```
   Help
/* Monte Carlo Simulation for Barrier option :
The program provides estimations for Price and De
    lta with
a confidence interval.  */
/* Quasi Monte Carlo simulation is not yet allow
    ed for this routine */

#include "bs1d_lim.h"

/* Check if the spot has crossed the barrier dur
    ing the time interval */
static int check_barrierout(int *inside,double ln
    spot,double lastlnspot,
              double barrier, double lastb
    arrier,
              int *inside_increment,
              double lnspot_increment,
    double lastlnspot_increment,
              double rap,double r,double
    time,
              int *correction_active,
              double rebate,int generator,
              double *price_sample,double
    *price_sample_increment)
{
  double proba,uniform;
  if (*inside)
  {
    proba=exp(-2.*rap*((lastlnspot-lastbarrier)
    *(lnspot-lastbarrier)-(lastlnspot-lastbarrier)*(
    barrier-lastbarrier)));
    uniform=Uniform(generator);
    *correction_active=1;
    if (uniform<proba)
    {
      *inside=0;
      *price_sample=exp(-r*time)*rebate;
    }
  }
  if (*inside_increment)
```

```
  {
    proba=exp(-2.*rap*((lastlnspot_increment-la
    stbarrier)*(lnspot_increment-lastbarrier)-(lastlns
    pot_increment-lastbarrier)*(barrier-lastbarrier))
    );
    if (!*correction_active)
      uniform=Uniform(generator);

    if (uniform<proba)
    {
      *inside_increment=0;
      *price_sample_increment=exp(-r*time)*re
    bate;
    }
  }
  return OK;
}

double regul(double x)
{
  if (x<=-1.)
    return 0.0;
  else{
    if ((x>-1.)&&(x<=0))
      return (x+1.)*exp(-1./(x*x*(x-1.)*(x-1.)
    ));
    else
      return 1.0;
  }
}
double der_regul(double x)
{
  if ((x<=-1) || (x>=0))
    return 0.0;
  else
    return (1.+2.*x*((2.*x-1)/(x*x*x*(x-1.)*(x-
    1.)*(x-1.))))*exp(-1./(x*x*(x-1.)*(x-1.)));

}
static int MC_OutBaldi_97(int upordown, double s,
     NumFunc_1 *PayOff, double l, double rebate,
```

```
    double t, double r, double divid, double sigma, int g
    enerator, long Nb, int M, double increment,
    double confidence,int delta_met, double *ptprice,
    double *ptdelta, double *pterror_price, double *pt
    error_delta, double *inf_price, double *sup_price,
    double *inf_delta, double *sup_delta)
{
  double h=t/(double)M;
  double time,lnspot,lastlnspot,lnspot_incremen
    t,lastlnspot_increment,price_sample,price_sample_
    increment,delta_sample,lns;
  double rloc,sigmaloc,barrier,lastbarrier,rap,
    g;
  double mean_price,var_price,mean_delta,var_de
    lta;
  long i;
  int k,inside,inside_increment,j;
  int correction_active,dummy;
  int init_mc, mc_or_qmc;
  int simulation_dim;

  double alpha, z_alpha,a,maxlnspot,minlnspot,
    temp,intder,intsto,intreg;
  double *tauM,*taum,*domprocess;
  tauM = (double*)malloc(sizeof(double)*M);
  taum = (double*)malloc(sizeof(double)*M);
  domprocess = (double*)malloc(sizeof(double)*(
    M+1));

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

  /*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= M;

barrier=log(l);
lns=log(s);
a = l-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,Nb);
/* Test after initialization for the generator
   */
if(init_mc == OK)
{
  mc_or_qmc= Rand_Or_Quasi(generator);
  /* Begin N iterations */
  for(i=1;i<=Nb;i++)
  {
    time=0.;
    lnspot=lns;
    intsto=0.0;
    intreg=0.0;
    intder=0.0;
    taum[0]=0.0;
    tauM[0]=0.0;
    /*Barrier at time*/
    barrier=log(l);
    maxlnspot=lns;
    domprocess[0]=0.0;
    minlnspot=lns;
    /*Inside=0 if the path reaches the barr
  ier*/
    inside=1;
    inside_increment=1;

    k=0;
```

```
  /*Simulation of i-th path until its exi
t if it does*/
  while (((inside) && (k<M)) ||((inside_
increment) && (k<M)))
  {
    correction_active=0;

    lastlnspot=lnspot;
    lastbarrier=barrier;

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

    /* Tools for computation of Malliavi
n Weights*/
    if (delta_met>1){
      if (lnspot>maxlnspot){
        tauM[k+1] = time;
        domprocess[k+1]=domprocess[k]-
maxlnspot;
        maxlnspot = lnspot;
        domprocess[k+1]+=maxlnspot;
      }else tauM[k+1]=tauM[k];
      if (lnspot<minlnspot){
        taum[k+1] = time;
        domprocess[k+1]=domprocess[k]+
minlnspot;
        minlnspot = lnspot;
        domprocess[k+1]-=minlnspot;
      }else taum[k+1]=taum[k];
      /*printf("%lf %lf %lf %lf %lf %lf{
n",lnspot,maxlnspot,minlnspot,taum[k],
          tauM[k],domprocess);*/
      intsto+=regul((a-2.*exp(domproces
s[k]))/a)*sqrt(h)*g/sigma;
      intreg+=regul((a-2.*exp(domproces
s[k]))/a)*h;
      temp=0.0;
```

```
      for(j=0;j<=k;j++){
        if ((j*h<=tauM[k])&& (j*h>=tau
m[k]))
          temp+=regul((a-2.*exp(dompr
ocess[k]))/a);
        if ((j*h>=tauM[k])&& (j*h<=tau
m[k]))
          temp+=regul((a-2.*exp(dompr
ocess[k]))/a);
      }

      intder+=der_regul((a-2.*exp(dompr
ocess[k]))/a)*temp*h*h;
      //printf("%lf %lf %lf %lf %lf %lf{
n",intsto,intreg,intder,temp,der_regul((a-2.*dom
process[k])/a),(a-2.*domprocess[k])/a);
    }

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

    barrier=log(l);

    /*Check if the i-th path has reached
the barrier at time*/
    if (inside)
      if (((upordown==0)&&(lnspot<barrie
r))||((upordown==1)&&(lnspot>barrier)))
      {
        inside=0;
        price_sample=exp(-r*time)*reba
te;
      }

      if (inside_increment)
        if (((upordown==0)&&(lnspot_inc
rement<barrier))||((upordown==1)&&(lnspot_incre
ment>barrier)))
      {
        inside_increment=0;
```

```
        price_sample_increment=exp(-r*
time)*rebate;
      }

      /*Check if the i-th path has reac
hed the barrier during (time-1,time)*/
      if (upordown==0)
        dummy=check_barrierout(&inside,
lnspot,lastlnspot,barrier,lastbarrier,
        &inside_increment,lnspot_incre
ment,lastlnspot_increment, rap,r,time,&correction_
active,rebate, generator,&price_sample,&price_sa
mple_increment);
      else
        dummy=check_barrierout(&inside_
increment,lnspot_increment,lastlnspot_increment,
        barrier,lastbarrier,&inside,lnsp
ot,lastlnspot,rap,r,time,&correction_active,reba
te,generator,&price_sample_increment,&price_sample
);

      k++;
  }/*while*/

  if (inside)
  {
    price_sample=exp(-r*t)*(PayOff->Compu
te)(PayOff->Par,exp(lnspot));
  }

  if (inside_increment)
  {
    price_sample_increment=exp(-r*t)*(Pay
Off->Compute)(PayOff->Par,exp(lnspot_increment));
  }

  /*Delta*/
  if (delta_met==1)
    delta_sample=(price_sample_increment-
price_sample)/(increment*s);
  else{
```

```
    if (!inside)
      delta_sample=sigma*10*exp(-r*t)*pr
ice_sample*(intsto/intreg+intder/(intreg*intreg))
/(s);
    else
      delta_sample=0.0;
    /*printf("%lf %lf %lf %lf %lf{n",delt
a_sample,price_sample,intsto,intreg,intder);*/
    delta_sample = (price_sample_incremen
t-price_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)Nb;
*pterror_price= sqrt(var_price/(double)Nb -
 SQR(*ptprice))/sqrt(Nb-1);
/*Delta*/
*ptdelta=mean_delta/(double) Nb;
*pterror_delta= sqrt(var_delta/(double)Nb-
SQR(*ptdelta))/sqrt((double)Nb-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_OutBaldi)(void *Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid,limit,rebate,increment=0.01;
  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)->Compu
    te)((ptOpt->Limit.Val.V_NUMFUNC_1)->Par,ptMod->T.
    Val.V_DATE);
  rebate=((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compu
    te)((ptOpt->Rebate.Val.V_NUMFUNC_1)->Par,ptMod->T.
    Val.V_DATE);

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

  return MC_OutBaldi_97(upordown,
    ptMod->S0.Val.V_PDOUBLE,
    ptOpt->PayOff.Val.V_NUMFUNC_1,
    limit,
    rebate,
    ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_
    DATE,
    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->Par[5].Val.V_RGINT13,
    &(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_OutBaldi)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

  if ((opt->OutOrIn).Val.V_BOOL==OUT)
    if ((opt->EuOrAm).Val.V_BOOL==EURO)
      if ((opt->Parisian).Val.V_BOOL==WRONG)

        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;
    Met->Par[5].Val.V_RGINT13= 2;

    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_OutBaldi)=
{
  "MC_BaldiCaramellinoIovino",
  {{"N iterations",LONG,100,ALLOW},
  {"RandomGenerator",GENER,100,ALLOW},
  {"TimeStepNumber M",INT2,100,ALLOW},
  {"Delta Increment Rel",DOUBLE,100,ALLOW},
  {"Confidence Value",DOUBLE,100,ALLOW},
  {"Delta Method: 1->Finite Difference 2-> Mal
```

```
    liavin",RGINT13,100,ALLOW},
  {" ",END,0,FORBID}},
  CALC(MC_OutBaldi),
  {{"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_OutBaldi),
  CHK_mc_generator,
  MET(Init)
} ;
```

References