Help

```
#include  "hes1d_std.h"
#include "enums.h"
#include "math/ESM_func.h"
#include "pnl/pnl_random.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2009+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_BroadieKaya_Heston)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(MC_BroadieKaya_Heston)(void*Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

int MCBroadieKaya(double S0, NumFunc_1  *pf, double T,
    double r, double divid, double v0,double K_heston,double Theta,
    double sigma,double rho, long N_sample,int N_t_grid,int     generator,  doub
    double *ptdelta, double *pterror_price, double *pterror_delta ,
    double *inf_price, double *sup_price, double *inf_delta, double
    *sup_delta)
{
  int i
  ;
  long k;
  double g1,g2;
  double price_sample, delta_sample, mean_price, mean_delt
    a, var_price, var_delta;
  double alpha, z_alpha;
  double u;
  double d, ekd, nekd, C0,B;
  double sq_rho, KTD,RS,KRS;

  double Vi;
  double V,log_S;
double lambda;
double gen;
int pois, N;
double Vst, mean, variance, h;
double *val;
double delta = T/N_t_grid;
double erT=exp((r-divid)*T);
int M;



delta = T/N_t_grid;
erT=exp((r-divid)*T);
M=10000;
val = malloc (sizeof(double) * M);

//Useful constant
d=4*K_heston*Theta/(sigma*sigma);
ekd=exp(-K_heston*delta);
nekd= 1.- ekd;
C0=pow(sigma,2.)*nekd/(4*K_heston);
B=ekd/C0;
sq_rho=sqrt(1-rho*rho);
KTD=K_heston*Theta*delta;
RS=rho/sigma;
KRS=K_heston*RS-0.5;


ESM_update_const_char( K_heston, sigma, delta, d);

/* Value to construct the confidence interval */
alpha= (1.- confidence)/2.;
z_alpha= pnl_inv_cdfnor(1.- alpha);

/*Initialisation*/
mean_price= 0.0;
mean_delta= 0.0;
var_price= 0.0;
var_delta= 0.0;

pnl_rand_init(generator,1,N_sample);
```

```
for(k=0; k<N_sample; k++ )
 {
  // N_path Paths
  V=v0;
  log_S=log(S0);
  for(i=0; i<N_t_grid; i++)
   {
    u=pnl_rand_uni(generator);
    g2=pnl_rand_normal(generator);

    Vi=V;
    lambda=B*Vi;
    if(d>1){
     g1=pnl_rand_normal(generator);
     gen=pow(g1+sqrt(lambda),2.)+pnl_rand_chi2(d-1.,
     generator);
    }
    else{
     pois=pnl_rand_poisson(lambda*0.5,generator);
     gen=pnl_rand_chi2(d+2*pois, generator);
    }

    V=C0*gen;

    Moments_ESM( Vi, V, K_heston, sigma, delta, d, &
 mean, &variance);

    h=M_PI/(mean+5.*sqrt(variance));


    values_all_ESM(M,Vi, V, K_heston, sigma, delta,
 d, 1.e-6, h, &N, val);
    Vst= inverse_ESM( u, h, N, val);

    log_S += RS *(V - Vi - KTD) + KRS*Vst+sq_rho*sq
 rt(Vst)*g2;

   }

  /*Price*/
```

```
    price_sample=(pf->Compute)(pf->Par,erT*exp(log_S));

    /* Delta */
    if(price_sample >0.0)
      delta_sample=(erT*exp(log_S)/S0);
    else  delta_sample=0.;

    /* Sum */
    mean_price+= price_sample;
    mean_delta+= delta_sample;

    /* Sum of squares */
    var_price+= SQR(price_sample);
    var_delta+= SQR(delta_sample);

  }
/* End of the N iterations */

/* Price estimator */
*ptprice=(mean_price/(double)N_sample);
*pterror_price= exp(-r*T)*sqrt(var_price/(double)N_sampl
  e-SQR(*ptprice))/sqrt((double)N_sample-1);
*ptprice= exp(-r*T)*(*ptprice);

/* Price Confidence Interval */
*inf_price= *ptprice - z_alpha*(*pterror_price);
*sup_price= *ptprice + z_alpha*(*pterror_price);


/* Delta estimator */
*ptdelta=exp(-r*T)*(mean_delta/(double)N_sample);
if((pf->Compute) == &Put)
  *ptdelta *= (-1);
*pterror_delta= sqrt(exp(-2.0*r*T)*(var_delta/(double)N_
  sample-SQR(*ptdelta)))/sqrt((double)N_sample-1);

/* Delta Confidence Interval */
*inf_delta= *ptdelta - z_alpha*(*pterror_delta);
*sup_delta= *ptdelta + z_alpha*(*pterror_delta);

free(val);
```

```
  return OK;
}

int CALC(MC_BroadieKaya_Heston)(void *Opt, void *Mod, Prici
    ngMethod *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 MCBroadieKaya(ptMod->S0.Val.V_PDOUBLE,
                 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.hal.V_PDOUBLE,
                 ptMod->LongRunVariance.Val.V_PDOUBLE,
                 ptMod->Sigma.Val.V_PDOUBLE,
                 ptMod->Rho.Val.V_PDOUBLE,
                 Met->Par[0].Val.V_LONG,
                 Met->Par[1].Val.V_INT,
                 Met->Par[2].Val.V_ENUM.value,
                 Met->Par[3].Val.V_RGDOUBLE12,
                 Met->Par[4].Val.V_PDOUBLE,
                 &(Met->Res[0].Val.V_DOUBLE),
                 &(Met->Res[1].Val.V_DOUBLE),
                 &(Met->Res[2].Val.V_DOUBLE),
                 &(Met->Res[3].Val.V_DOUBLE),
                 &(Met->Res[4].Val.V_DOUBLE),
                 &(Met->Res[5].Val.V_DOUBLE),
                 &(Met->Res[6].Val.V_DOUBLE),
                 &(Met->Res[7].Val.V_DOUBLE));
  return OK;

}
static int CHK_OPT(MC_BroadieKaya_Heston)(void *Opt, void *
    Mod)
{

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

  return  WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  //int type_generator;
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_LONG=10000;
      Met->Par[1].Val.V_INT=1;
      Met->Par[2].Val.V_ENUM.value=0;
      Met->Par[2].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[3].Val.V_RGDOUBLE12= 1.5;
      Met->Par[4].Val.V_DOUBLE= 0.95;
    }

  return OK;
}

PricingMethod MET(MC_BroadieKaya_Heston)=
{
  "MC_BroadieKaya",
  {{"N iterations",LONG,{100},ALLOW},
   {"TimeStepNumber",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"THRESHOLD",DOUBLE,{100},ALLOW},
   {"Confidence Value",DOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_BroadieKaya_Heston),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {"Error Price",DOUBLE,{100},FORBID},
   {"Error Delta",DOUBLE,{100},FORBID} ,
   {"Inf Price",DOUBLE,{100},FORBID},
   {"Sup Price",DOUBLE,{100},FORBID} ,
   {"Inf Delta",DOUBLE,{100},FORBID},
   {"Sup Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_BroadieKaya_Heston),
  CHK_mc,
  MET(Init)
};
```

References