Help

```
/* Control Variables Kemna & Vorst Monte Carlo simulation
    for a Call or Put Fixed Asian option.
   In the case of Monte Carlo simulation, the program prov
    ides estimations for price and delta with a confidence
    interval.
   In the case of Quasi-Monte Carlo simulation, the program
     just provides estimations for price and delta. */
#include <stdlib.h>
#include  "bs1d_pad.h"
#include "enums.h"

static double m_Mu[50000];

/* ------------------------------------------------------------
    ---------------- */
/* 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.)/6.)+1.;
}

static void Simul_StockAndAverage_RobbinsMonro(int     generator, int step_numbe
    divid, double sigma, NumFunc_2 *p, double K)
{
  int RM=5000;
  int sig_itere=0;
  double integral,  S_t, g1;
  double h = T / step_number;
  double sqrt_h = sqrt(h);
  double trend= (r -divid)- 0.5 * SQR(sigma);
```

```
int i,ii;
double dot1,a,b=1,payoff,payoffcarre,val_test,temp,expo,
  val;
double dot2;
double* NormalValue;
double *m_Theta;
double x_1=-0.0925,x_2=-0.00725;
NormalValue = malloc(sizeof(double)*step_number*RM);
m_Theta= malloc(sizeof(double)*(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. */

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

if ((p->Compute) == &Call_OverSpot2)
  {
    if(K==x)
a=0.1;
    else if(K<x)
a=0.001;
    else /*if(K>x)*/
a=5.;
  }
else /*if ((p->Compute) == &Put_OverSpot2)*/
  {
    if(K==x)
a=0.1;
    else if(K<x)
a=5.;
    else /*if(K>x)*/
a=0.001;
  }
for(ii=0;ii<RM;ii++) {

  dot1=0.;
  dot2=0.;
```

```
  integral=0.;

  g1= pnl_rand_gauss(step_number, CREATE, 0, generator);
  S_t=x;
  integral=x*(1.+(r-divid)*h/2.+sigma*sqrt_h*g1/2.);
  for(i=0 ; i< step_number-1 ; i++) {
    NormalValue[i+ii*step_number]=g1;
    S_t*=exp(trend *h + sigma*sqrt_h*g1);
    dot1+=g1*m_Mu[i];
    dot2+=m_Mu[i]*m_Mu[i];
    g1= pnl_rand_gauss(step_number, RETRIEVE, i,     generator);
    integral+= S_t*(1.+(r-divid)*h/2.+sigma*sqrt_h*g1/2.)
  ;
  }

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

  for(i=0 ; i< step_number-1 ; i++) {
    val=NormalValue[i+ii*step_number];
    temp=(m_Mu[i]-val)*expo*payoffcarre;
    m_Theta[i]=temp;
    val_test+=SQR(m_Mu[i]-gamma_step(ii,a,b)*temp);
  }
  val_test=sqrt(val_test);
  if(val_test<=step(sig_itere)) {
    for(i=0;i<step_number-1;i++) {
m_Mu[i]=m_Mu[i]-gamma_step(ii,a,b)*m_Theta[i];
    }
  }
  else {
    if(sig_itere-2*(sig_itere/2)==0)
for(i=0;i<step_number-1;i++)
  m_Mu[i]=x_1;
    else
for(i=0;i<step_number-1;i++)
  m_Mu[i]=x_2;
```

```
      sig_itere+=1;
    }
  }


  free(m_Theta);
  free(NormalValue);

  return;
}

/* --------------------------------------------------------
    ----------*/
/* Pricing of a asian option by the Monte Carlo Kemna & Vor
    st method
   Estimator of the price and the delta.
   s et K are pseudo-spot and pseudo-strike. */
/* --------------------------------------------------------
    --------- */
static int  FixedAsian_RobbinsMonro(double s, double K,
    double time_spent, NumFunc_2 *p, double t, double r, double div
    id, double sigma, long nb, int M, int generator, double
    confidence, double *ptprice,double *ptdelta, double *pt
    error_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_de
    lta, var_price, var_delta;
  int init_mc;
  int simulation_dim;
  double alpha, z_alpha,dot1,dot2; /* inc=0.001; */
  double *Normalvect;
  double integral, S_t, g1;
  double h = t /(double)M;
  double sqrt_h = sqrt(h);
  double trend= (r -divid)- 0.5 * SQR(sigma);
  int step_number=M;
```

```
Normalvect= malloc(sizeof(double)*(nb*step_number+1));

/* 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, sigma, p, K);

    dot2=0;
    for(i=0;i<step_number-1;i++)
dot2+=m_Mu[i]*m_Mu[i];

    for(ipath= 1;ipath<= nb;ipath++)
{
  /* Begin of the N iterations */

  g1= pnl_rand_gauss(step_number, CREATE, 0, generator);
  integral=s*(1.+(r-divid)*h/2.+sigma*sqrt_h*g1/2.);
  S_t=s;
  for(i=0 ; i< step_number-1 ; i++) {
    Normalvect[i+(ipath-1)*step_number]=g1;
    S_t *=exp(trend *h +sigma*sqrt_h*(g1+m_Mu[i]));
```

```
    g1= pnl_rand_gauss(step_number, RETRIEVE, i,     generator);
    integral+=S_t*(1.+(r-divid)*h/2.+sigma*sqrt_h*(g1+m_
  Mu[i])/2.);
  }
        dot1=0.;
  for(i=0;i<step_number-1;i++)
    dot1+=m_Mu[i]*Normalvect[i+(ipath-1)*step_number];

        price_sample=(p->Compute)(p->Par, s,integral/(
  double)step_number)*exp(-dot1-0.5*dot2);

  /* Delta */
  if(price_sample >0.0)
    delta_sample=(1-time_spent)*(integral/(s*(double)
  step_number))*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_OverSpot2)
```

```
  *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(Normalvect);
  return init_mc;
}



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

  double T, t_0, T_0;
  double r, divid, time_spent, pseudo_strike, true_strike,
    pseudo_spot;
  int return_value;

  r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
  divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);

  T= ptOpt->Maturity.Val.V_DATE;
  T_0 = ptMod->T.Val.V_DATE;
  t_0= (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUB
    LE;
  time_spent= (T_0-t_0)/(T-t_0);

  if(T_0 < t_0)
    {
      Fprintf(TOSCREEN,"T_0 < t_0, untreated case{n{n{n");
      return_value = WRONG;
    }
```

```
/* Case t_0 <= T_0 */
else
  {
    pseudo_spot= (1.-time_spent)*ptMod->S0.Val.V_PDOUBLE;
    pseudo_strike= (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0
  ].Val.V_PDOUBLE-time_spent*(ptOpt->PathDep.Val.V_NUMFUNC_2
  )->Par[4].Val.V_PDOUBLE;


    true_strike= (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].
  Val.V_PDOUBLE;

    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUB
  LE= pseudo_strike;

    if (pseudo_strike<=0.)
{
  Fprintf(TOSCREEN,"FORMULE ANALYTIQUE{n{n{n");
  return_value= Analytic_KemnaVorst(pseudo_spot,
            pseudo_strike,
            time_spent,
            ptOpt->PayOff.Val.V_NUMFUNC_2,
            T-T_0,
            r,
            divid,
            &(Met->Res[0].Val.V_DOUBLE),
            &(Met->Res[1].Val.V_DOUBLE));

}
    else
return_value= FixedAsian_RobbinsMonro(pseudo_spot,
              pseudo_strike,
              time_spent,
              ptOpt->PayOff.Val.V_NUMFUNC_2,
              T-T_0,
              r,
              divid,
              ptMod->Sigma.Val.V_PDOUBLE,
              Met->Par[2].Val.V_LONG,
              Met->Par[0].Val.V_INT2,
              Met->Par[1].Val.V_ENUM.value,
```

```
                    Met->Par[4].Val.V_DOUBLE,
                    &(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));

      (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUB
    LE=true_strike;
    }
  return return_value;
}



static int CHK_OPT(MC_FixedAsian_RobbinsMonro)(void *Opt,
    void *Mod)
{
  if ( (strcmp( ((Option*)Opt)->Name,"AsianCallFixedEuro")=
    =0) || (strcmp( ((Option*)Opt)->Name,"AsianPutFixedEuro")=
    =0) )
    return OK;

  return WRONG;
}



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

      Met->Par[0].Val.V_INT2= 50;
      Met->Par[1].Val.V_ENUM.value=0;
```

```
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumRNGs;
      Met->Par[2].Val.V_LONG= 20000;
      Met->Par[4].Val.V_DOUBLE= 0.95;


    }

  type_generator= Met->Par[1].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_FixedAsian_RobbinsMonro)=
{
  "MC_FixedAsian_RobbinsMonro",
  {{"TimeStepNumber",INT2,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"N iterations",LONG,{100},ALLOW},
   {"Confidence Value",DOUBLE,{100},ALLOW},
```

```
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_FixedAsian_RobbinsMonro),
  {{"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_FixedAsian_RobbinsMonro),
  CHK_ok,
  MET(Init)
};
```

References