Help

```
/* Monte Carlo Simulation for Parisian option :
   The program provides estimations for Price an
    d Delta with
   a confidence interval.  */
/* Quasi Monte Carlo simulation is not yet allow
    ed for this routine */

#include "bs1d_doublim.h"

static int MC_ParisianOut(NumFunc_1  *L,NumFunc_1
      *U,double s,
              NumFunc_1   *PayOff,double t,
    double delay,double r,double divid,double sigma,int g
    enerator,long M,int N,double increment, double
    confidence,double *ptprice,double *ptdelta,double
    *pterror_price,double *pterror_delta,double *inf
    _price, double *sup_price, double *inf_delta,
    double *sup_delta)
{
  double g, h;
  double time,lnspot,lastlnspot,price_sample,delt
    a_sample;
  double lnspot_increment,lastlnspot_increment,pr
    ice_sample_increment;
  double gt_increment,hd_increment;
  double rloc,sigmaloc,up,low,lastup,lastlow,prob
    a,rap,gt,hd;
  double mean_price,var_price,mean_delta,var_delt
    a;
  long i;
  double uniform,proba_increment;
  int k,inside,inside_increment,correction_activ
    e;
  int init_mc, mc_or_qmc;
  int simulation_dim;
  double alpha, z_alpha;

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

```
z_alpha= Inverse_erf(1.- alpha);

/*One forces N if necessary so that delay
  !!!!!!!!!!  WARNING        !!!!!!!!
  be greater than the time step increment h*/
h=t/(double)N;
if (delay<=h)
  {
    N=(int)ceil(t/delay)+1;
    h=t/(double)N;

    Fprintf(TOSCREEN,"WARNING!!! N is forced
  to %d{n",N);
  }

/*Initialisation*/
mean_price=0.0;
mean_delta=0.0;
var_price=0.0;
var_delta=0.0;
/* Maximum Size of the random vector we need
  in the simulation */
simulation_dim= N;

rloc=(r-divid-SQR(sigma)/2.)*h;
sigmaloc=sigma*sqrt(h);

/*Coefficient for the computation of the exit
  probability*/
rap=1./(sigmaloc*sigmaloc);

/*MC sampling*/
init_mc= InitGenerator(generator, simulation_
  dim,M);
/* Test after initialization for the generator
  */
if(init_mc == OK)
  {
    mc_or_qmc= Rand_Or_Quasi(generator);

    /* Begin M iterations */
```

```
  for(i=1;i<=M;i++)
{
  gt=0.;
  hd=0.;
  gt_increment=0.;
  hd_increment=0.;
  lnspot=log(s);

  /*Inside=0 if the path stays beyond the
barrier uninterruptedly
  for longer than delay*/
  inside=1;
  inside_increment=1;

  time=0.;
  k=0;

  /*Up and Down Barrier at time*/
  up=log((U->Compute)(U->Par,time));
  low=log((L->Compute)(L->Par,time));

  /*Simulation of i-th path until Inside=0*
/
  while  (((inside) && (k<N)) ||((inside_
increment) && (k<N)))
  {
    correction_active=0;

    lastlnspot=lnspot;
    lastup=up;
    lastlow=low;

    time+=h;
    g= Gaussians[mc_or_qmc](1, CREATE, 0,
generator);
    lnspot+=rloc+sigmaloc*g;

    lnspot_increment=lnspot+increment;
    lastlnspot_increment=lastlnspot+incre
ment;
```

```
    up=log((U->Compute)(U->Par,time));
    low=log((L->Compute)(L->Par,time));

    /*Check if the i-th path has reached
the barriers at time*/
    /*Otherwise there is no extinction*/
    if (inside)
    if (lnspot>up)
      {
      if (lastlnspot>up)
        {
        proba=exp(-2.*rap*((lastlnspot-
lastup)*(lnspot-lastup)-(lastlnspot-lastup)*(up-
lastup)));
        correction_active=1;
        uniform=Uniform(generator);
        if (uniform<proba)
          gt=time;
        }
      else gt=(time-h)+(up-lastlnspot)/(
lnspot-lastlnspot)*h;
      }

    if (inside_increment)
    if (lnspot_increment>up)
      {
      if (lastlnspot_increment>up)
        {
        proba_increment=exp(-2.*rap*((
lastlnspot_increment-lastup)*(lnspot_increment-la
stup)-(lastlnspot_increment-lastup)*(up-lastup)));
        if (!correction_active)
          uniform=Uniform(generator);
        if (uniform<proba)
          gt_increment=time;
        }
      else gt_increment=(time-h)+(up-la
stlnspot_increment)/(lnspot_increment-lastlnspot_
increment)*h;
      }
```

```
    if (inside_increment)
    if (lnspot_increment<low)
      {
      if (lastlnspot_increment<low)
        {
        proba_increment=exp(-2.*rap*((
lastlnspot_increment-lastlow)*(lnspot_increment-
lastlow)+(lastlnspot_increment-lastlow)*(low-la
stlow)));
        correction_active=1;
        uniform=Uniform(generator);
        if (uniform<proba_increment)
          gt_increment=time;
        }
      else gt_increment=(time-h)+(low-la
stlnspot_increment)/(lnspot_increment-lastlnspot_
increment)*h;
      }

    if (inside)
    if (lnspot <low)
      {
      if (lastlnspot <low)
        {
        proba =exp(-2.*rap*((lastlnspo
t -lastlow)*(lnspot -lastlow)+(lastlnspot -lastl
ow)*(low-lastlow)));
        if (!correction_active)
          uniform=Uniform(generator);
        if (uniform<proba)
          gt =time;
        }
      else gt =(time-h)+(low-lastlnspot
)/(lnspot -lastlnspot)*h;
      }

    if (inside) {

    if ((lnspot<=up)&&(lnspot>=low))
      gt=time;
    hd=time-gt;
```

```
    if(hd>delay)
      {
      inside=0;
      price_sample=0.;
      }
    }

    if (inside_increment) {

    if ((lnspot_increment<=up)&&(lnspot_
increment>=low))
      gt_increment=time;

    hd_increment=time-gt_increment;

    if(hd_increment>delay)
      {
      inside_increment=0;
      price_sample_increment=0.;
      }
    }

    k++;
  }

  if (inside)
  price_sample=exp(-r*t)*(PayOff->Compute)
(PayOff->Par,exp(lnspot));

  if (inside_increment)
  price_sample_increment=exp(-r*t)*(PayOf
f->Compute)(PayOff->Par,exp(lnspot_increment));

  /*Delta*/
  delta_sample=(price_sample_increment-pric
e_sample)/(increment*s);

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

```
      /*Sum of Squares*/
      var_price+= SQR(price_sample);
      var_delta+= SQR(delta_sample);
    }
      /* End N iterations */

      /*Price*/
      *ptprice=mean_price/(double)M;
      *pterror_price= sqrt(var_price/(double)M -
    SQR(*ptprice))/sqrt(M-1);
      /*Delta*/
      *ptdelta=mean_delta/(double) M;
      *pterror_delta= sqrt(var_delta/(double)M-SQ
    R(*ptdelta))/sqrt((double)M-1);

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

      /* Delta Confidence Interval */
      *inf_delta= *ptdelta - z_alpha*(*pterror_d
    elta);
      *sup_delta= *ptdelta + z_alpha*(*pterror_d
    elta);
    }
  return init_mc;
}


int CALC(MC_ParisianOut)(void*Opt,void *Mod,Pric
    ingMethod *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 MC_ParisianOut(ptOpt->LowerLimit.Val.V_
    NUMFUNC_1,
            ptOpt->UpperLimit.Val.V_
    NUMFUNC_1,
            ptMod->S0.Val.V_PDOUBLE,ptOpt->
    PayOff.Val.V_NUMFUNC_1,
            ptOpt->Maturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
            (ptOpt->LowerLimit.Val.V_
    NUMFUNC_1)->Par[1].Val.V_PDOUBLE,
            r,
            divid,
            ptMod->Sigma.Val.V_PDOUBLE,
            Met->Par[1].Val.V_INT,
            Met->Par[0].Val.V_LONG,
            Met->Par[2].Val.V_INT,
            Met->Par[3].Val.V_PDOUBLE,
            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));
}


int CHK_OPT(MC_ParisianOut)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  if ((opt->RebOrNo).Val.V_BOOL==NOREBATE)
    if (((opt->OutOrIn).Val.V_BOOL==OUT)&&((opt->
    EuOrAm).Val.V_BOOL==EURO)&&((opt->Parisian).Val.
    V_BOOL==OK))
      return OK;
```

```
  return  WRONG;
}



static int MET(Init)(PricingMethod *Met)
{
  static int first=1;
  int type_generator;

  type_generator= Met->Par[1].Val.V_INT;


  if (first)
    {
      Met->Par[0].Val.V_LONG=10000;
      Met->Par[1].Val.V_INT=0;
      Met->Par[2].Val.V_INT2=250;
      Met->Par[3].Val.V_PDOUBLE=0.01;
      Met->Par[4].Val.V_PDOUBLE= 0.95;

      first=0;
    }

  if(Rand_Or_Quasi(type_generator)==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_ParisianOut)=
{
  "MC_ParisianOut",
  {{"Iterations",LONG,100,ALLOW},
   {"RandomGenerator",GENER,100,ALLOW},
   {"TimeStepNumber",INT2,100,ALLOW},
   {"Delta Increment Rel",PDOUBLE,100,ALLOW},
   {"Confidence Value",DOUBLE,100,ALLOW},
   {" ",END,0,FORBID}},
  CALC(MC_ParisianOut),
  {{"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} ,
   {" ",END,0,FORBID}},
  CHK_OPT(MC_ParisianOut),
  CHK_mc_generator,
  MET(Init)
} ;
```

References