

[Help](#)

```
#include <stdlib.h>
#include "pnl/pnl_complex.h"
#include "pnl/pnl_specfun.h"
#include "pnl/pnl_vector.h"
#include "
href../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "
href../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2011+2) //The "#els
static int CHK_OPT(MC_GlassermanKimMod_Heston)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_GlassermanKimMod_Heston)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double UpInter = 10000;

static int rand_bessel(double mu, double z, int generator)
{
    //-----Inittialization of variable
    double p0;
    double tmp, u;
    int n;
    //-----Begin operation
    p0 = pow(z * 0.5, mu) / (pnl_bessel_i(mu, z) * pnl_sf_gamma_inc(mu + 1., 0.));
    u = pnl_rand_uni(generator);
    tmp = 0;
    n = 0;
    if (u <= p0)
        return 0;
    do
    {
        tmp = tmp + p0;
        p0 = p0 * z * z / (4.*(((double)n) + 1.) * (((double)n) + 1. + mu));
```



```

        n++;
    }
    while ((u > tmp + p0));
    return n;
}

```

```

//-----The calculus of Laplace Transform for the variable X2
static dcomplex Lap_X_2(dcomplex x, double t, double sigma, double kappa, double L)
{
    //-----Initialization of parameter
    dcomplex tmp2 = CZERO;
    dcomplex tmp1 = CZERO;
    dcomplex tmp3 = CZERO;
    double ptheta = 4.*kappa * theta / (sigma * sigma);

    //-----Begin operation
    //-----L calculus --> tmp2
    tmp1.r = kappa * kappa - 2.*sigma * sigma * x.r;
    tmp1.i = -2.*sigma * sigma * x.i;
    tmp2 = Cpow_real(tmp1, 0.5);
    //-----L/sinh(0.5t*L) --> tmp3
    tmp1 = CRmul(tmp2, 0.5 * t);
    tmp3 = Csinh(tmp1);
    tmp1 = Cinv(tmp3);
    tmp3 = Cmul(tmp2, tmp1);
    tmp1 = CRmul(tmp3, sinh(kappa * t * 0.5) / kappa);
    return Cpow_real(tmp1, ptheta * 0.5);
}

```

```

//-----The calculus of Laplace Transform for the variable X3
static dcomplex Lap_X_3(dcomplex x, double t, double sigma, double kappa)
{
    //-----Initialization of parameter
    dcomplex tmp2 = CZERO;
    dcomplex tmp1 = CZERO;
    dcomplex tmp3 = CZERO;

    //-----Begin operation

```



```

//-----L calculus --> tmp2
tmp1.r = kappa * kappa - 2.*sigma * sigma * x.r;
tmp1.i = -2.*sigma * sigma * x.i;
tmp2 = Cpow_real(tmp1, 0.5);
//-----L/sinh(0.5t*L) --> tmp3
tmp1 = CRmul(tmp2, 0.5 * t);
tmp3 = Csinh(tmp1);
tmp1 = Cinv(tmp3);
tmp3 = Cmul(tmp2, tmp1);
tmp1 = CRmul(tmp3, sinh(kappa * t * 0.5) / kappa);
return Cpow_real(tmp1, 2.);

}
//-----Sample gthe law X1 by trancation series.
static double X_1_sample(double order_tr, double t, double kappa, double sigma,
{
//-----Declaration of variable
double lambda_n;
double gamma_n;
int pss;
int j, n;
double tmp;

//-----Compte the sum part

tmp = 0.;
for (n = 1; n <= order_tr; n++)
{
lambda_n = 16.*M_PI * M_PI * ((double)n) * ((double)n) / (sigma * sigma *
gamma_n = (kappa * kappa * t * t + 4.*M_PI * M_PI * ((double)n) * ((double)

pss = pnl_rand_poisson(lambda_n * (v0 + vt), generator);

for (j = 1; j <= pss; j++)
tmp = tmp + pnl_rand_exp(1., generator) / gamma_n;
}

//-----compute the rest

lambda_n = 6.*(v0 + vt) * ((double)order_tr) / (sigma * sigma * t);

```



```

gamma_n = sigma * sigma * t * t / (3.*M_PI * M_PI * ((double)order_tr) * ((do

tmp = tmp + pnl_rand_gamma(lambda_n, gamma_n, generator);

//printf("The value of tmp = %f \ n",tmp);
return tmp;

}

```

```

//-----The calculus of cumulative function for the variable X2
static void Cumu_X_2_M(PnlMat *xv, double mprecision, double t, double kappa, do
{

```

```

//-----Declaration of variables

```

```

double ue;
int size_d;
double tmp1, tmp2, tmp3;
dcomplex ctmp1, ctmp2;
int N, k, h;
double p_theta;
double x;
double w;

```

```

//-----Calculus of ue

```

```

tmp3 = kappa * t * 0.5;

```

```

tmp1 = sigma * sigma * (-2. + kappa * t / tanh(tmp3)) / (4.*kappa * kappa);

```

```

tmp2 = pow(sigma, 4.) * (-8. + 2.*kappa * t / tanh(tmp3) + kappa * kappa * t *

```

```

p_theta = 4.*kappa * theta / (sigma * sigma);

```

```

ue = tmp1 * (p_theta) + mprecision * sqrt(tmp2 * (p_theta));

```

```

//-----Begin the loop on the variable xv

```

```

size_d = (int) dim + 1;

```



```

w = 0.01;
pnl_mat_resize(xv, size_d, 2);
h = 0;
for (h = 0; h < size_d ; h++)
{
    pnl_mat_set(xv, h, 0, w * tmp1 + ((double)(h) / (double)dim) * (ue - w * t
}

for (h = 0; h < size_d; h++)
{
    x = pnl_mat_get(xv, h, 0);
    //-----Calculus of the trancation --> N
    N = 2;
    tmp1 = 2 * M_PI / (x + ue);

    ctmp1 = CI;
    tmp3 = 0.000001 * M_PI * 0.5 * tmp1;
    do
    {
        N++;
        ctmp2 = RCmul(tmp1 * ((double)N), ctmp1);
        tmp2 = Cabs(Lap_X_2(ctmp2, t, sigma, kappa, theta));
        if (N == 10000)
            break;
    }
    while ((double)(tmp2 / (double)N) > tmp3);

    //-----Calculus of the sum
    tmp3 = 0.;
    k = 0;
    for (k = 1; k <= N; k++)
    {
        ctmp2 = RCmul(tmp1 * ((double)k), ctmp1);
        tmp3 = tmp3 + (double)((double)(sin(tmp1 * x * ((double)k))) * Creal(La

    }
    tmp3 = tmp3 * 2. / M_PI;

    //-----The rest of the sum
    pnl_mat_set(xv, h, 1, tmp3 + x * tmp1 / M_PI);

```



```

    }
    return;

}

static double I_inv_Cumu_X_Interp(PnlMat *xv, double u)
{
    double a, b;
    int k = 0;
    if (u <= pnl_mat_get(xv, 0, 1))
    {
        a = (pnl_mat_get(xv, 0, 1) - 0.) / (pnl_mat_get(xv, 0, 0) - 0.);
        b = 0.;
        return u / a;
    }
    for (k = 1; k < xv->m; k++)
    {
        if (u <= pnl_mat_get(xv, k, 1))
        {
            a = (pnl_mat_get(xv, k, 1) - pnl_mat_get(xv, k - 1, 1)) / (pnl_mat_get(xv, k, 0) - pnl_mat_get(xv, k - 1, 0));
            b = pnl_mat_get(xv, k, 1) - a * pnl_mat_get(xv, k, 0);
            return (u - b) / a;
        }
    }

    a = (1. - pnl_mat_get(xv, xv->m - 1, 1)) / (UpInter - pnl_mat_get(xv, xv->m - 1, 0));
    b = pnl_mat_get(xv, xv->m - 1, 1) - a * pnl_mat_get(xv, xv->m - 1, 0);
    return (u - b) / a;
}

//-----The calculus of cumulative function for the variable X2
static void Cumu_X_3_M(PnlMat *xv, double mprecision, double t, double kappa, double alpha)
{
    //-----Declaration of variables
    double ue;
    int size_d;
    double tmp1, tmp2, tmp3;
    dcomplex ctmp1, ctmp2;
    int N, k, h;

```



```

double p_theta;
double theta;
double x;
double w;
//-----Calculus of ue
theta = 2.*sigma * sigma / kappa;

tmp3 = kappa * t * 0.5;

tmp1 = sigma * sigma * (-2. + kappa * t / tanh(tmp3)) / (4.*kappa * kappa);

tmp2 = pow(sigma, 4.) * (-8. + 2.*kappa * t / tanh(tmp3) + kappa * kappa * t *

p_theta = 4.*kappa * theta / (sigma * sigma);

ue = tmp1 * (p_theta) + mprecision * sqrt(tmp2 * (p_theta));

//-----Begin the loop on the variable xv
size_d = (int) dim + 1;
w = 0.01;
pnl_mat_resize(xv, size_d, 2);
h = 0;
for (h = 0; h < size_d ; h++)
{
    pnl_mat_set(xv, h, 0, w * tmp1 + ((double)(h) / (double)dim) * (ue - w * t
}

for (h = 0; h < size_d; h++)
{
    x = pnl_mat_get(xv, h, 0);
    //-----Calculus of the trancation --> N
    N = 2;
    tmp1 = 2 * M_PI / (x + ue);
    ctmp1 = CI;
    tmp3 = 0.000001 * M_PI * 0.5 * tmp1;
    do
    {
        N++;
        ctmp2 = RCmul(tmp1 * ((double)N), ctmp1);
        tmp2 = Cabs(Lap_X_3(ctmp2, t, sigma, kappa));
    }
}

```



```

        if (N == 10000)
            break;
    }
    while ((double)(tmp2 / (double)N) > tmp3);

    //-----Calculus of the sum
    tmp3 = 0.;
    k = 0;
    for (k = 1; k <= N; k++)
    {
        tmp2 = RCmul(tmp1 * ((double)k), tmp1);
        tmp3 = tmp3 + (double)((double)(sin(tmp1 * x * ((double)k))) * Creal(La

    }
    tmp3 = tmp3 * 2. / M_PI;

    //-----The rest of the sum
    pnl_mat_set(xv, h, 1, tmp3 + x * tmp1 / M_PI);
}
return;
}

```

```

static double Sample_I_by_Inv(double t, double kappa, double sigma, double theta
{
    //-----Declaration of variable
    double tmp;
    double u;
    int i;
    int bess;
    int mprecision_X1;
    //-----Initializzation
    mprecision_X1 = 40;

    //-----Begin operations

    //----generate Z

```



```

tmp = 0.;
i = 0;
bess = rand_bessel(2.*theta * kappa / (sigma * sigma) - 1., 2.*kappa * sqrt(vt)

for (i = 1; i <= bess; i++)
{
    u = pnl_rand_uni(generator);

    tmp = tmp + I_inv_Cumu_X_Interp(F_X3, u);
}
//----generate int_0^t vs = X1 +X2 +X3 --> lambda

u = pnl_rand_uni(generator);
tmp = tmp + I_inv_Cumu_X_Interp(F_X2, u);

u = pnl_rand_uni(generator);
tmp = tmp + X_1_sample(mprecision_X1, t, kappa, sigma, v0, vt, generator

return tmp ;

}
//-----Sampling the transition probability (v(0)=X_t, v(1)=in
//-----dX_t = kappa(theta-X_t)dt + sigma sqrt(X_t)dW_t
//-----In the case of the inversion of the Lapla

static void Sample_C_By_Inv(PnlVect *v, double t, double kappa, double sigma, do
{
    //-----Declaration of variable
    double gamma, lambda;
    double tmp;
    //double tmp2;
    int j, pss;
    //-----Initialization of parammter
    tmp = 0.;
    j = 0;
    gamma = 4.*kappa / (sigma * sigma * (1. - exp(-kappa * t)));
    lambda = pnl_vect_get(v, 0) * gamma * exp(-kappa * t);
    //-----Begin operations
    //----generate the CIR process vt --> tmp

```



```

pss = pnl_rand_poisson(lambda * 0.5, generator);

for (j = 1; j <= pss; j++)
    tmp = tmp + pnl_rand_gamma(1., 2., generator);

tmp = tmp + pnl_rand_gamma(2.*kappa * theta / (sigma * sigma), 2., generator);
tmp = tmp / gamma;
//----generate the the integral --> lambda

lambda = Sample_I_by_Inv(t, kappa, sigma, theta , generator, pnl_vect_get(

//----set the new value
pnl_vect_set(v, 0, tmp);
pnl_vect_set(v, 1, lambda);
}

int MCGlassermanKimMod(double S0, NumFunc_1 *p, double T, double r, double q, d
{
    //-----Declaration of variable
    int j, call_put;
    PnlVect *vv;
    double tmp1, tmp, tmp2, tmp3;
    double tmpvar;
    int init_mc;
    double mt, sigmat;
    double epsilon;

    int dim;
    double mprecision2;
    double mprecision3;
    PnlMat *xv2;
    PnlMat *xv3;
    double K;

    if ((p->Compute) == &Call)
        call_put = 0;
    else
        call_put = 1;
    K = p->Par[0].Val.V_PDOUBLE;

```



```

//-----Initialization of variable
vv = pnl_vect_create_from_double(2, 0.);
pnl_vect_set(vv, 0, v0);
epsilon = 0.01;
init_mc = pnl_rand_init(generator, 1, (long)Nmc);
//-----Operation begins
tmp = 0.;
tmp2 = 0.;
tmpvar = 0.;

mprecision3 = 14;
mprecision2 = 5;
dim = 200;
xv3 = pnl_mat_create_from_double((int)dim + 1, 2, 0.);
xv2 = pnl_mat_create_from_double((int)dim + 1, 2, 0.);
Cumulative_X_3_M(xv3, mprecision3, T, kappa, sigma, dim);
Cumulative_X_2_M(xv2, mprecision2, T, kappa, sigma, theta, dim);

for (j = 1; j <= Nmc; j++)
{
    //printf("the value of the lopp is %d \ n",j);
    pnl_vect_set(vv, 0, v0);
    pnl_vect_set(vv, 1, 0.);
    Sample_C_By_Inv(vv, T, kappa, sigma, theta, generator, xv2, xv3);

    mt = (r - q - kappa * theta * rho / sigma) * T + rho * (pnl_vect_get(vv, 0));

    sigmat = sqrt((1 - rho * rho) * pnl_vect_get(vv, 1));

    //d1 = (log(S0/K)+mt+sigmat*sigmat)/sigmat;
    //d2 = d1 - sigmat;

    tmp1 = exp(mt + sigmat * pnl_rand_normal(generator));
    tmp3 = (S0 + epsilon) * tmp1;
    tmp1 = tmp1 * S0;

    if (call_put == 0) //call pricing
    {
        //tmp1= S0*exp(mt+0.5*sigma*sigma-r*T)*cdf_nor(d1)-K*exp(-r*T)*cdf_nor

```



```

        if (tmp1 >= K)
            tmp1 = tmp1 - K;
        else
            tmp1 = 0.;

        if (tmp3 >= K)
            tmp3 = tmp3 - K;
        else
            tmp3 = 0.;

    }
else
{
    //tmp1 =( 1.-K*exp(-r*T))*cdf_nor(d2)-S0*(1.-cdf_nor(d1));
    if (tmp1 <= K)
        tmp1 = -tmp1 + K;
    else
        tmp1 = 0.;

    if (tmp3 <= K)
        tmp3 = -tmp3 + K;
    else
        tmp3 = 0.;
}

tmp = tmp1 + tmp;
tmp2 = tmp3 + tmp2;

//-----confidence interval
tmpvar = tmp1 * tmp1 + tmpvar;
}

tmp = exp(-r * T) * tmp / ((double)Nmc);
tmp2 = exp(-r * T) * tmp2 / ((double)Nmc);
tmpvar = tmpvar / ((double) Nmc) - tmp * tmp;

*ptprice = tmp;
*ptdelta = (tmp2 - tmp) / epsilon;
*error_price = sqrt(tmpvar / ((double)Nmc));

```



```

//-----Free Memory
pnl_vect_free(&vv);
pnl_mat_free(&xv2);
pnl_mat_free(&xv3);

return init_mc;
}

int CALC(MC_GlassermanKimMod_Heston)(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 MCGlassermanKimMod(ptMod->S0.Val.V_PDOUBLE,
                               ptOpt->PayOff.Val.V_NUMFUNC_1,
                               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                               r,
                               divid, ptMod->Sigma0.Val.V_PDOUBLE
                               , ptMod->MeanReversion.Val.V_PDOUBLE,
                               ptMod->LongRunVariance.Val.V_PDOUBLE,
                               ptMod->Sigma.Val.V_PDOUBLE,
                               ptMod->Rho.Val.V_PDOUBLE,
                               Met->Par[0].Val.V_LONG, Met->Par[1].Val.V_ENUM.value
                               &(Met->Res[0].Val.V_DOUBLE),
                               &(Met->Res[1].Val.V_DOUBLE),
                               &(Met->Res[2].Val.V_DOUBLE)
                               );
}

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

```



```

    return  WRONG;
}
#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    //int type_generator;
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "mc_glassermankim_mod";
        Met->Par[0].Val.V_LONG = 100000;
        Met->Par[1].Val.V_ENUM.value = 0;
        Met->Par[1].Val.V_ENUM.members = &PremiaEnumMCRNGs;
    }
    return OK;
}

PricingMethod MET(MC_GlassermanKimMod_Heston) =
{
    "MC_GlassermanKimMod",
    { {"N iterations", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_GlassermanKimMod_Heston),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {"Error Price", DOUBLE, {100}, FORBID},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_GlassermanKimMod_Heston),
    CHK_mc,
    MET(Init)
};

```