Help

```
#include <stdlib.h>
#include  "hes1d_std.h"
#include "enums.h"

static double *m_Mu;
/* --------------------------------------------------------
    ---------------- */
/* Calculus of the average A'(T0,T) and C'(T0,T) of the
    asian option with one of the 3 different schemes
   One iteration of the Monte Carlo method called from the
    "FixedAsian_KemanVorst" function */
/* --------------------------------------------------------
    ---------------- */
static double gamma_step(int n,double a,double b)
{
  return a/(b+(double)n);
}

static double step(int n){
  return sqrt(log((double)n+1.)/10.)+50.;
}

static void Simul_StockAndAverage_RobbinsMonro(int     generator, int step_numbe
    divid, double sigma0,double k,double theta,double sigma2,
    double rho, NumFunc_1 *p)
{
  int RM=5000;
  int sig_itere=0;
  double    S_t, g1, g2,K;
  double h = T / step_number;
  double sqrt_h = sqrt(h), sqrt_rho = sqrt(1.-SQR(rho));
  int i,ii;
  double dot1,a,b=1,payoff,payoffcarre,val_test,temp,expo,
    val;
  double dot2;
  double x_1=0.,x_2=0.,test;
  double V_t, value;
  double *NormalValue,*m_Theta;

  NormalValue = malloc(sizeof(double)*2*step_number*RM);
```

```
m_Theta= malloc(sizeof(double)*(2*(step_number+1)));
K=p->Par[0].Val.V_DOUBLE;
/* Average Computation */
/* Trapezoidal scheme */
/* Simulation of M gaussian variables according to the     generator type,
   that is Monte Carlo or Quasi Monte Carlo. */

if ((p->Compute) == &Call){
  x_1=0.0095;x_2=0.0075;
}
if ((p->Compute) == &Put){
  x_1=-0.095;x_2=-0.075;
}

for(i=0;i<2*step_number;i++)
  m_Mu[i]=0.0;
/*choosing the coefficient of the steps sequence*/
test=K/x;
if ((p->Compute) == &Call)
  {
    if(sigma0 <= 0.02){
      if(test>=1.1)
        a=0.005;
      else if(test<1.1 && 0.9<=test)
        a=0.001;
      else if(test<0.9)
        a=0.0005;
      else
        a=0.0005;
    }else {
      if(test>=1.2)
        a=0.005;
      else if(1.1<=test && test<1.2)
        a=0.00025;
      else if(1.0<=test && test<1.1)
        a=0.0005;
      else if(test<1.0)
        a=0.00025;
      else
        a=0.0005;
    }
```

```
    }
  else /*if ((p->Compute) == &Put)*/
    {
      if(sigma0 <= 0.02){
        if(test>=1.1)
          a=0.005;
        else if(test<1.1 && 0.9<=test)
          a=0.01;
        else if(test<0.9)
          a=0.05;
        else
          a=0.005;
      }else {
        if(test>=1.2)
          a=0.0005;
        else if(1.1<=test && test<1.2)
          a=0.0025;
        else if(1.0<=test && test<1.1)
          a=0.005;
        else if(test<1.0)
          a=0.0025;
        else
          a=0.0005;
      }
    }
  for(ii=0;ii<RM;ii++) {

    dot1=0.;
    dot2=0.;

    g1= pnl_rand_gauss(2*step_number, CREATE, 0,     generator);
    S_t=x;
    V_t=sigma0;
    for(i=0 ; i< step_number ; i++)
      {
        g1= pnl_rand_gauss(step_number, RETRIEVE, 2*i,     generator);
        NormalValue[i+ii*step_number]=g1;
        S_t*=(1+(r-divid)*h + sqrt(V_t)*sqrt_h*g1);

        g2= pnl_rand_gauss(step_number, RETRIEVE, (2*i)+1,     generator);
        NormalValue[i+(ii+RM)*step_number]=g2;
```

```
    dot1+=g1*m_Mu[i]+g2*m_Mu[i+step_number];
    dot2+=m_Mu[i]*m_Mu[i]+m_Mu[i+step_number]*m_Mu[i+
step_number];
    value=rho*g1+sqrt_rho*g2;
    V_t=V_t+k*(theta-V_t)*h+sigma2*sqrt_h*sqrt(V_t)*val
ue;
    V_t=(V_t<0.0?(-V_t) : V_t);
  }

payoff=exp(-r*T)*(p->Compute)(p->Par,S_t);
payoffcarre=payoff*payoff;
expo=exp(-dot1+0.5*dot2);
val_test=0.;

for(i=0 ; i< step_number ; i++)
  {
    val=NormalValue[i+ii*step_number];
    temp=(m_Mu[i]-val)*expo*payoffcarre;
    m_Theta[i]=temp;
    val=NormalValue[i+(ii+RM)*step_number];
    temp=(m_Mu[i+step_number]-val)*expo*payoffcarre;
    m_Theta[i+step_number]=temp;
    val_test+=SQR(m_Mu[i]-gamma_step(ii,a,b)*m_Theta[i]
)+SQR(m_Mu[i+step_number]-gamma_step(ii,a,b)*m_Theta[i+
step_number]);
  }
val_test=sqrt(val_test);
if(val_test<=step(sig_itere)) {
  for(i=0;i<step_number;i++) {
    m_Mu[i]=m_Mu[i]-gamma_step(ii,a,b)*m_Theta[i];
    m_Mu[i+step_number]=m_Mu[i+step_number]-gamma_step(
ii,a,b)*m_Theta[i+step_number];
  }
}
else {
  if(sig_itere-2*(sig_itere/2)==0)
    for(i=0;i<step_number;i++){
      m_Mu[i]=x_1;
      m_Mu[i+step_number]=x_1;
    }
  else
```

```
        for(i=0;i<step_number;i++){
          m_Mu[i]=x_2;
          m_Mu[i+step_number]=x_2;
        }
      sig_itere+=1;
    }
  }
  free(m_Theta);
  free(NormalValue);

  return;
}

static int MCRobbinsMonro(double s, NumFunc_1  *p, double
    t, double r, double divid, double sigma0,double k,double th
    eta,double sigma2,double rho, long nb, int M,int     generator, double confi
    double *pterror_price, double *pterror_delta , double *inf_
    price, double *sup_price, double *inf_delta, double *sup_delta)
{
  long i,ipath;
  double price_sample, delta_sample, mean_price, mean_delt
    a, var_price, var_delta;
  int init_mc;
  int simulation_dim;
  double alpha, z_alpha,dot1,dot2; /* inc=0.001;*/
  double  S_t, g1,g2;
  double h = t /(double)M;
  double sqrt_h = sqrt(h), sqrt_rho = sqrt(1.-SQR(rho));
  int step_number=M;
  double V_t, value;

  m_Mu= malloc(sizeof(double)*50000);

  /* 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;

/* Size of the random vector we need in the simulation */
simulation_dim= M;

/* MC sampling */
init_mc= pnl_rand_init(generator, simulation_dim,nb);
/* Test after initialization for the generator */
if(init_mc == OK)
  {


    /* Price  */
    (void)Simul_StockAndAverage_RobbinsMonro(generator,
  M, t, s,r, divid, sigma0,k,theta,sigma2,rho, p);

    dot2=0.;
    for(i=0;i<step_number;i++)
      dot2+=m_Mu[i]*m_Mu[i]+m_Mu[i+step_number]*m_Mu[i+
  step_number];

    for(ipath= 1;ipath<= nb;ipath++)
      {
        /* Begin of the N iterations */
        g1= pnl_rand_gauss(2*step_number, CREATE, 0,     generator);
        S_t=s;dot1=0.;
        V_t=sigma0;
        for(i=0 ; i<step_number ; i++)
          {
            g1= pnl_rand_gauss(step_number, RETRIEVE, 2*
  i, generator);
            S_t*=(1+(r-divid)*h + sqrt(V_t)*sqrt_h*(g1+m_
  Mu[i]));
            g2= pnl_rand_gauss(step_number, RETRIEVE, (2*
  i)+1, generator);
            dot1+=m_Mu[i]*g1+m_Mu[i+step_number]*g2;
            value=rho*(g1+m_Mu[i])+sqrt_rho*(g2+m_Mu[i+
  step_number]);
            V_t=V_t+k*(theta-V_t)*h+sigma2*sqrt_h*sqrt(V_
  t)*value;
```

```
        V_t=(V_t<0.0?(-V_t) : V_t);
      }
    price_sample=(p->Compute)(p->Par, S_t)*exp(-dot1-
0.5*dot2);

    /* Delta */
    if(price_sample >0.0)
      delta_sample=(S_t/s)*exp(-dot1-0.5*dot2);
    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)nb);
  *pterror_price= exp(-r*t)*sqrt(var_price/(double)nb-
SQR(*ptprice))/sqrt((double)nb-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)nb);
  if((p->Compute) == &Put)
    *ptdelta *= (-1);
  *pterror_delta= sqrt(exp(-2.0*r*t)*(var_delta/(
double)nb-SQR(*ptdelta)))/sqrt((double)nb-1);

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

```
    }
  free(m_Mu);
  return init_mc;
}


int CALC(MC_RobbinsMonro_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 MCRobbinsMonro(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.Val.V_PDOUB
    LE,
                        ptMod->LongRunVariance.Val.V_PDOUB
    LE,
                        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_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));
```

```
}



static int CHK_OPT(MC_RobbinsMonro_Heston)(void *Opt, void
    *Mod)
{
  /*Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);
    if ((opt->EuOrAm).Val.V_BOOL==EURO)
    return OK;*/

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

  return  WRONG;
}



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

      Met->Par[0].Val.V_LONG=15000;
      Met->Par[1].Val.V_INT=100;
      Met->Par[2].Val.V_ENUM.value=0;
      Met->Par[2].Val.V_ENUM.members=&PremiaEnumRNGs;
      Met->Par[3].Val.V_DOUBLE= 0.95;


    }



  type_generator=  Met->Par[2].Val.V_ENUM.value;
```

```
  if(pnl_rand_or_quasi(type_generator)==PNL_QMC)
    {
      Met->Res[2].Viter=IRRELEVANT;
      Met->Res[3].Viter=IRRELEVANT;
      Met->Res[4].Viter=IRRELEVANT;
      Met->Res[5].Viter=IRRELEVANT;
      Met->Res[6].Viter=IRRELEVANT;
      Met->Res[7].Viter=IRRELEVANT;

    }
  else
    {
      Met->Res[2].Viter=ALLOW;
      Met->Res[3].Viter=ALLOW;
      Met->Res[4].Viter=ALLOW;
      Met->Res[5].Viter=ALLOW;
      Met->Res[6].Viter=ALLOW;
      Met->Res[7].Viter=ALLOW;
    }
  return OK;
}


PricingMethod MET(MC_RobbinsMonro_Heston)=
{
  "MC_RobbinsMoro_hes",
  {{"N iterations",LONG,{100},ALLOW},
   {"TimeStepNumber",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Confidence Value",DOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_RobbinsMonro_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_RobbinsMonro_Heston),
  CHK_mc,
  MET(Init)
};
```

References