```
   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_lim.h"


static int check_parisianin(double *gt,double ln
    spot,double lastlnspot,
           double barrier, double lastbarri
    er,
           double *gt_increment,
           double lnspot_increment,double la
    stlnspot_increment,
           double rap,int upordown,double h,
    double time,
           int *correction_active,int genera
    tor)
{
  double proba,uniform;

  if (((upordown==0)&&(lnspot<barrier))||((upord
    own==1)&&(lnspot>barrier)))
    {
      if (((lastlnspot>barrier)&&(upordown==1))||
    ((lastlnspot<barrier)&&(upordown==0)))
    {
      proba=exp(-2.*rap*((lastlnspot-lastbarri
    er)*(lnspot-lastbarrier)-(lastlnspot-lastbarrier)*
    (barrier-lastbarrier)));
      *correction_active=1;
      uniform=Uniform(generator);
      if (uniform<proba)
      *gt=time;
    }
      else *gt=(time-h)+(barrier-lastlnspot)/(ln
    spot-lastlnspot)*h;
```

```
    }
  else *gt=time;

  if (((upordown==0)&&(lnspot_increment<barrier))
    ||((upordown==1)&&(lnspot_increment>barrier)))
    {
      if (((lastlnspot_increment>barrier)&&(upord
    own==1))||((lastlnspot_increment<barrier)&&(upord
    own==0)))
    {
      proba=exp(-2.*rap*((lastlnspot_increment-
    lastbarrier)*(lnspot_increment-lastbarrier)-(la
    stlnspot_increment-lastbarrier)*(barrier-lastbarri
    er)));
      if (!*correction_active)
      uniform=Uniform(generator);
      if (uniform<proba)
      *gt_increment=time;
    }
      else *gt_increment=(time-h)+(barrier-lastl
    nspot_increment)/(lnspot_increment-lastlnspot_inc
    rement)*h;
    }
  else *gt_increment=time;

  return OK;

}

static int MC_ParisianIn(int upordown,double s,
    NumFunc_1 *PayOff,double l,double t,double delay,
    double r,double divid,double sigma,int generator,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,lns;
  double lnspot_increment,lastlnspot_increment,pr
```

```
  ice_sample_increment;
double rloc,sigmaloc,barrier,lastbarrier,rap;
double gt,hd,gt_increment,hd_increment;
double mean_price,var_price,mean_delta,var_delt
  a;
long i;
int k,inside,inside_increment;
int correction_active,dummy;
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;

barrier=log(l);
```

```
lns=log(s);

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.;
    lnspot=lns;

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

    time=0;
    k=0;

    /*Barrier at time*/
    barrier=log(l);

    /*Simulation of i-th path until Inside=0*
  /
    while (((inside) && (k<N)) ||((inside_inc
```

```
rement) && (k<N)))
  {
    correction_active=0;

    lastlnspot=lnspot;
    lastbarrier=barrier;

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

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

    barrier=log(l);

    /*Check if the i-th path has reached
the barrier at time*/
    /*Otherwise there is no extinction*/
    if (upordown==0)
    dummy=check_parisianin(&gt,lnspot,la
stlnspot,barrier,lastbarrier,
                  &gt_increment,lnsp
ot_increment,lastlnspot_increment,
                  rap,upordown,h,
time,&correction_active,generator);
    else
    dummy=check_parisianin(&gt_increment,
lnspot_increment,lastlnspot_increment,barrier,la
stbarrier,&gt,lnspot,lastlnspot,rap,upordown,h,
time,&correction_active,generator);

    hd=time-gt;
    hd_increment=time-gt_increment;

    if(hd>delay)
    {
      inside=0;
      if (time>t) time=t;
```

```
      price_sample=exp(-r*time)*Boundary(
exp(lnspot),PayOff,t-time,r,divid,sigma);
    }

    if(hd_increment>delay)
    {
      inside_increment=0;
      if (time>t) time=t;
      price_sample_increment=exp(-r*time)
*Boundary(exp(lnspot_increment),PayOff,t-time,r,
divid,sigma);
    }

    k++;
  }

  if (inside)
  price_sample=0.;

  if (inside_increment)
  price_sample_increment=0.;

  /*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*/
      if ( (PayOff->Compute)==&Put)
    *ptdelta=-mean_delta/(double) M;
      if ( (PayOff->Compute)==&Call)
    *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_ParisianIn)(void *Opt,void *Mod,Pric
    ingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid,limit;
  int upordown;

  r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
  divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
  limit=((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)
    ((ptOpt->Limit.Val.V_NUMFUNC_1)->Par,ptMod->T.
    Val.V_DATE);
```

```
  if ((ptOpt->DownOrUp).Val.V_BOOL==DOWN)
    upordown=0;
  else upordown=1;

  return MC_ParisianIn(upordown,
             ptMod->S0.Val.V_PDOUBLE,ptOpt->
    PayOff.Val.V_NUMFUNC_1,
             limit,
             ptOpt->Maturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
             (ptOpt->Limit.Val.V_NUMFUNC_1)-
    >Par[4].Val.V_PDOUBLE,
             r,
             divid,ptMod->Sigma.Val.V_PDOUBL
    E,
             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_ParisianIn)(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==IN)
      if ((opt->EuOrAm).Val.V_BOOL==EURO)
```

```
    if ((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_ParisianIn)=
{
  "MC_Parisianin",
  {{"Iterations",LONG,100,ALLOW},
   {"RandomGenerator",GENER,100,ALLOW},
   {"TimeStepNumber",INT2,100,ALLOW},
   {"Delta Increment Rel",DOUBLE,100,ALLOW},
   {"Confidence Value",DOUBLE,100,ALLOW},
   {" ",END,0,FORBID}},
  CALC(MC_ParisianIn),
  {{"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_ParisianIn),
  CHK_mc_generator,
  MET(Init)
} ;
```

References