Help

```
#include  "hesvasicek2d_std.h"
#include "enums.h"
#include "error_msg.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_integration.h"

//The "#else" part of the code will be freely available after the (year of creat
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(AP_FOURIERCOSINE_HESVAS2D)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(AP_FOURIERCOSINE_HESVAS2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double u0, kappa0, vbar0, rho120, rho130, rho140, rho340, gammav0, rho230
static dcomplex fd0, fg0;

static void funcB(double u, dcomplex *result)
{
  *result = CRmul(CI, u);
}
static void funcd(double u, double kappa, double gamma, double rho12, dcomplex *
{
  if (kappa == 0.0 || gamma == 0.0)      *result = CZERO;
  else    *result = Csqrt(Csub(Cpow_real(CRsub(CRmul(CI, rho12 * gamma * u), kap
}
static void funcg(double u, double kappa, double gamma, double rho12, dcomplex f
{
  if (Cimag(fd) == 0.0 && Creal(fd) == 0.0)   *result = CONE;
  else    *result = Cdiv(Csub(RCsub(kappa, RCmul(gamma * rho12 * u, CI)), fd), C
}
```

```
static void funcC(double u, dcomplex fd, dcomplex fg, double tau, double kappa,
{
  *result = Cmul(Cdiv(RCsub(1., Cexp(RCmul(-tau, fd))), RCmul(gamma * gamma, RCs
}
static void funcC1(double kappa, double gamma, double *result)
{
  if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
  else *result = gamma * gamma * (1. - exp(-kappa)) / (4.*kappa);
}
static void funcLambda(double kappa, double v0, double gamma, double *result)
{
  if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
  else *result = 4.*kappa * v0 * exp(-kappa) / (gamma * gamma * (1. - exp(-kappa
}
static void funcD1(double kappa, double vbar, double gamma, double *result)
{
  if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
  else *result = 4.*kappa * vbar / (gamma * gamma);
}
static void fLambda1(double fC1, double fLambda, double fD1, double *result)
{
  if (fD1 == 0.0 || fLambda == 0.0) *result = 0.0;
  else *result = sqrt(fC1 * (fLambda - 1.) + fC1 * fD1 + fC1 * fD1 / (2.*(fD1 +
}
static void funcbeta1(double kappa, double vbar, double gamma, double *result)
{
  if (vbar - gamma * gamma / 8. / kappa < 0.0)
    {
      *result = 0.0;
      printf("Invalid parameter for vbar, gammav, kappa!\ n");
    }
  else if (kappa == 0.0 || gamma == 0.0) *result = 0.0;
  else *result = sqrt(vbar - gamma * gamma / 8. / kappa);
}
static void funcbeta2(double v0, double beta1, double *result)
{
  *result = sqrt(v0) - beta1;
}
static void funcbeta3(double beta1, double beta2, double Lambda1, double *result
{
  if (beta2 == 0.0 || (Lambda1 - beta1) == 0.0) *result = 0.0;
```

```
  else *result = -log((Lambda1 - beta1) / beta2);
}
static double intgf1r(double x, void *p)
{
  dcomplex fC;
  funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
  return (kappa0 * vbar0 + rho230 * gammav0 * etad0 * (beta10 + beta20 * exp(-be
}

static double intgf2r(double x, void *p)
{
  dcomplex fC;
  funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
  return rho230 * etad0 * gammav0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}


static double intgf3r(double x, void *p)
{
  dcomplex fC;
  funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
  return rho240 * gammav0 * etaf0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
static double intgf1i(double x, void *p)
{
  dcomplex fC;
  funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
  return (kappa0 * vbar0 + rho230 * gammav0 * etad0 * (beta10 + beta20 * exp(-be
}
static double intgf2i(double x, void *p)
{
  dcomplex fC;
  funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
  return rho230 * etad0 * gammav0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
static double intgf3i(double x, void *p)
{
  dcomplex fC;
  funcC(u0, fd0, fg0, x, kappa0, gammav0, rho120, &fC);
  return rho240 * gammav0 * etaf0 * (beta10 + beta20 * exp(-beta30 * x)) * (exp(
}
```

```
static double intgfzeta(double x, void *p)
{
  return (rho130 * etad0 * (exp(-lambdad0 * (T0 - x)) - 1.) / lambdad0 - rho140
}
static void funcA(double u, double v0, double kappa, double vbar, double gammav,
{
  double intg1r, intg2r, intg3r, intg1i, intg2i, intg3i, intgzeta;
  double fC1, fLambda, fD1, Lambda1, beta1, beta2, beta3;
  dcomplex fd, fg;
  PnlFunc func1r, func2r, func3r, func1i, func2i, func3i, funczeta;
  int n;

  n = 50;
  intg1r = 0.;
  intg2r = 0.;
  intg3r = 0.;
  intg1i = 0.;
  intg2i = 0.;
  intg3i = 0.;
  intgzeta = 0.;

  funcd(u, kappa, gammav, rho12, &fd);
  funcg(u, kappa, gammav, rho12, fd, &fg);
  funcbeta1(kappa, vbar, gammav, &beta1);
  funcbeta2(v0, beta1, &beta2);
  funcC1(kappa, gammav, &fC1);
  funcLambda(kappa, v0, gammav, &fLambda);
  funcD1(kappa, vbar, gammav, &fD1);
  fLambda1(fC1, fLambda, fD1, &Lambda1);
  funcbeta3(beta1, beta2, Lambda1, &beta3);

  fd0 = fd;
  fg0 = fg;
  u0 = u;
  kappa0 = kappa;
  vbar0 = vbar;
  rho120 = rho12;
  rho130 = rho13;
  rho140 = rho14;
  rho340 = rho34;
  gammav0 = gammav;
```

```
  rho230 = rho23;
  etad0 = etad;
  lambdad0 = lambdad;
  rho240 = rho24;
  etaf0 = etaf;
  lambdaf0 = lambdaf;
  beta10 = beta1;
  beta20 = beta2;
  beta30 = beta3;
  T0 = T;

  func1r.F = intgf1r;
  func2r.F = intgf2r;
  func3r.F = intgf3r;
  func1i.F = intgf1i;
  func2i.F = intgf2i;
  func3i.F = intgf3i;
  funczeta.F = intgfzeta;
  intg1r = pnl_integration(&func1r, 0.0, T, n, "simpson");
  intg2r = pnl_integration(&func2r, 0.0, T, n, "simpson");
  intg3r = pnl_integration(&func3r, 0.0, T, n, "simpson");
  intg1i = pnl_integration(&func1i, 0.0, T, n, "simpson");
  intg2i = pnl_integration(&func2i, 0.0, T, n, "simpson");
  intg3i = pnl_integration(&func3i, 0.0, T, n, "simpson");
  intgzeta = pnl_integration(&funczeta, 0.0, T, n, "simpson");
  *result = Cadd(RCadd(intg1r + u * intg2i - u * intg3i, RCmul(intg1i - u * intg
}
static void chf(double u, double v0, double kappa, double vbar, double gammav, d
{
  dcomplex fb, fd, fg, fc, fa, expon;
  funcB(u, &fb);
  funcd(u, kappa, gammav, rho12, &fd);
  funcg(u, kappa, gammav, rho12, fd, &fg);
  funcC(u, fd, fg, T, kappa, gammav, rho12, &fc);
  funcA(u, v0, kappa, vbar, gammav, rho12, rho13, rho14, rho23, rho24, rho34, et
  expon = Cadd(Cadd(fa, RCmul(v0, fc)) , RCmul(log(csi0 * exp(-pf0 * T + pd0 * T
  *result = Cexp(expon);
}

//COSINE method to calculate inverse fourier integral
static double cosine_function_xi(double d, double c, double dma, double cma, dou
```

```
{
  double res;
  res = 1. / (1 + fnpi * fnpi) * ((cos(dma) + fnpi * sin(dma)) * exp(d) - (cos(c
  return res;
}
static double cosine_function_psi(double d, double c, double dma, double cma, do
{
  double res;
  res = (fnpi == 0.) ? (d - c) : (sin(dma) - sin(cma)) / fnpi;
  return res;
}
static double cosine_Vk(double K, double a, double b, double fnpi)
{
  double cma, bma, result;
  cma = -a * fnpi;
  bma = (b - a) * fnpi;
  result = 2. / (b - a) * K * (cosine_function_xi(b, 0, bma, cma, fnpi) - cosine
  return result;
}


static void cosine_bound(double L, double csi0, double strike, double pd0, doubl
{
  double c1, zeta, fC1, fLambda, Lambda1, fD1, beta1, beta2, beta3;
  funcbeta1(kappa, vbar, gammav, &beta1);
  funcbeta2(v0, beta1, &beta2);
  funcC1(kappa, gammav, &fC1);
  funcLambda(kappa, v0, gammav, &fLambda);
  funcD1(kappa, vbar, gammav, &fD1);
  fLambda1(fC1, fLambda, fD1, &Lambda1);
  funcbeta3(beta1, beta2, Lambda1, &beta3);

  zeta = etad * etad * log(exp(-lambdad * T)) * pow(lambdad, -0.3e1) / 0.2e1 + r

  c1 = csi0 / strike * exp(-pf0 * T) / exp(-pd0 * T) + v0 / 2. / kappa * (exp(-k

  *a = c1 - L;
  *b = c1 + L;
}

static void HHW4d(double v0, double kappa, double vbar, double gammav, double pd
```

```
{
  double sum, a, b, invbma, fnpi, K;
  dcomplex phi = CZERO;
  int i;
  double L;

  K = strike;
  L = 10; //Integration Domain
  invbma = 0.0;
  fnpi = 0.0;
  sum = 0.;
  i = 0;
  cosine_bound(L, csi0, strike, pd0, pf0, v0, kappa, vbar,  gammav, etad,  lambd
  invbma = M_PI / (b - a);
  chf(fnpi, v0, kappa, vbar, gammav, pd0, etad, lambdad, pf0, etaf, lambdaf, rho
  sum = 0.5 * phi.r * cosine_Vk(K, a, b, fnpi);
  for (i = 1; i < N; i++)
    {
      fnpi += invbma;
      chf(fnpi, v0, kappa, vbar, gammav, pd0, etad, lambdad, pf0, etaf, lambdaf,
      sum += (phi.r * cos(fnpi * (-a)) - phi.i * sin(fnpi * (-a))) * cosine_Vk(K
    }
  *Price = sum * exp(-pd0 * T);
}

/*Compute Price Option*/

int  AP_FourierCosin_HesVas2d(double s0, NumFunc_1  *p, double T, PnlVect *vrq,
{
  double v0, kappav, thetav, sigmav, r0, kappar, sigmar, q0, kappaq, sigmaq, rho

  strike = p->Par[0].Val.V_PDOUBLE;

  //parameters of the model
  v0 = GET(vrq, 0);
  r0 = GET(vrq, 1);
  q0 = GET(vrq, 2);

  kappav = GET(kappa, 0);
  kappar = GET(kappa, 1);
  kappaq = GET(kappa, 2);
```

```
  thetav = GET(theta, 0);

  sigmav = GET(sigma, 0);
  sigmar = GET(sigma, 1);
  sigmaq = GET(sigma, 2);

  rhoSv = GET(rho, 0);
  rhoSr = GET(rho, 1);
  rhoSq = GET(rho, 2);
  rhovr = GET(rho, 3);
  rhovq = GET(rho, 4);
  rhorq = GET(rho, 5);

  HHW4d(v0, kappav, thetav, sigmav, r0, kappar, sigmar, q0, kappaq, sigmaq, rhoS

  //Price
  *ptprice = Price;

  return OK;
}

int CALC(AP_FOURIERCOSINE_HESVAS2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;

  return AP_FourierCosin_HesVas2d(ptMod->S0.Val.V_PDOUBLE,
                                  ptOpt->PayOff.Val.V_NUMFUNC_1,
                                  ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DA
                                  ptMod->vrq.Val.V_PNLVECT,
                                  ptMod->kappa.Val.V_PNLVECT,
                                  ptMod->theta.Val.V_PNLVECT,
                                  ptMod->sigma.Val.V_PNLVECT,
                                  ptMod->rho.Val.V_PNLVECT,
                                  Met->Par[0].Val.V_PINT,
                                  &(Met->Res[0].Val.V_DOUBLE));
}

static int CHK_OPT(AP_FOURIERCOSINE_HESVAS2D)(void *Opt, void *Mod)
{
```

```
  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0))
    return OK;
  return WRONG;
}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->HelpFilenameHint = "ap_fouriercosine_hesvas2d";
      Met->Par[0].Val.V_INT = 128;
    }

  return OK;
}

PricingMethod MET(AP_FOURIERCOSINE_HESVAS2D) =
{
  "AP_FOURIERCOSINE_HESVAS2D",
  { {"Number of integration steps (power of 2)", INT2, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(AP_FOURIERCOSINE_HESVAS2D),
  { {"Price", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(AP_FOURIERCOSINE_HESVAS2D),
  CHK_ok,
  MET(Init)
};
```