```
   Help
// Written by P. Tankov and J. Poirot, June-September 2006
// This file is part of PREMIA software copying and usage
    restrictions apply

extern "C"{
#include "cgmy1d_std.h"
#include "error_msg.h"
#include "enums.h"
#include "pnl/pnl_cdf.h"
}
#include <cmath>
#include "math/cgmy/cgmy.h"
#include "math/cgmy/rnd.h"

extern "C"{

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_TankovPoirot_CGMY)(void *Opt, void *
    Mod)
{
  return NONACTIVE;
}
int CALC(MC_TankovPoirot_CGMY)(void*Opt,void *Mod,Pricing
    Method *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

  // Pricing a european put option on a stock driven by
    CGMY process
  // By Monte Carlo using the algorithm by Poirot and Tank
    ov (2006)
  // Input parameters
  // T          : option maturity
  // S0         : initial stock price
  // r          : interest rate
  // q          : dividend yield
```

```
// K          : strike
// type       : use 1 for call, any other value for put
// C, G, M, Y : process parameters
// Ntraj      : number of Monte Carlo simulations
// Output values
// price, delta, and the standard deviations of MC estimates
// return value: zero if success, nonzero if error
// 1 is returned if alphap or alphan is equal to 1 (this
  case is not supported)
static int MonteCarlo_TankovPoirotCGMY(double S0,NumFunc_
  1  *p,double T,double r,double divid,double C,double G,
  double M,double Y,long Ntraj,int generator,double confidence,
  double *ptprice,double *ptdelta,double *inf_price, double *sup_
  price, double *inf_delta, double *sup_delta)
{
  double K;
  int type;
  double price,delta,stdprice,stddelta;
  double alphap,alphan,lambdap,lambdan,cp,cn;
  int simulation_dim= 1;
  int init_mc;
   double alpha, z_alpha;

  alphap=Y;
  alphan=Y;
  lambdap=M;
  lambdan=G;
  cp=C;
  cn=C;
  if((alphap==1.)||(alphan==1.)) return BAD_ALPHA_TEMPSTA
  BLE;

  K=p->Par[0].Val.V_DOUBLE;
  if ((p->Compute)==&Put)
    type=0;
  else
    type=1;

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

```
/*MC sampling*/
init_mc= pnl_rand_init(generator,simulation_dim,Ntraj);
if(init_mc == OK)
  {
    price = 0; stdprice = 0;
    delta = 0; stddelta = 0;
    if((alphap==1)||(alphan==1)) return 1;
    double gcp = -tgamma(2.-alphap)/alphap/(alphap-1)*
pow(lambdap,alphap) * cp*(pow(1.-1./lambdap,alphap)-1.+alp
hap/lambdap);
    double gcn = -tgamma(2.-alphan)/alphan/(alphan-1)*
pow(lambdan,alphan) * cn*(pow(1.+1./lambdan,alphan)-1.-alp
han/lambdan);
    double c = -tgamma(2.-alphap)/alphap/(alphap-1)*po
w(lambdap,alphap) * cp*(alphap-1)-tgamma(2.-alphan)/alphan/
(alphan-1)*pow(lambdan,alphan) * cn*(alphan-1)+lambdan*gc
n-lambdap*gcp;
    double sigmap = pow(-cp*T*tgamma(2.-alphap)/alphap/
(alphap-1)*cos(M_PI*alphap/2),1./alphap);
    double sigman = pow(-cn*T*tgamma(2.-alphan)/alphan/
(alphan-1)*cos(M_PI*alphan/2),1./alphan);
    double mup = gcp*T - cp*T*tgamma(2.-alphap)/(1.-alp
hap)*pow(lambdap,alphap-1);
    double mun = gcn*T + cn*T*tgamma(2.-alphan)/(1.-alp
han)*pow(lambdan,alphan-1);
    /*double stdconst = exp(tgamma(2.-alphap)/alphap/(
alphap-1)*pow(lambdap,alphap) * cp*T*(pow(2.,alphap-1)-1)+tg
amma(2.-alphan)/alphan/(alphan-1)*pow(lambdan,alphan) * cn*
T*(pow(2.,alphan-1)-1));*/
    double XTP, XTN, XT, WT;
    /*double m = log(K/S0)-(r-divid)*T;*/
    /*double R;*/
    StableRnd Pos(alphap,sigmap,1,mup,generator);
    StableRnd Neg(alphan,sigman,-1,mun,generator);
    for(long i=0; i<Ntraj; i++){
      XTP = Pos.next();
      XTN = Neg.next();
      XT = XTP+XTN;
      WT = exp(-lambdap*XTP+lambdan*XTN-c*T);
      double payoff = (K*exp(-r*T)-S0*exp(-divid*T+XT))
```

```
  *WT;
        if(payoff>0) {
          price+=(payoff/Ntraj);
          stdprice+=(payoff*payoff/Ntraj);
          delta-=(exp(-divid*T+XT)*WT/Ntraj);
          stddelta+=(exp(-2*divid*T+2*XT)*WT*WT/Ntraj);
        }
      }
      stdprice=sqrt((1./(Ntraj-1))*(stdprice-price*price)
  );
      stddelta=sqrt((1./(Ntraj-1))*(stddelta-delta*delta)
  );
      if(type==1) {
        price += S0*exp(-divid*T)-K*exp(-r*T);
        delta += exp(-divid*T);
      }

      *ptprice=price;
      *ptdelta=delta;


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

    /* Delta Confidence Interval */
    *inf_delta= *ptdelta - z_alpha*(stddelta);
    *sup_delta= *ptdelta + z_alpha*(stddelta);

    }
  return OK;

}

int CALC(MC_TankovPoirot_CGMY)(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  MonteCarlo_TankovPoirotCGMY(ptMod->S0.Val.V_PDO
    UBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->Maturity.Val.V_DA
    TE-ptMod->T.Val.V_DATE,r,divid, ptMod->C.Val.V_PDOUBLE,pt
    Mod->G.Val.V_PDOUBLE,ptMod->M.Val.V_PDOUBLE,ptMod->Y.Val.V_
    PDOUBLE,Met->Par[0].Val.V_LONG,Met->Par[1].Val.V_ENUM.value,
    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));
  }

static int CHK_OPT(MC_TankovPoirot_CGMY)(void *Opt, void *
    Mod)
  {
    if ( (strcmp( ((Option*)Opt)->Name,"CallEuro")==0) || (
    strcmp( ((Option*)Opt)->Name,"PutEuro")==0) )
      return OK;

    return WRONG;
  }

#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met,Option *Opt)
  {
    static int first=1;
    if (first)
      {
        Met->Par[0].Val.V_LONG=10000000;
        Met->Par[1].Val.V_ENUM.value=0;
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[2].Val.V_PDOUBLE= 0.95;

        first=0;
      }
    return OK;
  }

  PricingMethod MET(MC_TankovPoirot_CGMY)=
```

```
  {
    "MC_TankovPoirot_CGMY",
    {{"N iterations",LONG,{100},ALLOW},
     {"RandomGenerator",ENUM,{100},ALLOW},
      {"Confidence Value",DOUBLE,{100},ALLOW},
     {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(MC_TankovPoirot_CGMY),
    {{"Price",DOUBLE,{100},FORBID},
     {"Delta",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_TankovPoirot_CGMY),
   CHK_mc,
    MET(Init)
  } ;
}
```

References