Help

```
extern "C" {

#include "
href../../../mod/doublehes1d/doublehes1d_vol/doublehes1d_vol_h_src.pdfhes1d_vol.
#include "
href../../../common/numfunc_h_src.pdfnumfunc.h"
#include "pnl/pnl_random.h"
#include "pnl/pnl_vector.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_cdf.h"
#include "pnl/pnl_mathtools.h"
}
#include <cmath>
#include <
href../../../common/math/highdim_solver/highdim_vector_h_src.pdfvector>

extern "C" {

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2017+2) //The "#els
  static int CHK_OPT(MC_HES_VARIANCESWAP)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(MC_HES_VARIANCESWAP)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else

static void optimalIntegrationDirectionHullWhite(double V_0,const double sigma,c
const double delta = T/(double)N;
for(int i=1;i<=N;i++){
double sigm = sigma*sqrt(theta + (V_0-theta)*exp(-kappa*delta*i));
double constante = V_0*sqrt(1.0-rho*rho)*sigm*sqrt(delta)/(exp(delta*(-kappa-0.5
LET(v,i-1)= constante*( exp(delta*N*(-kappa-0.5*sigm*sigm)) - exp(delta*i*(-kapp
}
}
```

```
static void generatePathsHeston(const double sigma,const double r,const double k
const double delta = T/N;
LET(S,0) = S_0;
LET(V,0) = V_0;
for(int i=1;i<=N;i++){
LET(S,i) = GET(S,i-1)*exp(delta*(r-divid-0.5*GET(V,i-1))+sqrt(GET(V,i-1)*delta)*
LET(V,i) = MAX(GET(V,i-1) - kappa*delta*MAX(GET(V,i-1),0.0) + delta*kappa*theta
}
}

static double discreteRealizedVariance(PnlVect *S,int N){
double sum = 0.;
for(int i=0;i<N;i++)
sum += pow(log(GET(S,i+1)/GET(S,i)),2);
return sum;
}

static double density_xi2(const double u,int k){
return pow(u,0.5*k - 1.)*exp(-0.5*u)*pow(0.5,0.5*k)/pnl_sf_gamma(0.5*k);//tgamma
}

static void generateConditionalZ(int N,PnlVect *Z,double pi1,PnlRng *rng){
pnl_vect_rng_normal(Z, N, rng);
double norme2xi = pnl_vect_norm_two(Z);// compute pi_1: ||Z||_2
double constante = pi1/norme2xi;
pnl_vect_mult_scalar(Z,constante);
}

static void generateConditionalW(int N,PnlVect *W,PnlVect *v,double pi2,PnlRng *
for(int i=0;i<N;i++)
LET(W,i) = GET(v,i)*pi2 + (1.-GET(v,i)*GET(v,i))*pnl_rng_normal(rng);
}

  static double conditionalMonteCarlo(double K,const double sigma,const double r
double sum=0.;
if(n==0) return 0;
for(int i=0;i<n;i++){
generateConditionalZ(N,Z,pi1,rng);
generateConditionalW(N,W,v,pi2,rng);
generatePathsHeston(sigma,r,kappa,rho,T,N,S_0,V_0,Z,W,S,V,theta,divid);
sum += (discreteRealizedVariance(S,N) - K*K/10000.0);//payoff(RV)
```

```
}
sum /= M;
return sum;
}

 /*////////////////////////////////////////////////////*/
  static int mc_hes_varswap(double V_0, double kappa, double theta, double sigma
  {
    pnl_rand_init(generator, 1, N);

PnlRng *rng = pnl_rng_create(generator);
pnl_rng_sseed(rng, 0);

double a=-4.0,b=4.0,c=0.0,d=15.*sqrt(N);
int pi_1_steps = 30;// pi_1 = khi^2 of degree N
int pi_2_steps = 40;// pi_2 = standard normal distribution
double dx = (b-a)/(double)(pi_2_steps-1.);
double dy = (d-c)/(double)(pi_1_steps-1.);
double x,y;

double price = 0;
int n;

PnlVect *v = pnl_vect_create_from_zero(N);// v = 0
PnlVect *W = pnl_vect_new();// W = 0
PnlVect *Z = pnl_vect_new();// Z = 0
PnlVect *V = pnl_vect_create_from_zero(N+1);// V = 0
PnlVect *S = pnl_vect_create_from_zero(N+1);// S = 0

optimalIntegrationDirectionHullWhite(V_0,sigma,kappa,theta,T,N,rho,v);// computa

for(int i=0;i<=pi_2_steps;i++){// integration over pi_2
x = a + i*dx;
pnl_vect_rng_normal(W, N, rng);
double pi2 = pnl_vect_scalar_prod(v,W);// compute pi_2: <v,W>
double density_pi_2 = pnl_normal_density(x);// computaion of the density of pi_2
for(int j=0;j<=pi_1_steps;j++){// integration over pi_1
y = c + j*dy;
pnl_vect_rng_normal(Z, N, rng);
double pi1 = pnl_vect_norm_two(Z);// compute pi_1: ||Z||_2
double density_pi_1 = density_xi2(y,N);// computaion of the density of pi_1 at y
```

```
n = floor(M*density_pi_1*density_pi_2);
price += conditionalMonteCarlo(K,sigma,r,kappa,rho,T,N,S_0,V_0,Z,W,S,V,v,n,M,pi1
}
}

*Price=exp(-r*T)*dx*dy*price*10000.0;

pnl_vect_free (&W);
pnl_vect_free (&Z);
pnl_vect_free (&V);
pnl_vect_free (&v);
pnl_vect_free (&S);
pnl_rng_free (&rng);

return OK;

  }


  int CALC(MC_HES_VARIANCESWAP)(void *Opt, void *Mod, PricingMethod *Met)
  {
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;
    double r, divid, strike, spot;
    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;
    spot = ptMod->S0.Val.V_DOUBLE;

    return mc_hes_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, spot,
```

```
   Met->Par[0].Val.V_LONG,
                Met->Par[1].Val.V_INT,
                Met->Par[2].Val.V_ENUM.value,
             &(Met->Res[0].Val.V_DOUBLE)/*PRICE*/);

  }

  static int CHK_OPT(MC_HES_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->init = 1;
      Met->HelpFilenameHint = "mc_hes_varianceswap";

      Met->Par[0].Val.V_LONG = 15000;
      Met->Par[1].Val.V_INT = 20;
      Met->Par[2].Val.V_ENUM.value = 0;
      Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
    }

    return OK;
  }

  PricingMethod MET(MC_HES_VARIANCESWAP) =
  {
    "MC_HES_VARIANCESWAP",
    {   {"N iterations", LONG, {100}, ALLOW},
    {"TimeStepNumber", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
{" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(MC_HES_VARIANCESWAP),
    {
```

```
      {"Price in 10000 variance points", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_HES_VARIANCESWAP),
    CHK_ok ,
    MET(Init)
  } ;

  /*////////////////////////////////////*/
}
```