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 "
href../../../mod/cgmy1d/cgmy1d_std/cgmy1d_std_h_src.pdfcgmy1d_std.h"
#include "
href../../../common/enums_h_src.pdfenums.h"
#include "pnl/pnl_cdf.h"
}
#include <cmath>
#include "
href../../../common/math/cgmy/cgmy_h_src.pdfmath/cgmy/cgmy.h"
#include "
href../../../common/math/cgmy/rnd_h_src.pdfmath/cgmy/rnd.h"


// Pricing a european option on a CGMY-driven stock
// By Monte Carlo using the simulation algorithm by Madan and Yor (2005)
// 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
// eps        : jumps smaller than eps are truncated in the Madan-Yor algorithm
// Ntraj      : number of Monte Carlo simulations
// Output values
// price, delta, and the standard deviations of MC estimates
// return value: always zero
// possible default parameter values:
// C = 0.5, Y = 0.5, G = 2, M = 3.5, eps = 0.0001, Ntraj = 100000;

extern "C" {

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
  static int CHK_OPT(MC_MadanYor_CGMY)(void *Opt, void *Mod)
```

```
  {
    return NONACTIVE;
  }
  int CALC(MC_MadanYor_CGMY)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else

  static int MonteCarlo_MadanYorCGMY(double S0, NumFunc_1  *p, double T, double
  {
    double K;
    double price, delta, stdprice, stddelta;
    int init_mc;
    double alpha, z_alpha;

    K = p->Par[0].Val.V_DOUBLE;

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

    /* MC init */
    init_mc = pnl_rand_init(generator, 1, Ntraj);
    if (init_mc != OK) return init_mc;

    price = 0;
    stdprice = 0;
    delta = 0;
    stddelta = 0;
    CGMYSimulator csim(C, G, M, Y, eps_cm, generator);
    double gam = csim.gamma_mart();
    for (int i = 0; i < Ntraj; i++)
      {
        double a = csim.sim(T);
        double payoff = K * exp(-r * T) - S0 * exp(-q * T + gam * T + a);
        if (payoff > 0)
          {
            price += payoff;
            stdprice += payoff * payoff;
            delta -= exp(-q * T + gam * T + a);
```

```
          stddelta += exp(-2 * q * T + 2 * gam * T + 2 * a);
        }
    }

  price /= Ntraj;
  stdprice /= Ntraj;
  delta /= Ntraj;
  stddelta /= Ntraj;

  stdprice = sqrt((1. / (Ntraj - 1)) * (stdprice - price * price));
  stddelta = sqrt((1. / (Ntraj - 1)) * (stddelta - delta * delta));
  if ((p->Compute) == &Call)
    {
      price += S0 * exp(-q * T) - K * exp(-r * T);
      delta += exp(-q * 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_MadanYor_CGMY)(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  MonteCarlo_MadanYorCGMY(ptMod->S0.Val.V_PDOUBLE,
                                    ptOpt->PayOff.Val.V_NUMFUNC_1,
                                    ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_
                                    r, divid,  ptMod->C.Val.V_PDOUBLE,
                                    ptMod->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->Par[3].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_MadanYor_CGMY)(void *Opt, void *Mod)
  {
    if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)O
      return OK;

    return FAIL;
  }

#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
    static int first = 1;
    if (first)
      {
        Met->Par[0].Val.V_LONG = 50000;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[2].Val.V_PDOUBLE = 0.95;
        Met->Par[3].Val.V_PDOUBLE = 0.0001;

        first = 0;
```

```
    }
  return OK;
}

PricingMethod MET(MC_MadanYor_CGMY) =
{
  "MC_Madan_Yor",
  { {"N iterations", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Confidence Value", DOUBLE, {100}, ALLOW},
    {"Eps (jumps smaller than eps are truncated)", DOUBLE, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_MadanYor_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_MadanYor_CGMY),
  CHK_mc,
  MET(Init)
} ;
}
```