

## [Help](#)

```
#include "
href../../mod/cgmy1d/cgmy1d_std/cgmy1d_std_h_src.pdfcgmy1d_std.h"
#include "pnl/pnl_cdf.h"
#include"pnl/pnl_random.h"
#include"pnl/pnl_specfun.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2014+2) //The "#els
static int CHK_OPT(MC_TwoLevel_CGMY)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_TwoLevel_CGMY)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

/* Compute the positive or negative jump size between the smallest and the bigger
static double jump_generator_CGMY(double C, double M_G, double Y, int generator,
{
    double U2, U1, Z;
    double pui = 1 / Y;

    U1 = pnl_rand_uni(generator);
    U2 = pnl_rand_uni(generator);
    Z = eps / (pow(U1, pui));
    if (U2 > exp(-M_G * Z))
    {
        Z = 0;
    }

    return Z;
}

static double jump_generator_CGMY_tilde(double C, double M_G, double Y, int gene
{
    double U2 = 2, U1, Z = 0;
    double pui = 1 / Y;
```

```

    U1 = pnl_rand_uni(generator);
    U2 = pnl_rand_uni(generator);
    Z = pow(pow(eps, -Y) - U1 * (pow(eps, -Y) - pow(eps_beta, -Y)), -pui);
    if (U2 > exp(-M_G * Z))
    {
        Z = 0;
    }
    return Z;
}
//Compute the function sigma(epsilon)

static double sigm(double C, double Y, double eps)
{
    double Z;
    Z = sqrt(2 * C / (2 - Y)) * pow(eps, 1 - Y / 2);
    return Z;
}

//Compute the function kappa

static double kappa(double C, double G, double M, double Y, double u)
{
    double Z;
    if (Y > 1)
    {
        Z = C * pnl_sf_gamma(-Y) * (pow(M, Y) * (pow(1 - u / M, Y) - 1 + (u * Y /
    }
    if (Y < 1)
    {
        Z = C * pnl_sf_gamma(-Y) * (pow(M, Y) * (pow(1 - u / M, Y) - 1) + pow(G, Y
    }
    return Z;
}

/* Calculus of the optimal parameter which minimizes the variance by using the C
static void Simul_theta(double eps, int type_generator, double Strike, double S0
{
    int RM = 100000, x; /* number of iterations of Robbins-Monro algorithm */
    long ii, k;
    double a = 1, b = 1, payoffcarre1, payoffcarre2, temp1, temp2, expo1, expo2;
    double drift1, drift12, drift, lambda_p, lambda_m, Xp, Xm, z_p, z_m;

```

```

double gamma_step;
double jump_number_p, jump_number_m, gradient_p1, gradient_p2;
PnlVect *m_Mu1, *m_Mu2;

m_Mu1 = pnl_vect_create_from_double(RM + 1, 0);
m_Mu2 = pnl_vect_create_from_double(RM + 1, 0);

*theta1 = 0;
*theta2 = 0;
jump_number_p = 0;
jump_number_m = 0;

lambda_p = C * pow(eps, -Y) / Y ; /* positive jump intensity with epsilon^beta
lambda_m = C * pow(eps, -Y) / Y ; /* negative jump intensity with epsilon^beta

pnl_rand_init(type_generator, 1, 100000);

for (ii = 1; ii < RM; ii++)
{
//simulation of the positive jump number
jump_number_p = pnl_rand_poisson1(lambda_p, T, type_generator);
//simulation of the negative jump number
jump_number_m = pnl_rand_poisson1(lambda_m, T, type_generator);
//computation of Xg and Xd
Xp = 0;
Xm = 0;
if (jump_number_p > 0)
{
for (k = 1; k <= jump_number_p; k++)
{
z_p = jump_generator_CGMY(C, M, Y, type_generator, eps);
Xp = Xp + z_p;
}
}

if (jump_number_m > 0)
{

```

```

    for (k = 1; k <= jump_number_m; k++)
    {
        z_m = jump_generator_CGMY(C, G, Y, type_generator, eps);
        Xm = Xm + z_m;
    }
}

/* c'est le deuxième terme dans les deux poids de Girsanov pour chaque MC
drift1 = (r - divid - kappa(C, G, M, Y, 1)) * GET(m_Mu1, ii) + kappa(C,
drift12 = (r - divid - kappa(C, G, M, Y, 1)) * GET(m_Mu2, ii) + kappa(C,
/* le premier terme correspond à la martingale et le deuxième c'est une co

if (Y > 1)
{
    drift = (r - divid - kappa(C, G, M, Y, 1)) - C * (pow(M, Y - 1) * pn

}

if (Y < 1)
{
    drift = (r - divid - kappa(C, G, M, Y, 1));
}

if (S0 * exp(Xp - Xm + drift * T) > Strike)
{
    x = 1;
}
else
{
    x = 0;
}

payoffcarre1 = SQR(MAX(S0 * exp(Xp - Xm + drift * T) - Strike, 0)); /* Pay
payoffcarre2 = T * x * SQR(S0 * exp(Xp - Xm + drift * T)); /* Payoff of th

if (Y > 1)
{
    gradient_p1 = (r - divid) - kappa(C, G, M, Y, 1) + C * pnl_sf_gamma(
/* Gradient de kappa_+theta */

    gradient_p2 = (r - divid) - kappa(C, G, M, Y, 1) + C * pnl_sf_gamma(
/* Gradient de kappa_+theta */

```

```

    }

    if (Y < 1)
    {
        gradient_p1 = (r - divid) - kappa(C, G, M, Y, 1) + C * pnl_sf_gamma(
        /* Gradient de kappa_+theta */

        gradient_p2 = (r - divid) - kappa(C, G, M, Y, 1) + C * pnl_sf_gamma(
        /* Gradient de kappa_+theta */
    }

    expo1 = exp(-GET(m_Mu1, ii) * (Xp - Xm + drift * T) + drift1 * T);
    expo2 = exp(-GET(m_Mu2, ii) * (Xp - Xm + drift * T) + drift12 * T);

    temp1 = (gradient_p1 * T - (Xp - Xm + drift * T)) * expo1 * payoffcarre1;
    temp2 = (gradient_p2 * T - (Xp - Xm + drift * T)) * expo2 * payoffcarre2;

    gamma_step = a / (b + ii);

    /* if the (n+1)th estimator of theta is in the right interval [-G,M]*/
    LET(m_Mu1, ii + 1) = GET(m_Mu1, ii) - gamma_step * temp1;
    LET(m_Mu2, ii + 1) = GET(m_Mu2, ii) - gamma_step * temp2;

    if ((GET(m_Mu1, ii + 1) > M) || (GET(m_Mu1, ii + 1) < -G))
    {
        LET(m_Mu1, ii + 1) = 0;
    }

    if ((GET(m_Mu2, ii + 1) > M) || (GET(m_Mu2, ii + 1) < -G))
    {
        LET(m_Mu2, ii + 1) = 0;
    }

}

*theta1 = GET(m_Mu1, RM);
*theta2 = GET(m_Mu2, RM);

```

```

    pnl_vect_free(&m_Mu1);
    pnl_vect_free(&m_Mu2);

    return;
}

static int MonteCarlo_TwoLevelCGMY(double S0, NumFunc_1 *p, double T, double r,
{
    long i, k;
    double eps_beta, drift1, drift12, drift_beta, drift_beta2, drift, drift2;
    double lambda_p_beta, lambda_m_beta, lambda_p_tilde, lambda_m_tilde;
    double Xp, Xm, Xp_beta, Xm_beta, Xp_tilde, Xm_tilde;
    int jump_number_p_beta, jump_number_m_beta, jump_number_p_tilde, jump_number_m_tilde;
    double price_sample1, mean_price1, var_price1, price_sample2, mean_price2, var_price2;
    double square_ptprice1, square_ptprice2;
    double theta1, theta2;
    double beta;
    double puiss, puiss2;
    int n_paths1, n_paths2;
    double K;
    double alpha, z_alpha;

    K = p->Par[0].Val.V_DOUBLE;
    beta = Y / 2.;
    eps_beta = pow(eps, beta);
    puiss = 2 * (2 - Y);
    puiss2 = beta * (2 - Y);
    n_paths1 = floor(SQR((2 - Y) / (2 * C * T)) / pow(eps, puiss));
    n_paths2 = floor((2 * (2 - Y) / (C * T)) * ((pow(eps, puiss2) - pow(eps, 2 - Y)) / (2 - Y)));

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

    Simul_theta(eps, generator, K, S0, T, r, divid, M, G, Y, C, &theta1, &theta2);

    theta1 = 0; //5.3;//5.7;
    theta2 = 0; //2.5;//0.4;

    lambda_p_beta = C * pow(eps_beta, -Y) / Y; /* positive jump intensity with eps */
    lambda_m_beta = C * pow(eps_beta, -Y) / Y; /* negative jump intensity with eps */

```

```

lambda_p_tilde = C * (pow(eps, -Y) - pow(eps_beta, -Y)) / Y; /* positive jump
lambda_m_tilde = C * (pow(eps, -Y) - pow(eps_beta, -Y)) / Y; /* negative jump

/* c'est le deuxième terme dans les deux poids de Girsanov pour chaque MC */
drift1 = (r - divid - kappa(C, G, M, Y, 1)) * theta1 + kappa(C, G, M, Y, t
drift12 = (r - divid - kappa(C, G, M, Y, 1)) * theta2 + kappa(C, G, M, Y,

if (Y > 1)
{
    /* le premier terme correspond à la martingale et le deuxième c'est une co
    drift = (r - divid - kappa(C, G, M, Y, 1)) - C * (pow(M, Y - 1) * pnl_sf

    drift2 = (r - divid - kappa(C, G, M, Y, 1)) - C * (pow(M, Y - 1) * pnl_s

    /* le premier terme correspond à la martingale et le deuxième c'est une co
    drift_beta = (r - divid - kappa(C, G, M, Y, 1)) - C * (pow(M, Y - 1) * p

    drift_beta2 = (r - divid - kappa(C, G, M, Y, 1)) - C * (pow(M, Y - 1) *
}

if (Y < 1)
{
    drift = (r - divid - kappa(C, G, M, Y, 1));
}
/* generator initialisation */
pnl_rand_init(generator, 1, 10000000);
//pnl_rand_init(generator,1,n_paths2);

/* Initialisation */

mean_price1 = 0;
var_price1 = 0;

/* First Monte Carlo Sampling n_paths1 */

for (i = 0; i < n_paths1; i++)
{

```

```

/* Begin of the n_paths1 iterations */

/* simulation of the positive jump number */
jump_number_p_beta = pnl_rand_poisson1(lambda_p_beta, T, generator);
/* simulation of the negative jump number */
jump_number_m_beta = pnl_rand_poisson1(lambda_m_beta, T, generator);

/* computation of Xp and Xm */
Xp = 0;
Xm = 0;
if (jump_number_p_beta > 0)
{
    for (k = 1; k <= jump_number_p_beta; k++)
    {
        Xp += jump_generator_CGMY(C, M - theta1, Y, generator, eps_beta);
    }
}

if (jump_number_m_beta > 0)
{
    for (k = 1; k <= jump_number_m_beta; k++)
    {
        Xm += jump_generator_CGMY(C, G + theta1, Y, generator, eps_beta);
    }
}

/* Call Price */
if (Y > 1)
{
    price_sample1 = exp(-r * T) * MAX(S0 * exp(Xp - Xm + drift_beta * T) -
}

if (Y < 1)
{
    price_sample1 = exp(-r * T) * MAX(S0 * exp(Xp - Xm + drift * T) - K, 0
}

/* mean */
mean_price1 += price_sample1 / n_paths1;

/* variance */
var_price1 += SQR(price_sample1);
}

```



```

square_ptprice1 = var_price1 / n_paths1;
variance1 = square_ptprice1 - SQR(mean_price1);
/* End of the n_paths1 iterations */

/* Initialisation */
mean_price2 = 0;
var_price2 = 0;

/*Second Monte Carlo sampling n_paths2 */
for (i = 0; i < n_paths2; i++)
{
    /* Begin of the n_paths2 iterations */

    /* simulation of the positive jump number */
    jump_number_p_beta = pnl_rand_poisson1(lambda_p_beta, T, generator);
    /* simulation of the negative jump number */
    jump_number_m_beta = pnl_rand_poisson1(lambda_m_beta, T, generator);
    /* computation of Xp_beta and Xm_beta */
    Xp_beta = 0;
    Xm_beta = 0;

    if (jump_number_p_beta > 0)
    {
        for (k = 1; k <= jump_number_p_beta; k++)
        {
            Xp_beta += jump_generator_CGMY(C, M - theta2, Y, generator, eps_be
        }
    }

    if (jump_number_m_beta > 0)
    {
        for (k = 1; k <= jump_number_m_beta; k++)
        {
            Xm_beta += jump_generator_CGMY(C, G + theta2, Y, generator, eps_be
        }
    }

    /* simulation of the positive jump number */
    jump_number_p_tilde = pnl_rand_poisson1(lambda_p_tilde, T, generator);
    /* simulation of the negative jump number */

```

```

jump_number_m_tilde = pnl_rand_poisson1(lambda_m_tilde, T, generator);
/* computation of Xp_tilde and Xm_tilde */
Xp_tilde = 0;
Xm_tilde = 0;

if (jump_number_p_tilde > 0)
{
    for (k = 1; k <= jump_number_p_tilde; k++)
    {
        Xp_tilde += jump_generator_CGM_tilde(C, M - theta2, Y, generator,
    }
}
if (jump_number_m_tilde > 0)
{
    for (k = 1; k <= jump_number_m_tilde; k++)
    {
        Xm_tilde += jump_generator_CGM_tilde(C, G + theta2, Y, generator,
    }

}
/* Call Price of the difference */
if (Y > 1)
{
    price_sample2 = exp(-r * T) * (MAX(S0 * exp(Xp_beta - Xm_beta + Xp_til
}

if (Y < 1)
{
    price_sample2 = exp(-r * T) * (MAX(S0 * exp(Xp_beta - Xm_beta + Xp_til
}

/* mean */
mean_price2 += price_sample2 / n_paths2;
/* variance */
var_price2 += SQR(price_sample2);
}

/* End of the n_paths2 iterations */

```

```

/* Statistical Romberg Price */

square_ptprice2 = var_price2 / n_paths2;

*ptprice = mean_price1 + mean_price2; /* European price using the Statistical

variance2 = square_ptprice2 - SQR(mean_price2);
pterror_price = variance1 + variance2;

/* Price Confidence Interval */
*inf_price = *ptprice - z_alpha * sqrt(pterror_price / (double)n_paths1);
*sup_price = *ptprice + z_alpha * sqrt(pterror_price / (double)n_paths1);

return OK;

}

int CALC(MC_TwoLevel_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_TwoLevelCGMY(ptMod->S0.Val.V_PDOUBLE,
                                    ptOpt->PayOff.Val.V_NUMFUNC_1,
                                    ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                                    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_ENUM.value,
                                    Met->Par[1].Val.V_PDOUBLE,
                                    Met->Par[2].Val.V_PDOUBLE,

```

```

        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE),
        &(Met->Res[2].Val.V_DOUBLE));
    }

static int CHK_OPT(MC_TwoLevel_CGMY)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0))
        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_ENUM.value = 0;
        Met->Par[0].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[1].Val.V_PDOUBLE = 0.95;
        Met->Par[2].Val.V_PDOUBLE = 0.001;
        Met->HelpFilenameHint = "mc_twolevel_cgmy";

        first = 0;
    }
    return OK;
}

PricingMethod MET(MC_TwoLevel_CGMY) =
{
    "MC_TwoLevel",
    { {"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_TwoLevel_CGMY),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Inf Price", DOUBLE, {100}, FORBID},
    }
}

```

```

        {"Sup Price", DOUBLE, {100}, FORBID} ,
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_TwoLevel_CGMY),
    CHK_mc,
    MET(Init)
} ;

```