```
   Help
/* Monte Carlo Simulation for Barrier 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"

/* Check if the spot has crossed the barrier dur
    ing the time interval */
static int check_barrierin(int *inside,double ln
    spot,double lastlnspot,
          double barrier, double lastbarri
    er,
          int *inside_increment,
          double lnspot_increment,double la
    stlnspot_increment,
          double rap, double time,
          int *correction_active,
          int generator,
          double *exit_time, double *exit_
    time_increment)
{
  double proba,uniform;

  if (*inside)
    {
      proba=exp(-2.*rap*((lastlnspot-lastbarrier)
    *(lnspot-lastbarrier)-(lastlnspot-lastbarrier)*(
    barrier-lastbarrier)));
      /* Simulation of an uniform variable */
      uniform=Uniform(generator);

      *correction_active=1;
      if (uniform<proba)
    {
      *inside=0;
      *exit_time=time;
    }
```

```
    }
  if (*inside_increment)
    {
      proba=exp(-2.*rap*((lastlnspot_increment-la
    stbarrier)*(lnspot_increment-lastbarrier)-(lastlns
    pot_increment-lastbarrier)*(barrier-lastbarrier))
    );
      if (!*correction_active)
    /* Simulation of an uniform variable */
    uniform=Uniform(generator);

      if (uniform<proba)
    {
      *inside_increment=0;
      *exit_time_increment=time;
    }
    }
  return OK;
}

static int MC_InBaldi_97(int upordown, double s,
    NumFunc_1 *PayOff, double l, double rebate, double t,
    double r, double divid, double sigma, int generator,
    long Nb, int M, double increment, double confidenc
    e, double *ptprice, double *ptdelta, double *pt
    error_price, double *pterror_delta, double *inf_price,
     double *sup_price, double *inf_delta, double *
    sup_delta)
{
  double h=t/(double)M;
  double time, lnspot, lastlnspot, price_sample,
    delta_sample, exit_time;
  double lnspot_increment, lastlnspot_increment,
    price_sample_increment, exit_time_increment;
  double rloc, sigmaloc, z, sigmat, barrier, la
    stbarrier, rap, g;
  double mean_price, var_price, mean_delta, var_
    delta;
  long i;
  int k, inside, inside_increment, correction_ac
    tive, 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);

/*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;

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

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

/*MonteCarlo sampling*/
init_mc= InitGenerator(generator, simulation_
  dim,Nb);
if(init_mc == OK)
  {
    mc_or_qmc= Rand_Or_Quasi(generator);

    /* Begin N iterations */
    for(i=1;i<=Nb;i++)
  {
    time=0.;
    lnspot=log(s);

    /*Up and Down Barrier at time*/
```

```
  barrier=log(l);

  /*Inside=0 if the path reaches the barrie
r*/
  inside=1;
  inside_increment=1;
  k=0;

  /*Simulation of i-th path until its exit
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;

    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<barrier))
||((upordown==1)&&(lnspot>barrier)))
      {
      inside=0;
      exit_time=time;
      }

    if (inside_increment)
    if (((upordown==0)&&(lnspot_incremen
```

```
t<barrier))||((upordown==1)&&(lnspot_increment>ba
rrier)))
      {
      inside_increment=0;
      exit_time_increment=time;
      }

    /*Check if the i-th path has reached
the barrier during (time-1,time)*/
    if (upordown==0)
    dummy= check_barrierin(&inside,lnspot
,lastlnspot,barrier,lastbarrier,
    &inside_increment,lnspot_increment,la
stlnspot_increment, rap,time,
    &correction_active, generator,&exit_
time,&exit_time_increment);
    else
    dummy= check_barrierin(&inside_incre
ment,lnspot_increment,lastlnspot_increment,barrie
r,
    lastbarrier,&inside,lnspot,lastlnspo
t,rap,time,&correction_active,generator,&exit_
time_increment,&exit_time);

    k++;
  }
  /*Inside=0 means that the payoff does no
t nullify
  Inside=1 means that the payoff is equal
to the rebate*/

  if (inside==0)
  {
    if (t-exit_time>0)
    price_sample=exp(-r*exit_time)*Bound
ary(l,PayOff,t-exit_time,r,divid,sigma);
    else
    price_sample=exp(-r*t)*(PayOff->Compu
te)(PayOff->Par,l);
  }
  else
```

```
  price_sample=exp(-r*t)*rebate;

  if (inside_increment==0)
  {
    if (t-exit_time_increment>0)
    price_sample_increment=exp(-r*exit_
time_increment)*Boundary(l,PayOff,t-exit_time_incre
ment,r,divid,sigma);
    else
    price_sample_increment=exp(-r*t)*(Pay
Off->Compute)(PayOff->Par,l);
  }
  else
  price_sample_increment=exp(-r*t)*rebate;

  /*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)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_InBaldi)(void*Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r,divid,limit,rebate;
  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);
  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_InBaldi_97(upordown,
            ptMod->S0.Val.V_PDOUBLE,
```

```
            ptOpt->PayOff.Val.V_NUMFUNC_1,
            limit,
            rebate,
            ptOpt->Maturity.Val.V_DATE-pt
    Mod->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->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_InBaldi)(void *Opt, void *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);


  if ((opt->OutOrIn).Val.V_BOOL==IN)
    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;

      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_InBaldi)=
{
  "MC_BaldiCaramellinoIovino",
```

```
  {{"N iterations",LONG,100,ALLOW},
   {"RandomGenerator",GENER,100,ALLOW},
   {"TimeStepNumber M",INT2,100,ALLOW},
   {"Delta Increment Rel",PDOUBLE,100,ALLOW},
   {"Confidence Value",DOUBLE,100,ALLOW},
   {" ",END,0,FORBID}},
  CALC(MC_InBaldi),
  {{"Price",DOUBLE,100,FORBID},
   {"Delta",DOUBLE,100,FORBID} ,
   {"ErrorPrice",DOUBLE,100,FORBID},
   {"ErrorDelta",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_InBaldi),
  CHK_mc_generator,
  MET(Init)
} ;
```

References