Help

```
extern "C" {
#include  "
href../../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "
href../../../common/enums_h_src.pdfenums.h"
#include "
href../../../common/math/clustering_h_src.pdfmath/clustering.h"
}

#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_random.h"
#include <cstdlib>

#include <cmath>
#include <
href../../../common/math/highdim_solver/highdim_vector_h_src.pdfvector>
using namespace std;


#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2018+2) //The "#els
extern "C" {
  static int CHK_OPT(MC_Giles_Heston)(void *Opt, void *Mod)
  {
return NONACTIVE;
  }
  int CALC(MC_Giles_Heston)(void *Opt, void *Mod, PricingMethod *Met)
  {
return AVAILABLE_IN_FULL_PREMIA;
  }
}
#else

int MCGiles(double s0, NumFunc_1  *p, double T, double strike, double r, double
{
  int call_or_put;
  double alpha = 1, beta = 2, c1 = 1, c2 = 1;
  double m=2;//pow(m,l) is the number of time steps
  //double eps = 1/double(n);
  double price_first = 0, price_second = 0;
```

```
  double L =0;
  PnlRng *rng;

  if ((p->Compute) == &Call)
    call_or_put = 1;
  else
    call_or_put = 0;

  rng = pnl_rng_create(generator);
  pnl_rng_sseed(rng, 0);
  L = floor(log(sqrt(2)*c1*pow(T,alpha)*pow(eps,-1))/(alpha*log(m)));

  //---------------------------------
  // Calculation of the first party
  //---------------------------------
  double h0 = T/pow(m,0);
  double N0 = floor(2*pow(eps,-2)*c2*pow(T,(beta-1)/2)*pow((1-pow(m,(1-beta)/2))
  double rho1=1-rho*rho;
  for (double i = 0;i<N0;i++){
int n0 = (int)pow(m,0);
double log_s_t = log(s0);
double v_t = v0;
for (int j =0;j<n0;j++){
  double G1 =pnl_rand_normal(generator);
  double G2 =pnl_rand_normal(generator);

  log_s_t = log_s_t + (r-divid-1/2.*v_t)*(T/(double)n0)+ sqrt(v_t)*sqrt(T/(doubl
  v_t = fabs(v_t+ka*(theta-v_t)*(T/(double)n0)+sigma*sqrt(v_t)*sqrt(T/(double)n0

}

if(call_or_put)
price_first = price_first + MAX(exp(log_s_t)-strike,0);
else
  price_first = price_first + MAX(strike-exp(log_s_t),0);

  }
  price_first = price_first/N0*exp(-r*T);


  //-----------------------------------------------
```

```
  // Calculation of the second party - antithetic
  //-----------------------------------------------
  for (double l =0;l<L+1;l++){
double nl = pow(m,l);
double hl = T/(double)nl;
double Nl = floor(2.*pow(eps,-2)*c2*pow(T,(beta-1)/2.)*pow((1-pow(m,-(beta-1)/2.

double esti_second = 0.;

for (double j=0;j<Nl;j++){
  double v_t_f = v0;
  double v_t_c = v0;
  double v_t_a = v0;
  double v_t_f_harf = v0;
  double v_t_a_harf = v0;

  double log_s_t_f = log(s0);
  double log_s_t_c = log(s0);
  double log_s_t_a = log(s0);
  double log_s_t_f_harf = log(s0);
  double log_s_t_a_harf = log(s0);

  vector<double> G1(nl+1,0);
  vector<double> G2(nl+1,0);
  vector<double> G3(nl+1,0);
  vector<double> G4(nl+1,0);

  //-------------------------------------------------------------
  //Fine path simulation and random variable generator
  //-------------------------------------------------------------
  for(double k = 0;k<nl;k++){
G1[k+1] = G1[k] +pnl_rand_normal(generator)*sqrt(T/(double)(nl*2.));
G2[k+1] = G2[k] +pnl_rand_normal(generator)*sqrt(T/(double)(nl*2.));
G3[k+1] = G3[k] +pnl_rand_normal(generator)*sqrt(T/(double)(nl*2.));
G4[k+1] = G4[k] +pnl_rand_normal(generator)*sqrt(T/(double)(nl*2.));

log_s_t_f_harf = log_s_t_f + (r-divid-1/2.*v_t_f)*(T/nl)/2. + sqrt(v_t_f)*(sqrt(
v_t_f_harf = fabs(v_t_f + ka*(theta-v_t_f)*(T/nl)/2. + sigma * sqrt(v_t_f)*(G2[k

log_s_t_f = log_s_t_f_harf + (r-divid-1/2.*v_t_f_harf)*(T/nl)/2. + sqrt(v_t_f_ha
v_t_f = fabs(v_t_f_harf + ka * (theta-v_t_f_harf)*(T/nl)/2. + sigma*sqrt(v_t_f_h
```

```
  }

  //-----------------------------------------------------------
  //Antithetic path simulation and random variable generator
  //-----------------------------------------------------------
  for(double l =0;l<nl;l++){
log_s_t_a_harf = log_s_t_a + (r-divid-1/2.*v_t_a)*(T/nl)/2. + sqrt(v_t_a)*(sqrt(
v_t_a_harf = fabs(v_t_a + ka*(theta-v_t_a)*(T/nl)/2. + sigma*sqrt(v_t_a)*(G4[l+1

log_s_t_a = log_s_t_a_harf + (r-divid-1/2.*v_t_a_harf)*(T/nl)/2. + sqrt(v_t_a_ha
v_t_a = fabs(v_t_a_harf + ka * (theta-v_t_a_harf)*(T/nl)/2. + sigma * sqrt(v_t_a

  }

  //-----------------------------------------------------------
  //Coarse path simulation and random variable generator
  //-----------------------------------------------------------
  for(double p=0;p<nl;p++){
log_s_t_c = log_s_t_c +(r-divid-1/2.*v_t_c)*(T/nl) + sqrt(v_t_c)*(sqrt(rho1)*((G
v_t_c = fabs(v_t_c + ka*(theta-v_t_c)*(T/nl) + sigma*sqrt(v_t_c)*((G2[p+1]-G2[p]
  }
   if(call_or_put)
  esti_second += (((MAX(exp(log_s_t_f)-strike,0)+MAX(exp(log_s_t_a)-strike,0))/2
else
  esti_second += (((MAX(strike-exp(log_s_t_f),0)+MAX(strike-exp(log_s_t_a),0))/2
}
esti_second = esti_second/Nl*exp(-r*T);
price_second = price_second +esti_second;
  }

  *ptprice = price_first+price_second;

  return OK;

}

extern "C" {
  int CALC(MC_Giles_Heston)(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 MCGiles(ptMod->S0.Val.V_PDOUBLE,
                   ptOpt->PayOff.Val.V_NUMFUNC_1,
                   ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
   ptOpt->PayOff.Val.V_NUMFUNC_1->Par[0].Val.V_PDOUBLE,
                   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_PDOUBLE,Met->Par[1].Val.V_ENUM.v
   &(Met->Res[0].Val.V_DOUBLE)
   );

  }



  static int CHK_OPT(MC_Giles_Heston)(void *Opt, void *Mod)
  {

if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt)-
  return OK;

return  WRONG;
  }
}
#endif //PremiaCurrentVersion

extern "C" {
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
//int type_generator;
if (Met->init == 0)
  {
Met->init = 1;
```

```
Met->HelpFilenameHint = "MC_Giles_Heston";
Met->Par[0].Val.V_PDOUBLE = 0.01;
Met->Par[1].Val.V_ENUM.value = 0;
Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
  }


return OK;
  }


  PricingMethod MET(MC_Giles_Heston) =
  {
"MC_Giles_Heston",
{ { "Accuracy of the simulation", DOUBLE, {100}, ALLOW},
  {"Random Generator", ENUM, {0}, ALLOW},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CALC(MC_Giles_Heston),
{ {"Price", DOUBLE, {100}, FORBID},
  {" ", PREMIA_NULLTYPE, {0}, FORBID}
},
CHK_OPT(MC_Giles_Heston),
CHK_mc,
MET(Init)
  };
}
```