```
   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 and Delta wit
    h
   a confidence interval (for MC only).  */


#include  "bs1d_pad.h"
#include "enums.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 confid
    ence, double *ptprice, double *ptdelta, double *pterror_pric
    e, 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_PNL_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, mea
    n_delta, var_price, var_delta;
  PnlVect *U = pnl_vect_create(0);
  double alpha, z_alpha;
```

```
/* Value to construct the confidence interval */
alpha= (1.- confidence)/2.;
z_alpha= pnl_inv_cdfnor(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 simulation */
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= pnl_rand_init(generator, simulation_dim, N);
/* Test after initialization for the generator */
if(init_mc != OK)
    return init_mc;

MC_OR_PNL_QMC= pnl_rand_or_quasi(generator);
/* We test if simulation is MC or PNL_QMC.
 This involves two parts in the program because simulat
  ion for random vector
 must be called from different functions */

/* MC simulation case */
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 pseudo random number generator without problem
  of independence*/
    pnl_vect_rand_uni_d (U, 2, 0, 1, generator);
    gs = pnl_inv_cdfnor(pnl_vect_get(U, 0));
    un = pnl_vect_get(U, 1);

    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, strike, 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;

      if (MC_OR_PNL_QMC == PNL_MC)
      {
          /*Sum of squares*/
          var_price+= SQR(price_sample);
          var_delta+= SQR(delta_sample);
      }
  } /* End N iterations */

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

  if (MC_OR_PNL_QMC == PNL_MC)
  {
      /* irrelevant if PNL_QMC ! */
      *pterror_price= sqrt(exp(-2.0*r*t)*var_price/(double)
    N - SQR(*ptprice))/sqrt(N-1);
      *inf_price= *ptprice - z_alpha*(*pterror_price);
      *sup_price= *ptprice + z_alpha*(*pterror_price);
      *pterror_delta=sqrt(exp(-2.0*r*t)*(var_delta/(double)
    N-SQR(*ptdelta)))/sqrt((double)N-1);
      *inf_delta= *ptdelta - z_alpha*(*pterror_delta);
      *sup_delta= *ptdelta + z_alpha*(*pterror_delta);
  }
  pnl_vect_free (&U);
  return OK;
}



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_PDO
    UBLE,
          (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_DA
    TE,
          r,
          divid,
          ptMod->Sigma.Val.V_PDOUBLE,
          Met->Par[0].Val.V_LONG,
          Met->Par[1].Val.V_ENUM.value,
          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));
}



static int CHK_OPT(MC_LookBackMax_Andersen)(void *Opt, voi
    d *Mod)
{
  if ((strcmp(((Option*)Opt)->Name,"LookBackCallFixedEuro")
    ==0) || (strcmp( ((Option*)Opt)->Name,"    LookBackPutFloatingEuro")==0) )
    return OK;
  return WRONG;
}
```

```
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  int type_generator;
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_LONG=10000;
      Met->Par[1].Val.V_ENUM.value=0;
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumRNGs;
      Met->Par[2].Val.V_DOUBLE= 0.95;
    }

  type_generator= Met->Par[1].Val.V_ENUM.value;

  if(pnl_rand_or_quasi(type_generator)==PNL_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",ENUM,{100},ALLOW},
   {"Confidence Value",DOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{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} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_LookBackMax_Andersen),
  CHK_ok,
  MET(Init)
};
```

References