```
   Help
#include  "bs1d_limdisc.h"
#include "enums.h"
#include "pnl/pnl_cdf.h"

#define EPSILON 0.001

static int MC_down_out(long p,double t,double k,double spo
    t_temps,double bar_inf,double r,double v,long Nb,int type_    generator,doub
    delta,double *pterror_price,double *pterror_delta, double *
    inf_price, double *sup_price, double *inf_delta, double *sup_
    delta)
{

  double s1=0.,s2=0.,s3=0.,s,s_2,s_3,d,d1,b=0.,b1=0.;

  double  ln_c,ln_c1,cte_deux1,cte_deux,h,h1,Nh,Nh1,uni,a,
    a1,
    ln_prod,ln_prod1,x,x1,ln_spot_temps,ln_spot_temps1,ln_
    bar_inf,
    t_j=t/(double)p,var_price,var_delta;

  long i,n,j=1;

  int simulation_dim= 1;
  int init_mc;
  double alpha, z_alpha;

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


  n=p;

  while (j<=temps/t_j)
    j=j+1;

  t_j=j*t_j;

  ln_spot_temps=log(spot_temps);
```

```
ln_spot_temps1=log(spot_temps+EPSILON);

ln_bar_inf=log(bar_inf);
ln_c=(r-(SQR(v))/2.)*t/(double)n;
ln_c1=(r-(SQR(v))/2.)*(t_j-temps);

d=v*sqrt(t/(double)n);
d1=v*sqrt(t_j-temps);

cte_deux=v*sqrt(t/(double)n);
cte_deux1=v*sqrt(t_j-temps);

n=0;


init_mc= pnl_rand_init(type_generator, simulation_dim,Nb)
  ;
if(init_mc == OK)
  {
    do
{
  n=n+1;
  i=1;

  h=(ln_bar_inf-ln_spot_temps-ln_c1)/cte_deux1;
  h1=(ln_bar_inf-ln_spot_temps1-ln_c1)/cte_deux1;

  Nh=cdf_nor(h);
  Nh1=cdf_nor(h1);

  uni=pnl_rand_uni(type_generator);

  a=pnl_inv_cdfnor(Nh+uni*(1.-Nh));
  a1=pnl_inv_cdfnor(Nh1+uni*(1.-Nh1));

  ln_prod=ln_spot_temps+ln_c1+d1*a;
  ln_prod1=ln_spot_temps1+ln_c1+d1*a1;

  x=1.-Nh;
  x1=1.-Nh1;
```

```
  while(i<=p-j)
    {
      h=(ln_bar_inf-ln_prod-ln_c)/cte_deux;
      h1=(ln_bar_inf-ln_prod1-ln_c)/cte_deux;

      Nh=cdf_nor(h);
      Nh1=cdf_nor(h1);

      uni=pnl_rand_uni(type_generator);

      a=pnl_inv_cdfnor(Nh+uni*(1.-Nh));
      a1=pnl_inv_cdfnor(Nh1+uni*(1.-Nh1));

      ln_prod=ln_c+d*a+ln_prod;
      ln_prod1=ln_c+d*a1+ln_prod1;

      x=(1.-Nh)*x;
      x1=(1.-Nh1)*x1;

      i=i+1;
    }

  s=MAX(exp(ln_prod)-k,0.)*x;
  s_2=MAX(exp(ln_prod1)-k,0.)*x1;
  s_3=(s_2-s)/EPSILON;

  s1=s1+s;
  s2=s2+s_2;
  s3=s3+s_3;

  b=b+SQR(s);
  b1=b1+SQR(s_3);

  var_price=(b/(double)n)-SQR(s1/(double)n);
  var_delta=(b1/(double)n)-SQR(s3/(double)n);

}
    while((n<=Nb));

    /* Price */
    *ptprice=exp(-r*(t-temps))*(s1/(double)Nb);
```

```
      *pterror_price=2.*sqrt(var_price/Nb);

      /*Delta*/
      *ptdelta=exp(-r*(t-temps))*(s3/(double)Nb);
      *pterror_delta=2.*sqrt(var_delta/Nb);

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

      /* Delta Confidence Interval */
      *inf_delta= *ptdelta - z_alpha*(*pterror_delta);
      *sup_delta= *ptdelta + z_alpha*(*pterror_delta);
    }
  return init_mc;

}

int CALC(MC_VarianceReduction)(void*Opt,void *Mod,Pricing
    Method *Met)
{
  TYPEOPT* ptOpt=( TYPEOPT*)Opt;
  TYPEMOD* ptMod=( TYPEMOD*)Mod;
  double r;
  int return_value;

  r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
  if(ptMod->Divid.Val.V_DOUBLE>0)
    {
      Fprintf(TOSCREEN,"Divid >0, untreated case{{n{n{n");
      return_value = WRONG;
    }
  else
    return_value=MC_down_out((ptOpt->Limit.Val.V_NUMFUNC_1)
    ->Par[2].Val.V_INT2/*ptOpt->NumberDate.Val.V_INT*/,
           ptOpt->Maturity.Val.V_DATE,
           (ptOpt->PayOff.Val.V_NUMFUNC_1)->Par[0].Val.
    V_PDOUBLE,
           ptMod->S0.Val.V_PDOUBLE,
           ((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((pt
    Opt->Limit.Val.V_NUMFUNC_1)->Par, ptMod->T.Val.V_DATE),
```

```
            r,
            ptMod->Sigma.Val.V_PDOUBLE,
            Met->Par[0].Val.V_LONG,
            Met->Par[1].Val.V_ENUM.value,
            ptMod->T.Val.V_DATE,
            Met->Par[2].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));

  return return_value;
}



static int CHK_OPT(MC_VarianceReduction)(void *Opt, void *
    Mod)
{
  return strcmp( ((Option*)Opt)->Name,"CallDownOutDiscEuro"
    );
}



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=&PremiaEnumMCRNGs;
      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_VarianceReduction)=
{
  "MC_VarianceReduction",
  {{"N iterations",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Confidence Value",DOUBLE,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_VarianceReduction),
```

```
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {"PriceError",DOUBLE,{100},FORBID},
   {"DeltaError",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_VarianceReduction),
  CHK_mc,
  MET(Init)
} ;
```

References