

## [Help](#)

```
#include "
href../../../../mod/lmm1d_cgmy/lmm1d_cgmy_stdh/lmm1d_cgmy_stdh_src.pdf
#include "
href../../../../common/enums_h_src.pdfenums.h"
#include"pnl/pnl_vector.h"
#include"pnl/pnl_random.h"
#include"pnl/pnl_specfun.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_integration.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2012+2) //The "#els
static int CHK_OPT(MC_LOGLEVY_SWAPTION)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_LOGLEVY_SWAPTION)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

// The logarithme of the Laplace transform of CGMY at time 1
static double kappaCGMY(double C_1, double G_1, double M_1, double Y_1, double u
{
    return (C_1 * pnl_sf_gamma(-Y_1) * (
        pow(M_1, Y_1) * (pow(1.0 - u_1 / M_1, Y_1) - 1.0 + u_1 * Y_1 / M_1)
        pow(G_1, Y_1) * (pow(1.0 + u_1 / G_1, Y_1) - 1.0 - u_1 * Y_1 / G_1))
}

//drift of the Libor rate , 1rst order expansion (cf b1_i = \ theta_i - \ sum_j
static double b1(PnlMat *Log_L1, int i_1, int k_1, double de_1, int N_1, PnlVec
{
    int j_1;

    double b_1 = -pnl_vect_get(theta_1, i_1);

    for (j_1 = i_1 + 1; j_1 < N_1; j_1++)
```

```

        b_l -= de_l * exp(pnl_mat_get(Log_Ll, j_l, k_l)) / (1.0 + de_l * exp(pnl_mat

    return b_l;
}

//drift of the Libor rate , 2nd order expansion (b2_i = b1_i - \sum_kl ... see
static double b2(PnlMat *Log_Ll, int i_l, int k_l, double de_l, int N_l, PnlVect
{
    int j_l, l_l;

    double b_l = b1(Log_Ll, i_l, k_l, de_l, N_l, la_l, theta_l, eta_l) ;
    for (j_l = i_l + 1; j_l < N_l; j_l++)
        for (l_l = j_l + 1; l_l < N_l; l_l++)
            b_l -= de_l * de_l * exp(pnl_mat_get(Log_Ll, j_l, k_l)) * exp(pnl_mat_get(

    return b_l;
}

//drift using the auxillary variables Z in the first order log_levy approximation
static double b1Z(PnlMat *Z, int i_l, int k_l, double de_l, int N_l, PnlVect *la
{
    int j_l;
    double b_l = -pnl_vect_get(theta_l, i_l);

    for (j_l = i_l + 1; j_l < N_l; j_l++)
        b_l -= pnl_mat_get(Z, j_l, k_l) * pnl_mat_get(eta_l, i_l, j_l);
    return b_l;
}

/* static double SmallJumpDrift(double C_l,double G_l,double M_l,double Y_l, dou
/* { */
/* double result = 0.0; */
/* //int n = 1000; */
/* int i; */
/* for(i=1;i<n;i++) */
/* result -= C_l*i*Epsilon/(double)n * exp(-G_l*i*Epsilon/(double)n)*exp( - (1.0
/* for(i=1;i<n;i++) */
/* result += C_l*i*Epsilon/(double)n * exp(-M_l*i*Epsilon/(double)n)*exp( - (1.0
/* return result; */

```

```
/* } */
```

```
static double BigJumpDrift(double C_1, double G_1, double M_1, double Y_1, double n)
{
    double result = 0.0;
    double Uper = 4.0 / MIN(G_1, M_1);
    double Lower = Epsilon;
    double x;
    int N = 20000;
    //int n = 1000;
    int i;
    for (i = 1; i < n; i++)
    {
        x = Lower + (Uper - Lower) * i / (double)n;
        result -= C_1 * x * exp(-G_1 * x) * exp(-(1.0 + Y_1) * log(x)) * (Uper - Lower);
        result += C_1 * x * exp(-M_1 * x) * exp(-(1.0 + Y_1) * log(x)) * (Uper - Lower);
    }

    Lower = 4.0 / MIN(G_1, M_1);
    Uper = 25.0 / MIN(G_1, M_1);
    for (i = 0; i < N; i++)
    {
        x = Lower + (Uper - Lower) * i / (double)N;
        result -= C_1 * x * exp(-G_1 * x) * exp(-(1.0 + Y_1) * log(x)) * (Uper - Lower);
        result += C_1 * x * exp(-M_1 * x) * exp(-(1.0 + Y_1) * log(x)) * (Uper - Lower);
    }

    return result;
}
```

```
/* static void calculateSwaptionPrice(double C_1,double G_1,double M_1,double Y_1,
/* { */
/* double Uniform1,Uniform2, pareto, pareto_p, pareto_n, binomial, dH,bigJumpDrift */
/* int i,j,k,p,l, m_0, m; */
/* double intensity, Payoff, Payoffvar; */
/* double sum_plus, Payoff_receiv, var_receiv ; */
/* int numJumps,indexSwapMat; */
```

```

/* double zeta[N_l][N_l][N_l]; */

/* PnlVect *theta, *std, *c; */
/* PnlMat *Log_L,*eta, *Discount; */

/* dt_l = de_l/(double)m_l; */

/* m_0 = (int) pnl_vect_get(T_l,0)/dt_l + 1; */
/* dt_0 = pnl_vect_get(T_l,0)/(double)m_0; */

/* Log_L = pnl_mat_create_from_double(N_l,n_l,0.0); */
/* Discount = pnl_mat_create_from_double(N_l,n_l, 0.0); */

/* std      = pnl_vect_create_from_double(N_l, 0.0); */

/* SmallJumpDrift( C_l, G_l, M_l, Y_l,  epsilon_l, 5000); */
/* bigJumpDrift = BigJumpDrift( C_l, G_l, M_l, Y_l,  epsilon_l, 10000); */

/* for(i=0;i<N_l;i++) */
/* for(j=0;j<n_l;j++) */
/* *pnl_mat_lget(Log_L,i,j)=pnl_vect_get(G0_l,i); */

/* intensity = 2.0*C_l*pow(epsilon_l,-Y_l)/Y_l;    // intensity calculating from
/* // the majoring density is given by:  $C 1_{\{|X| > \epsilon\}} |x|^{-Y-1}$  */

/* indexSwapMat = 0;// getIndex( T_l, N_l+1, SwapMat );// T[indexSwapMat] = Sw
/* //printf("index of mat = %d\n", indexSwapMat); */
/* c = pnl_vect_create_from_double(N_l-indexSwapMat,de_l*K); */
/* *pnl_vect_lget(c,0) = -1.0; */
/* *pnl_vect_lget(c,N_l-indexSwapMat-1) = 1.0 + de_l*K; */

/* theta = pnl_vect_create_from_double(N_l,0.0); */
/* eta    = pnl_mat_create_from_double(N_l,N_l, 0.0); */

```

```

/* for(i=0; i<N_l ; i++) */
/*   { */

/*       *pnl_vect_lget(theta,i) = kappaCGMY(C_l,G_l,M_l,Y_l,pnl_vect_get(la_l,i)
/* for(j=i; j<N_l ; j++) */
/*   { */

/* *pnl_mat_lget(eta,i,j) = kappaCGMY(C_l,G_l,M_l,Y_l,pnl_vect_get(la_l,i)+ pnl_

/* for(k=j; k<N_l ; k++) */
/*   { */

/*       zeta[i][j][k] = kappaCGMY(C_l,G_l,M_l,Y_l,pnl_vect_get(la_l,i)+ pnl_vect_
/*       kappaCGMY(C_l,G_l,M_l,Y_l,pnl_vect_get(la_l,i)+ pnl_vect_get(la_l,j)) - k
/*       kappaCGMY(C_l,G_l,M_l,Y_l,pnl_vect_get(la_l,j)+ pnl_vect_get(la_l,k)) +
/*       kappaCGMY(C_l,G_l,M_l,Y_l,pnl_vect_get(la_l,j)) + kappaCGMY(C_l,G_l,M_l,Y

/*   } */

/* } */

/* } */

/* Payoff = 0.0; */
/* Payoffvar = 0.0; */

/* Payoff_receiv =0.0; */
/* var_receiv = 0.0; */

/* for(k=0;k<n_l;k++) */
/*   { */
/*       for(i=0;i<=indexSwapMat;i++) */
/*           { */
/*               if(i==0) */
/*               { */
/*                   m = m_0; */
/*                   dt = dt_0; */
/*               } */

```

```

/* else */
/* { */
/* m = m_l; */
/* dt = dt_l; */
/* } */
/* for(p=0;p<m;p++) */
/* { */
/*     numJumps = (int) pnl_rand_poisson(intensity*dt,0); // simulating the num

/*     dH = 0.0; */

/*     for(l=0;l<numJumps;l++) */
/*     { */
/*         Uniform1 = pnl_rand_uni (0); */
/*         Uniform2 = pnl_rand_uni (0); */

/*         pareto=epsilon_l*pow(Uniform1,-1.0/Y_l);           // simulating jump si
/*         pareto_p=( exp(-M_l*pareto)>Uniform2 ? pareto:0.0); // if jump is positiv
/*         pareto_n=( exp(-G_l*pareto)>Uniform2 ? pareto:0.0); // if jump is negativ

/*         binomial = pnl_rand_bernoulli(0.5,0);           // decides if the jum
/*         dH += binomial*pareto_p-(1-binomial)*pareto_n;   //aggregates the numb

/*     } */
/*     for(j=N_l-1;j>=indexSwapMat;j--) */
/*     { */
/*         *pnl_mat_lget(Log_L,j,k) = pnl_mat_get(Log_L,j,k) + b2(Log_L, j, k, de_l,N_l
/*                                     bigJumpDrift*pnl_vect_get(la_l,j)*
/*     } */

/* } */

/*     } */

/* sum = 0.0; */
/* sum_plus = 0.0; */
/* prod = 1.0;// 1.0+de*exp( pnl_mat_get(Log_L,0,k); */

/* for(j=N_l-1; j>=indexSwapMat;j--) */

```

```

/* { */
/*   prod *= 1.0+de_l*exp( pnl_mat_get(Log_L,j,k)); */
/* sum -= pnl_vect_get(c,j-indexSwapMat)*prod; */
/* sum_plus += pnl_vect_get(c,j-indexSwapMat)*prod; */
/* } */
/*       Payoff += MAX(sum,0.0) ; */
/* Payoff_receiv += MAX(sum_plus,0.0); */
/*       Payoffvar +=MAX(sum,0.0)*MAX(sum,0.0) ; */
/* var_receiv += MAX(sum_plus,0.0)*MAX(sum_plus,0.0) ; */

/*   } */
/* Payoff /= (double)n_l; */
/* Payoff_receiv /= (double)n_l; */
/* var_receiv /= (double)n_l; */
/* Payoffvar /= (double)n_l; */

/*   Payoffvar = 1.96*sqrt( (Payoffvar-Payoff* Payoff)/(double)n_l ); */
/*   var_receiv = 1.96*sqrt( (var_receiv-Payoff_receiv* Payoff_receiv)/(double)n_l );

/* if(swaption_payer_receiver ==0) //Is Receiver */
/* { */
/* *mean_l   = Payoff*pnl_vect_get(B0_l,N_l); */
/* *confid_l = Payoffvar * pnl_vect_get(B0_l,N_l); */
/* } */
/* else //Is Payer */
/* { */
/* *mean_l   = Payoff_receiv*pnl_vect_get(B0_l,N_l); */
/* *confid_l = var_receiv * pnl_vect_get(B0_l,N_l); */
/* } */
/* //printf("PFRA   = %f   and Confid_Inter = %f \ n", *mean_l, *confid_l); */

/* pnl_vect_free(&theta); */
/* pnl_vect_free(&std); */
/* pnl_vect_free(&c); */

/* pnl_mat_free(&Log_L); */
/* pnl_mat_free(&eta); */
/* pnl_mat_free(&Discount); */

```

```
/* } */
```

```
//simulates Libor rates using the first order Drift expansion and first order lo
//Calculates Swaption price
static void calculateSwaptionPriceLogLevy(double C_l, double G_l, double M_l, do
{
    double Uniform1, Uniform2, pareto, pareto_p, pareto_n, binomial, dH, bigJumpDr
    int i, j, k, p, l, q, m, m_0;
    double sum_plus, Payoff_receiv, var_receiv ;
    double intensity, Payoff, Payoffvar, sum, prod, dt_l, dt, dt_0;
    int numJumps, indexSwapMat;

    PnlVect *theta, *std, *b, *A, *DHZ, *c;
    PnlMat *Log_L , *eta, *Discount, *Z;
    double ***zeta;

    int izeta0, izeta1;
    zeta = malloc(sizeof(double **)*N_l);

    for (izeta0 = 0; izeta0 < N_l; ++izeta0)
    {
        zeta[izeta0] = malloc(sizeof(double *)*N_l);
        for (izeta1 = 0; izeta1 < N_l; ++izeta1)
        {
            zeta[izeta0][izeta1] = malloc(sizeof(double) * N_l);
        }
    }

    dt_l = de_l / (double)m_l;
    m_0 = (int) pnl_vect_get(T_l, 0) / dt_l + 1;
    dt_0 = pnl_vect_get(T_l, 0) / (double)m_0;

    indexSwapMat = 0;//getIndex( T_l, N_l+1, SwapMat );// T[indexSwapMat] = SwapM
    //printf("index of mat = %d\ n", indexSwapMat);
```



```

c = pnl_vect_create_from_double(N_1 - indexSwapMat, de_1 * K);
*pnl_vect_lget(c, 0) = -1.0;
*pnl_vect_lget(c, N_1 - indexSwapMat - 1) = 1.0 + de_1 * K;

```

```

Log_L = pnl_mat_create_from_double(N_1, n_1, 0.0);
Discount = pnl_mat_create_from_double(N_1, n_1, 0.0);

```

```

std      = pnl_vect_create_from_double(N_1, 0.0);

```

```

for (i = 0; i < N_1; i++)
    for (j = 0; j < n_1; j++)
        *pnl_mat_lget(Log_L, i, j) = pnl_vect_get(G0_1, i);

```

```

bigJumpDrift = BigJumpDrift(C_1, G_1, M_1, Y_1, epsilon_1, 100 * 5000);
//printf("BigDrift = %f\ n",bigJumpDrift);

```

```

intensity = 2.0 * C_1 * pow(epsilon_1, -Y_1) / Y_1; // intensity calculating f
// the majoring density is given by:  $C_1 \{ |X| > \epsilon \} |x|^{-Y-1}$ 

```

```

b = pnl_vect_create_from_double(N_1, 0.0);
A = pnl_vect_create_from_double(N_1, 0.0);
DHZ = pnl_vect_create_from_double(N_1, 0.0);

```

```

theta = pnl_vect_create_from_double(N_1, 0.0);
eta    = pnl_mat_create_from_double(N_1, N_1, 0.0);
Z = pnl_mat_create_from_double(N_1, n_1, 0.0);

```

```

for (i = 0; i < N_1; i++)
{
    *pnl_mat_lget(Log_L, i, 0) = log(pnl_vect_get(L0_1, i));
}

```

```

    for (k = 0; k < n_l; k++)
        *pnl_mat_lget(Z, i, k) = de_l * pnl_vect_get(L0_l, i) / (1.0 + de_l * pn
    }

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

    *pnl_vect_lget(theta, i) = kappaCGMY(C_l, G_l, M_l, Y_l, pnl_vect_get(la_l, i)

    for (j = i; j < N_l ; j++)
    {

        *pnl_mat_lget(eta, i, j) = kappaCGMY(C_l, G_l, M_l, Y_l, pnl_vect_get(la_l, j)
        for (k = j; k < N_l ; k++)
        {

            zeta[i][j][k] = kappaCGMY(C_l, G_l, M_l, Y_l, pnl_vect_get(la_l, i)
                        kappaCGMY(C_l, G_l, M_l, Y_l, pnl_vect_get(la_l, i)
                        kappaCGMY(C_l, G_l, M_l, Y_l, pnl_vect_get(la_l, j)
                        kappaCGMY(C_l, G_l, M_l, Y_l, pnl_vect_get(la_l, j)

        }

    }

}

//Calculating the A's, i.e. the drift of Z_i (see eq. 3.16)
// Note that terms cancel out in the compound Poisson approximation, the remainder
for (i = 0; i < N_l; i++)
{
    *pnl_vect_lget(b, i) = b2(Log_L, i, 0, de_l, N_l, la_l, theta, eta, zeta);

    *pnl_vect_lget(A, i) = de_l * pnl_vect_get(L0_l, i) / pow(1.0 + de_l * pnl_vect_get(la_l, i) * C_l * pnl_sf_gamma(-Y_l + 1)

}

```

Payoff = 0.0;

```

Payoffvar = 0.0;
Payoff_receiv = 0.0;
var_receiv = 0.0;

for (k = 0; k < n_l; k++)
{
    for (i = 0; i <= indexSwapMat; i++)
    {
        if (i == 0)
        {
            m = m_0;
            dt = dt_0;
        }
        else
        {
            m = m_l;
            dt = dt_l;
        }
        for (p = 0; p < m; p++)
        {
            numJumps = (int) pnl_rand_poisson(intensity * dt_l, 0); // simulat

            for (q = 0; q < N_l; q++)
                *pnl_vect_lget(DHZ, q) = 0.0;
            dH = 0.0;
            for (l = 0; l < numJumps; l++)
            {
                Uniform1 = pnl_rand_uni(0);
                Uniform2 = pnl_rand_uni(0);

                pareto = epsilon_l * pow(Uniform1, -1.0 / Y_l); // simu
                pareto_p = (exp(-M_l * pareto) > Uniform2 ? pareto : 0.0); //
                pareto_n = (exp(-G_l * pareto) > Uniform2 ? pareto : 0.0); //

                binomial = pnl_rand_bernoulli(0.5, 0); // deci
                dH += binomial * pareto_p - (1 - binomial) * pareto_n; //aggre

                for (q = 0; q < N_l; q++)
                    *pnl_vect_lget(DHZ, q) += de_l * pnl_vect_get(L0_l, q) * exp
            }
        }
    }
}

```

```

        for (j = N_1 - 1; j >= indexSwapMat; j--)
        {
            *pnl_mat_lget(Z, j, k) = pnl_mat_get(Z, j, k) + pnl_vect_get(A, j, k);
            *pnl_mat_lget(Log_L, j, k) = pnl_mat_get(Log_L, j, k) + b1Z(Z, j, k) +
                bigJumpDrift * pnl_vect_get(la_1, j, k);
        }
    }

}

sum = 0.0;
prod = 1.0;
sum_plus = 0.0;
for (j = N_1 - 1; j >= indexSwapMat; j--)
{
    prod *= 1.0 + de_1 * exp(pnl_mat_get(Log_L, j, k));
    sum -= pnl_vect_get(c, j - indexSwapMat) * prod;
    sum_plus += pnl_vect_get(c, j - indexSwapMat) * prod;
}
Payoff += MAX(sum, 0.0) ;
Payoff_receiv += MAX(sum_plus, 0.0);
Payoffvar += MAX(sum, 0.0) * MAX(sum, 0.0) ;
var_receiv += MAX(sum_plus, 0.0) * MAX(sum_plus, 0.0) ;
}
Payoff /= (double)n_1;
Payoffvar /= (double)n_1;
Payoff_receiv /= (double)n_1;
var_receiv /= (double)n_1;

//printf("mean = %f, var = %f \ n",Payoff,Payoffvar);
Payoffvar = 1.96 * sqrt((Payoffvar - Payoff * Payoff) / (double)n_1);
var_receiv = 1.96 * sqrt((var_receiv - Payoff_receiv * Payoff_receiv) / (double)n_1);

if (swaption_payer_receiver == 0) /////Is Receiver
{
    *mean_1 = Payoff * pnl_vect_get(B0_1, N_1);
    *confid_1 = Payoffvar * pnl_vect_get(B0_1, N_1);
}

```

```

    }
else
{
    *mean_l    = Payoff_receiv * pnl_vect_get(B0_l, N_l);
    *confid_l = var_receiv * pnl_vect_get(B0_l, N_l);
}

//printf("PFRA  = %f  and Confid_Inter = %f \n", *mean_l, *confid_l);

for (izeta0 = 0; izeta0 < N_l; ++izeta0)
{
    for (izeta1 = 0; izeta1 < N_l; ++izeta1)
    {
        free(zeta[izeta0][izeta1]);
    }
    free(zeta[izeta0]);
}
free(zeta);

pnl_vect_free(&theta);
pnl_vect_free(&std);
pnl_vect_free(&b);
pnl_vect_free(&A);
pnl_vect_free(&DHZ);
pnl_vect_free(&c);

pnl_mat_free(&Log_L);
pnl_mat_free(&eta);
pnl_mat_free(&Discount);
pnl_mat_free(&Z);
}

static int mc_loglevy_cgmy_swaption(NumFunc_1 *p, double l0, double sigma, double
{
    int m = 5; //Number of discretization steps
    int swaption_payer_receiver;
    int N;
    PnlVect *la; //volatility parameters (\ lambda_i's in the

```

```

double epsilon = 0.001;           //truncation for small jumps
int i;
double r0; //Flat continuous yield to maturity
PnlVect *T, *B0, *L0, *G0, *Z0;

swaption_payer_receiver = ((p->Compute) == &Put);
N = (swap_maturity - swaption_maturity) / period;

la = pnl_vect_create_from_double(N + 1, sigma);

T = pnl_vect_create_from_double(N + 1, swaption_maturity);           //T: vect
L0 = pnl_vect_create_from_double(N, 10);           // initial Libor curve

r0 = log(1.0 + period * 10) / period;
B0 = pnl_vect_create_from_double(N + 1, exp(-r0 * swaption_maturity));

G0 = pnl_vect_create_from_double(N, 0.0);           // initial Log_Libor curve
Z0 = pnl_vect_create_from_double(N, 0.0);           // initial values for the

//Initializing T, B0, G0, Z0
for (i = 1; i < N + 1; i++)
{
    *pnl_vect_lget(T, i) = period + pnl_vect_get(T, i - 1) ;
    *pnl_vect_lget(B0, i) = exp(-r0 * pnl_vect_get(T, i));
    /*pnl_vect_lget(L0,i-1) = (pnl_vect_get(B0,i-1)/(double) pnl_vect_get(B0,
    *pnl_vect_lget(G0, i - 1) = log(pnl_vect_get(L0, i - 1));
    *pnl_vect_lget(Z0, i - 1) = period * pnl_vect_get(L0, i - 1) / (1.0 + per
}

pnl_rand_sseed(0, 25);

// calculateSwaptionPrice(C,G,M,Y, epsilon, period, N,T,L0,B0, G0,la, n, m, p
pnl_rand_sseed(0, 25);

calculateSwaptionPriceLogLevy(C, G, M, Y, epsilon, period, N, T, L0, B0, G0, 1

pnl_vect_free(&T); // *T, *B0, *L0, *G0, *Z0;

```

```

    pnl_vect_free(&B0);
    pnl_vect_free(&L0);
    pnl_vect_free(&G0);
    pnl_vect_free(&Z0);
    pnl_vect_free(&la);

    return OK;
}

int CALC(MC_LOGLEVY_SWAPTION)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    return mc_loglevy_cgmy_swaption(
        ptOpt->PayOff.Val.V_NUMFUNC_1,
        ptMod->l0.Val.V_PDOUBLE,
        ptMod->Sigma.Val.V_PDOUBLE,
        ptMod->C.Val.V_PDOUBLE,
        ptMod->G.Val.V_PDOUBLE,
        ptMod->M.Val.V_PDOUBLE,
        ptMod->Y.Val.V_PDOUBLE,
        ptOpt->BMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
        ptOpt->OMaturity.Val.V_DATE - ptMod->T.Val.V_DATE,
        ptOpt->Nominal.Val.V_PDOUBLE,
        ptOpt->FixedRate.Val.V_PDOUBLE,
        ptOpt->ResetPeriod.Val.V_DATE,
        Met->Par[0].Val.V_ENUM.value,
        Met->Par[1].Val.V_LONG,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(MC_LOGLEVY_SWAPTION)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "PayerSwaption") == 0) || (strcmp(((Option
        return OK;
    else
        return WRONG;
}

#endif //PremiaCurrentVersion

```

```

static PremiaEnumMember skovmand_method_members[] =
{
    { "Log Levy", 1},
    { "Second Order Drift Expansion", 2},
    { NULL, NULLINT }
};

static DEFINE_ENUM(skovmand_method, skovmand_method_members);

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

        Met->Par[0].Val.V_ENUM.value = 0;
        Met->Par[0].Val.V_ENUM.members = &PremiaEnumRNGs;
        Met->Par[1].Val.V_LONG = 1000;
        Met->Par[2].Val.V_ENUM.value = 1;
        Met->Par[2].Val.V_ENUM.members = &skovmand_method;
    }
    return OK;
}

PricingMethod MET(MC_LOGLEVY_SWAPTION) =
{
    "MC_LogLevy_Swaption",
    {
        {"RandomGenerator", ENUM, {100}, ALLOW},
        {"N Simulation", LONG, {100}, ALLOW},
        {"Method", ENUM, {100}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_LOGLEVY_SWAPTION),
    {
        {"Price", DOUBLE, {100}, FORBID},
        {"Price Error", DOUBLE, {100}, FORBID},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_LOGLEVY_SWAPTION),

```



```
    CHK_ok,  
    MET(Init)  
} ;
```