```
   Help
/* Monte Carlo Simulation for double Barrier opti
    on :
   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 proba_barrierin(double lnspot,double
    lastlnspot,
          double lastlow, double lastup,
    double low,double up,
          double rap,double *proba,int *type
    _barrier)
{

  if ((lnspot+lastlnspot)<(lastup+lastlow))
    {
      *proba=exp(-2.*rap*((lastlnspot-lastlow)*(
    lnspot-lastlow)-(lastlnspot-lastlow)*(low-lastlow
    )));
      *type_barrier=0;
    }
  else
    {
      *proba=exp(-2.*rap*((lastlnspot-lastup)*(ln
    spot-lastup)-(lastlnspot-lastup)*(up-lastup)) );
      *type_barrier=1;
    }
  return OK;
}

static int MC_InBaldi_97(NumFunc_1 *L, NumFunc_1
    *U, NumFunc_1 *Rebate, double s, NumFunc_1 *Pay
    Off, double t, double r, double divid, double sig
    ma, int generator, long Nb, int M, double incremen
    t, double confidence, double *ptprice, double *pt
```

```
    delta, double *pterror_price, double *pterror_del
    ta, double *inf_price, double *sup_price, double
    *inf_delta, double *sup_delta)
{
  double h=t/(double)M;
  double time,lnspot,lastlnspot,price_sample,exi
    t_time,exit_time_increment;
  double lnspot_increment,lastlnspot_increment,pr
    ice_sample_increment,delta_sample;
  double rloc, sigmaloc, up, low, lastup, lastlow
    , proba, rap, z, proba_increment, uniform, g;
  double mean_price,var_price,mean_delta,var_delt
    a;
  long i;
  int k,inside,type_barrier=2,type_barrier_incre
    ment=2,inside_increment,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);

  /*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);

    for(i=1;i<=Nb;i++)
  {
    time=0.;
    lnspot=log(s);

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

    /*Inside=0 if the path reaches the barrie
  rs*/
    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)))
    {
      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*/
    if (inside)
    {
      if (lnspot>up)
      {
        type_barrier=1;
        inside=0;
        exit_time=time;
      }
      if (lnspot<low)
      {
        type_barrier=0;
        inside=0;
        exit_time=time;
      }
    }

    if (inside_increment)
    {
      if (lnspot_increment>up)
      {
        type_barrier_increment=1;
        inside_increment=0;
        exit_time_increment=time;
      }


      if (lnspot_increment<low)
      {
        type_barrier_increment=0;
        inside_increment=0;
        exit_time_increment=time;
      }
    }

    /*Check if the i-th path has reached
the barriers during (time-1,time)*/
```

```
    if ((inside)&&(inside_increment))
    {
      dummy=proba_barrierin(lnspot,lastl
nspot,lastlow,lastup,low,up, rap,&proba,&type_ba
rrier);
      dummy=proba_barrierin(lnspot_incre
ment, lastlnspot_increment,lastlow,lastup,low,up,
rap ,&proba_increment,&type_barrier_increment);

      uniform=Uniform(generator);
      if (uniform<proba)
      {
        inside=0;
        exit_time=time;
      }
      if (uniform<proba_increment)
      {
        inside_increment=0;
        exit_time_increment=time;
      }
    }

    if ((inside)&&(!inside_increment))
    {
      dummy=proba_barrierin(lnspot,lastl
nspot,lastlow,lastup,low,up, rap,&proba,&type_ba
rrier);
      if (Bernoulli(proba,generator))
      {
        inside=0;
        exit_time=time;
      }
    }

    if ((!inside)&&(inside_increment))
    {
      dummy=proba_barrierin(lnspot_incre
ment, lastlnspot_increment,lastlow,lastup,low,up,
rap ,&proba_increment,&type_barrier_increment);
      if (Bernoulli(proba_increment,gen
erator))
```

```
      {
        inside_increment=0;
        exit_time_increment=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)
    {
      if(type_barrier==1)
      price_sample=exp(-r*exit_time)*Bou
ndary((U->Compute)(U->Par,exit_time),PayOff,t-exi
t_time,r,divid,sigma);
      else
      price_sample=exp(-r*exit_time)*Bou
ndary((L->Compute)(L->Par,exit_time),PayOff,t-exi
t_time,r,divid,sigma);
    } else {
      if(type_barrier==1)
      price_sample=exp(-r*t)*(PayOff->
Compute)(PayOff->Par,(L->Compute)(L->Par,t));
      else
      price_sample=exp(-r*t)*(PayOff->
Compute)(PayOff->Par,(L->Compute)(L->Par,t));
    }
  }
  else
  price_sample=exp(-r*t)*(Rebate->Compute)
(Rebate->Par,t);

  if (inside_increment==0)
  {
    if (t-exit_time_increment>0)
    {
```

```
      if(type_barrier_increment==1)
      price_sample_increment=exp(-r*exi
t_time_increment)*Boundary((U->Compute)(U->Par,
exit_time),PayOff,t-exit_time_increment,r,divid,si
gma);
      else
      price_sample_increment=exp(-r*exi
t_time_increment)*Boundary((L->Compute)(L->Par,
exit_time),PayOff,t-exit_time_increment,r,divid,si
gma);
    } else
      {
      if(type_barrier==1)
        price_sample_increment=exp(-r*t)
*(PayOff->Compute)(PayOff->Par,(L->Compute)(L->
Par,t));
      else
        price_sample_increment=exp(-r*t)
*(PayOff->Compute)(PayOff->Par,(L->Compute)(L->
Par,t));
      }
  }
  else
  price_sample_increment=exp(-r*t)*(Reba
te->Compute)(Rebate->Par,t);

  /*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);
}

  /*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;

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

  return MC_InBaldi_97(ptOpt->LowerLimit.Val.V_
    NUMFUNC_1,
             ptOpt->UpperLimit.Val.V_
    NUMFUNC_1,
             ptOpt->Rebate.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,
              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->Parisian).Val.V_BOOL==WRONG)
    if (((opt->OutOrIn).Val.V_BOOL==IN)&&((opt->
    EuOrAm).Val.V_BOOL==EURO))
      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_InBaldi",
  {{"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} ,
   {"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_InBaldi),
  CHK_mc_generator,
  MET(Init)
} ;
```

References