```
   Help
/* Monte Carlo and Quasi-Monte Carlo Simulation
    for Lookback option on maximum:
   Call Fixed Euro and Put Floating Euro.
   The program provides estimations for Price an
    d Delta with
   a confidence interval (for MC only).  */


#include  "bs1d_pad.h"


static double inverse_max(double s1, double s2,
    double h,double sigma,double un)
{
  return ((s1+s2)+sqrt(SQR(s1-s2)-2*SQR(sigma)*h*
    log(1.-un)))/2.;
}



static int  LookBackSup_AndersenMontecarlo(
    double s, double pad, double strike, NumFunc_2 *p,
    double t, double r, double divid, double sigma, long
    N, int generator, double confidence, double *ptpr
    ice, double *ptdelta, double *pterror_price,
    double *pterror_delta, double *inf_price, double *sup_
    price, double *inf_delta, double *sup_delta)

{
  long i;
  double gs, un,max_log_norm,log_pad,log_s;
  int init_mc, mc_or_qmc;
  int simulation_dim;
  double forward, forward_stock, exp_sigmaxwt, S_
    T, S_max,sigma_sqrt;
  double price_sample=0., delta_sample=0., mean_
    price, mean_delta, var_price, var_delta;

  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;
/* Size of the random vector we need in the si
  mulation */
simulation_dim= 2;

/*Median forward stock and delta values*/
sigma_sqrt=sigma*sqrt(t);
forward= exp(((r-divid)-SQR(sigma)/2.0)*t);
forward_stock= s*forward;
log_s=log(s);
log_pad=log(pad);

/* Monte Carlo sampling */
init_mc= InitGenerator(generator, simulation_
  dim, N);
/* Test after initialization for the generator
  */
if(init_mc == OK)
  {
    mc_or_qmc= Rand_Or_Quasi(generator);
    /* We test if simulation is MC or QMC.
 This involves two parts in the program becau
  se simulation for random vector
 must be called from different functions */

    /* MC simulation case */
    if(mc_or_qmc == MC)
{
  for(i=1; i<=N; i++)
    /* Begin N iterations */
    {
      /* For MC simulation, generation of two
```

```
independent variables,
 a gaussian one and a uniform one, can be
realized with the
 same preudo random number generator withou
t problem of independence*/
    /* Simulation of a gaussian variable ac
cording to the generator type */
    gs= Gaussians[mc_or_qmc](1, CREATE, 0, g
enerator);

    /* Second variable: uniform to generate
the maximum */
    un= Uniform(generator);
    exp_sigmaxwt= exp(sigma_sqrt*gs);
    S_T= forward_stock*exp_sigmaxwt;

    max_log_norm=inverse_max(log_s, log(S_T)
, t, sigma, un);
    S_max= exp(MAX(log(pad),max_log_norm));

    /* Price and Delta */
    /* CallFixedEuro */
    if (p->Compute == &Call_OverSpot2)
{
  price_sample= (p->Compute)(p->Par, strik
e, S_max);
  delta_sample= 0;
  if(pad==s)
    delta_sample=S_max/s;
  else
    {
      if(log_pad>max_log_norm)
  delta_sample=0.;
      else  delta_sample=S_max/s;
    }
}
    else
/* PutFloatingEuro */
if (p->Compute == &Put_StrikeSpot2)
  {
    price_sample= (p->Compute)(p->Par, S_T,
```

```
 S_max);
    if(pad==s)
      delta_sample=price_sample/s;
    else
      {
  if(log_pad>max_log_norm)
    delta_sample=-S_T/s;
  else  delta_sample=price_sample/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= exp(-r*t)*(mean_price/(double) N);
*pterror_price= sqrt(exp(-2.0*r*t)*var_pric
e/(double)N - SQR(*ptprice))/sqrt(N-1);

/* Price Confidence Interval */
*inf_price= *ptprice - z_alpha*(*pterror_pri
ce);
*sup_price= *ptprice + z_alpha*(*pterror_pri
ce);

/* Delta */
*ptdelta=exp(-r*t)*mean_delta/(double) N;
*pterror_delta=sqrt(exp(-2.0*r*t)*(var_delt
a/(double)N-SQR(*ptdelta)))/sqrt((double)N-1);

/* Delta Confidence Interval */
*inf_delta= *ptdelta - z_alpha*(*pterror_del
ta);
*sup_delta= *ptdelta + z_alpha*(*pterror_del
```

```
  ta);
}

    /* QMC simulation */
    else
{
  for(i=1; i<=N; i++)
    /* Begin N iterations */
    {
      /* In QMC simulation, to generate two
  independent variables,
   a gaussian one and a uniform one, we have
  to use a two-dimensional
   low-discrepancy sequence. Then we call th
  e function 'D-uniform'
   with parameter simulation_dim=2.*/
      /* First variable  transformed into a g
  aussian */
      gs= Inverse_erf(D_Uniform(simulation_dim
  , CREATE, 0, generator, mc_or_qmc));

      /* Second variable, uniform to generate
  the maximum */
      un= D_Uniform(simulation_dim, RETRIEVE,
  1, generator, mc_or_qmc);

      exp_sigmaxwt= exp(sigma_sqrt*gs);
      S_T= forward_stock*exp_sigmaxwt;

      max_log_norm=inverse_max(log_s, log(S_T)
  , t, sigma, un);
      S_max= exp(MAX(log(pad),max_log_norm));

      /* Price and Delta */
      /* CallFixedEuro */
      if (p->Compute == &Call_OverSpot2)
  {
    price_sample= (p->Compute)(p->Par, strik
  e, S_max);
    delta_sample= 0;
    if(pad==s)
```

```
        delta_sample=S_max/s;
      else
        {
          if(log_pad>max_log_norm)
      delta_sample=0.;
          else  delta_sample=S_max/s;
        }
    }
        else
    /* PutFloatingEuro */
    if (p->Compute == &Put_StrikeSpot2)
      {
        price_sample= (p->Compute)(p->Par, S_T,
     S_max);
        if(pad==s)
          delta_sample=price_sample/s;
        else
          {
      if(log_pad>max_log_norm)
        delta_sample=-S_T/s;
      else  delta_sample=price_sample/s;
          }
      }

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

      }
  }

    /* End N iterations */

    /* Price */
    *ptprice= exp(-r*t)*(mean_price/(double) N);

    /* Delta */
    *ptdelta=exp(-r*t)*mean_delta/(double) N;
  }
```

```
  return init_mc;
}



int CALC(MC_LookBackMax_Andersen)(void *Opt, void
     *Mod, PricingMethod *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 LookBackSup_AndersenMontecarlo(ptMod->S0
    .Val.V_PDOUBLE,
          (ptOpt->PathDep.Val.V_NUMFUNC_2)->
    Par[4].Val.V_PDOUBLE,
          (ptOpt->PayOff.Val.V_NUMFUNC_2)->
    Par[0].Val.V_PDOUBLE,
          ptOpt->PayOff.Val.V_NUMFUNC_2,
          ptOpt->Maturity.Val.V_DATE-ptMod->
    T.Val.V_DATE,
          r,
          divid,
          ptMod->Sigma.Val.V_PDOUBLE,
          Met->Par[0].Val.V_LONG,
          Met->Par[1].Val.V_INT,
          Met->Par[2].Val.V_DOUBLE,
          &(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_LookBackMax_Andersen)(void *Opt,
    void *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"
    LookBackCallFixedEuro")==0) || (strcmp( ((Option*)Opt)->Name,"
    LookBackPutFloatingEuro")==0) )
    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_DOUBLE= 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_LookBackMax_Andersen)=
{
  "MC_LookBackMax_Andersen",
  {{"N iterations",LONG,100,ALLOW},
   {"RandomGenerator",GENER,100,ALLOW},
   {"Confidence Value",DOUBLE,100,ALLOW},
   {" ",END,0,FORBID}},
  CALC(MC_LookBackMax_Andersen),
  {{"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_LookBackMax_Andersen),
  CHK_ok,
  MET(Init)
};
```

References