

[Help](#)

```
extern "C" {
#include "kould_std.h"
#include "enums.h"
}

extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
    static int CHK_OPT(MC_Kou)(void *Opt, void *Mod)
    {
        return NONACTIVE;
    }
    int CALC(MC_Kou)(void *Opt, void *Mod, PricingMethod *Met)
    {
        return AVAILABLE_IN_FULL_PREMIA;
    }
#else
    static int Kou_Mc(NumFunc_1 *P, double S0, double T, double r, double divid, d
    {

        double K;
        K = P->Par[0].Val.V_DOUBLE;
        int j, n, np, n0 = n_points / 2;
        double s0, s, y, nu, pas = T / n_points, u;
        double *W, *g;
        W = new double[n_points + 1];
        g = new double[2];
        nu = (r - divid) - sigma * sigma / 2 - lambda * (p * lambdap / (lambdap - 1)
        double k = log(K / S0);

        pnl_rand_init(generator, 1, n_paths);

        //Put options case
        s = 0;
        n = 0;
        for (int i = 0; i < n_paths; i++)
        {
            W[0] = 0;
            for (j = 1; j < 2 * n0; j += 2)
            {
```

```

        g[0] = pnl_rand_normal(generator);
        g[1] = pnl_rand_normal(generator);

        W[j] = sigma * g[0] * sqrt(pas) + nu * pas + W[j - 1];
        W[j + 1] = sigma * g[1] * sqrt(pas) + nu * pas + W[j];
    }
    W[n_points] = sigma * pnl_rand_normal(generator) * sqrt(pas) + nu * pas
    np = pnl_rand_poisson(lambda * T, generator);

    s0 = 0;
    for (j = 1; j <= np; j++)
    {
        u = pnl_rand_uni(generator);

        if (1 - p <= u)
            s0 += -log(1 - (u - 1 + p) / p) / lambdap;
        else
            s0 += log(u / (1 - p)) / lambdam;
    }
    y = W[n_points] + s0;
    if (y <= k)
    {
        s += K - S0 * exp(y);
        n++;
    }
    }

//Put options
*ptprice = exp(-r * T) * s / n_paths;
*ptdelta = (*ptprice - exp(-r * T) * K * (double)n / n_paths) / S0;

//Call options
if ((P->Compute) == &Call)
{
    *ptprice += S0 * exp(-divid * T) - K * exp(-r * T);
    *ptdelta += exp(-divid * T);
}

delete [] W;
delete [] g;
return OK;
}

```

```
int CALC(MC_Kou)(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 Kou_Mc(ptOpt->PayOff.Val.V_NUMFUNC_1, ptMod->S0.Val.V_PDOUBLE, ptOpt)
}

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

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met, Option *Mod)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->Par[0].Val.V_ENUM.value = 0;
        Met->Par[0].Val.V_ENUM.members = &PremiaEnumMCRNGs;
        Met->Par[1].Val.V_PINT = 100;
        Met->Par[2].Val.V_LONG = 100000;
    }
    return OK;
}

PricingMethod MET(MC_Kou) =
{
    "MC_Kou",
    { {"RandomGenerator", ENUM, {100}, ALLOW},
      {"Number of discretization steps", LONG, {100}, ALLOW}, {"N iterations", L
```

```
    CALC(MC_Kou),  
    {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID}, {" ", P  
    CHK_OPT(MC_Kou),  
    CHK_ok,  
    MET(Init)  
  } ;  
}
```