2 * (gder / (1. - g) * exp1 + g * fac * exp0) /

  return fun;
}
static double secderiv(int ind, double spot, double strk, double ti, double ri
{
  double cf1, cf2, logs, logk;
```

```
  logs = log(spot);
  logk = log(strk);

  cf1 = funtozero(ind, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,
  cf2 = funtozero(ind, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,

  return 100.*(cf2 - cf1);
}
static double thirdderiv(int ind, double spot, double strk, double ti, double
{
  double cf1, cf2, cf3, logs, logk;

  logs = log(spot);
  logk = log(strk);

  cf1 = funtozero(ind, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,
  cf2 = funtozero(ind, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,
  cf3 = funtozero(ind, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,

  return 100.*(cf2 + cf3 - 2.*cf1);
}
static double fourthderiv(int ind, double spot, double strk, double ti, double
{
  double cf1, cf2, cf3;
  /*
   cf1 = thirdderiv(ind, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2
   cf2 = thirdderiv(ind, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2
  */
  cf1 = secderiv(ind, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, r
  cf2 = secderiv(ind, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, r
  cf3 = secderiv(ind, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, r

  return 100.*(cf2 + cf3 - 2.*cf1); //10.*(cf2-cf1);
}

static double spmapprox(int ind, double spot, double strk, double ti, double r
{
  double  logk;
  double uu, ww, deriv2, deriv3, prob, addterm;
  logk = log(strk);
```

```
// s-p for K
uu = alpha - ind + 1.;
if (uu == 0.)
  {
    deriv2 = secderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigm
    deriv3 = thirdderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, si
    prob = 0.5 - deriv3 / (6.*deriv2 * sqrt(2.*M_PI * deriv2));
    return prob;
  }
// s-p for Ko
ww = 2.0 * (uu * logk - mgf(ind, spot, strk, ti, ri, dividi, sigma0, ka, the
if (ww > 0.)
  {
    ww = sqrt(ww);
  }
else
  {
    printf("ww<0!\ n");
    ww = 1.;
  }
if (uu < 0.)
  {
    ww *= -1.;
  }
//sec deriv
deriv2 = secderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2,
deriv3 = thirdderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2
//deriv4 = fourthderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sig

// probability approx Luganini-Rice formula
if (deriv2 > 0.)
  {
    //znam = uu*sqrt(deriv2);
    addterm = 0.;//(deriv4/(deriv2*deriv2)/8. - 5.*deriv3*deriv3/(deriv2*der
    prob = 1. - cdf_nor(ww) + pnl_normal_density(ww) * (1.0 / (uu * sqrt(der
  }
else
  {
    printf("deriv2<0!\ n");
    prob = 0.;
  }
```

```
  return prob;
}

static double liebermanpar_heston(int ind, double spot, double strk, double ti
{
  double k1, k2, k3, k4, logs, logk, tt, z3;

  logs = log(spot);
  logk = log(strk);

  k1 = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2, rho
  k2 = secderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, rhow
  k3 = thirdderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, rh
  k4 = fourthderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, r

  tt = (logk - k1) / k2;

  z3 = tt * (1. - tt * (k3 / (2.*k2) - (k3 * k3 / (2.*k2 * k2) - k4 / (6.*k2))

  return z3;
}

static double glassermanpar_heston(int ind, double spot, double strk, double t
{
  double k1, k2, k3, k4, logs, logk, tt, z3;

  logs = log(spot);
  logk = log(strk);

  z3 = liebermanpar_heston(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, s
  k1 = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2, rho
  k2 = secderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, rhow
  k3 = thirdderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, rh
  k4 = fourthderiv(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigma2, r

  tt = (logk - k1) / k2;

  z3 += tt * (1. - tt * (k3 / (2.*k2) - (k3 * k3 / (2.*k2 * k2) - k4 / (6.*k2)

  return z3;
```

```
}

static double optimalpar_heston(double spot, double strk, double ti, double ri
{
  double minalpha, maxalpha, dsqr, mu;
  double logs, logk, term1, term3;
  double la, ra, ca, lf, rf, cf, h;

  logs = log(spot);
  logk = log(strk);
  mu = ka / sigma2;

  term1 = 0.5 + rhow * (mu - rhow);
  term3 = 1.0 - rhow * rhow;
  dsqr = sqrt(term1 * term1 + (mu - rhow) * (mu - rhow) * term3);
  minalpha = (-term1 - dsqr) / term3;
  maxalpha = (dsqr - term1) / term3;

  la = 0.99 * minalpha;
  lf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2, rho
  ra = 0.99 * maxalpha;
  rf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2, rho

  h = (ra - la) / 20.;
  ca = la + h;
  cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2, rho

  if ((lf * rf > 0.) && (lf > 0.))
    {
      //Searching s.p. to the left from minalpha
      ra = la;
      rf = lf;
      ca = ra - h;
      cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,

      while ((rf * cf > 0.))
        {
          ra = ca;
          rf = cf;
          ca -= h;
          cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sig
```

```
        }
      la = ca;
      lf = cf;
    }

  if ((lf * rf > 0.) && (rf < 0.))
    {
      //Searching s.p. to the right from maxalpha
      la = ra;
      lf = rf;
      ca = la + h;
      cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,

      while ((lf * cf > 0.))
        {
          la = ca;
          lf = cf;
          ca += h;
          cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sig
        }
      ra = ca;
      rf = cf;
    }
// Binary search
  ca = (la + ra) / 2.;
  cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2, rho

  while ((fabs(cf) > 0.00001) && (ca - la > 0.0000001) && (ra - ca > 0.0000001
    {
      if (lf * cf < 0.)
        {
          ra = ca;
          rf = cf;
        }
      else
        {
          la = ca;
          lf = cf;
        }
      ca = (la + ra) / 2.;
```

```
      cf = funtozero(1, logs, logk, ti, ri, dividi, sigma0, ka, theta, sigma2,
    }

  return ca;
}
//==================================================================
static int ap_spmHeston(int ifCall, double spot, double strk, double ti, doubl
{
  double optimalsigma, spmprice, prob0, prob1;

  T = ti;
  sigma = sigma2;
  rho = rhow;
  k = ka;
  v = sigma0;
  r = ri;
  divid = dividi;
  teta = theta;
  S = spot;
  strike = strk;

  countfuneval = 0;

  if (flag_saddlepoint == 1)
    {
      optimalsigma = optimalpar_heston(spot, strk, ti, ri, dividi, sigma0, ka,

      prob0 = spmapprox(0, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigm
      prob1 = spmapprox(1, spot, strk, ti, ri, dividi, sigma0, ka, theta, sigm
      spmprice = -(exp(-ri * ti) * strk * prob0 - spot * prob1);
      if (!ifCall)
        {
          spmprice = spmprice - spot + strike * exp(-r * ti);
        }
      *ptprice = spmprice;
    }
  else
    {
      optimalsigma = glassermanpar_heston(1, spot, strike, ti, r, divid, sigma
      prob0 = spmapprox(0, spot, strike, ti, r, divid, sigma0, ka, theta, sigm
      prob1 = spmapprox(1, spot, strike, ti, r, divid, sigma0, ka, theta, sigm
```

```
        spmprice = -(exp(-r * ti) * strike * prob0 - spot * prob1);
        if (!ifCall)
          {
            spmprice = spmprice - spot + strike * exp(-r * ti);
          }
        *ptprice = spmprice;
      }

    // Lieberman approx
    /* optimalsigma = liebermanpar_heston(1, spot, strike, ti, r, divid, sigma0,
     prob0=spmapprox(0, spot, strike, ti, r, divid, sigma0, ka, theta, sigma2, r
     prob1=spmapprox(1, spot, strike, ti, r, divid, sigma0, ka, theta, sigma2, r
     spmprice = -(exp(-r*ti)*strike*prob0 - spot*prob1);
     if (!ifCall)
        {
            spmprice = spmprice-spot+strike*exp(-r*ti);
     }
     */
    return OK;
  }

//==========================================================================

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

    NumFunc_1 *p;

    if (ptMod->Sigma.Val.V_PDOUBLE == 0.0)
      {
        Fprintf(TOSCREEN, "BLACK-SHOLES MODEL\ n\ n\ n");
        return WRONG;
      }
    else
      {
        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;
        ifcall = ((p->Compute) == &Call);

        return ap_spmHeston(ifcall, ptMod->S0.Val.V_PDOUBLE,
                            strike/*ptOpt->PayOff.Val.V_NUMFUNC_1*/,
                            ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                            r,
                            divid, 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,
                            Met->Par[0].Val.V_DOUBLE,
                            Met->Par[1].Val.V_ENUM.value,
                            &(Met->Res[0].Val.V_DOUBLE)
                           );
      }

  }

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

    return  WRONG;
  }

#endif //PremiaCurrentVersion

  static PremiaEnumMember SaddlepPointsMembers[] =
  {
    { "Exact", 1 },
    { "Glasserman Kim Approximation", 2 },
    { NULL, NULLINT }
  };

  static DEFINE_ENUM(SaddlepPoints, SaddlepPointsMembers);

  static int MET(Init)(PricingMethod *Met, Option *Opt)
```

```
  {
    if (Met->init == 0)
      {
        Met->Par[0].Val.V_PDOUBLE = 1.0;
        Met->Par[1].Val.V_ENUM.value = 1;
        Met->Par[1].Val.V_ENUM.members = &SaddlepPoints;
        Met->HelpFilenameHint = "ap_spm_heston";
        Met->init = 1;
      }

    return OK;
  }

  PricingMethod MET(AP_SPM_Heston) =
  {
    "AP_SaddlePoint_Heston",
    {{"Sigma parameter", PDOUBLE, {100}, ALLOW}, {"Saddlepoint method", ENUM, {1
    CALC(AP_SPM_Heston),
    { {"Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(AP_SPM_Heston),
    CHK_ok,
    MET(Init)
  };

}
```