

[Help](#)

```
#include "
href../../../../mod/doublehes1d/doublehes1d_std/doublehes1d_std_h_src.pdfhes1d_std.
#include "
href../../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2009+2) //The "#els
static int CHK_OPT(MC_KahlJackel_Heston)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_KahlJackel_Heston)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

int MCKahlJackel(double S0, NumFunc_1 *pf, double T, double r, double divid, do
{
    double delta = T / N_t_grid;
    int i;
    long k;
    double g1, g2;
    double price_sample, delta_sample, mean_price, mean_delta, var_price, var_delt
    double alpha, z_alpha;
    double KD, sq_delta, SD, sq_rho, V, log_S, Vpos1, Vpos2;
    double KDT, KDp1, SD2, RD1, RD2, RDS;
    double erT = exp((r - divid) * T);

    //Useful constants
    KD = K_heston * delta;
    sq_delta = sqrt(delta);
    SD = sigma * sq_delta;
    sq_rho = sqrt(1 - rho * rho);

    KDT = KD * Theta;
    KDp1 = KD + 1.;
    SD = sigma * sqrt(delta);
    SD2 = 0.25 * pow(sigma, 2.) * delta;
    RD1 = rho * sq_delta;
```

```

RD2 = sq_rho * sq_delta;
RDS = 0.25 * rho * delta * sigma;

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

/*Initialisation*/
mean_price = 0.0;
mean_delta = 0.0;
var_price = 0.0;
var_delta = 0.0;

pnl_rand_init(generator, 1, N_sample);

for (k = 0; k < N_sample; k++)
{
    // N_path Paths
    V = v0;
    log_S = log(S0);
    for (i = 0; i < N_t_grid; i++)
    {
        g1 = pnl_rand_normal(generator);
        g2 = pnl_rand_normal(generator);
        Vpos1 = MAX(V, 0.);
        V += KDT + SD * sqrt(Vpos1) * g1 + SD2 * (pow(g1, 2.) - 1.);
        V /= KDp1;
        Vpos2 = MAX(V, 0.); // max(V(t+delta),0)

        log_S += -0.25 * delta * (Vpos1 + Vpos2) + RD1 * sqrt(Vpos1) * g1 +
                0.5 * (sqrt(Vpos1) + sqrt(Vpos2)) * RD2 * g2 + RDS * (pow(g1,
    }

    /*Price*/
    price_sample = (pf->Compute)(pf->Par, erT * exp(log_S));

    /* Delta */
    if (price_sample > 0.0)
        delta_sample = (erT * exp(log_S) / S0);
    else delta_sample = 0.;
}

```

```

        /* Sum */
        mean_price += price_sample;
        mean_delta += delta_sample;

        /* Sum of squares */
        var_price += SQR(price_sample);
        var_delta += SQR(delta_sample);

    }
    /* End of the N iterations */

    /* Price estimator */
    *ptprice = (mean_price / (double)N_sample);
    *pterror_price = exp(-r * T) * sqrt(var_price / (double)N_sample - SQR(*ptprice));
    *ptprice = exp(-r * T) * (*ptprice);

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

    /* Delta estimator */
    *ptdelta = exp(-r * T) * (mean_delta / (double)N_sample);
    if ((pf->Compute) == &Put)
        *ptdelta *= (-1);
    *pterror_delta = sqrt(exp(-2.0 * r * T) * (var_delta / (double)N_sample - SQR(*ptdelta)));

    /* Delta Confidence Interval */
    *inf_delta = *ptdelta - z_alpha * (*pterror_delta);
    *sup_delta = *ptdelta + z_alpha * (*pterror_delta);

    return OK;
}

int CALC(MC_KahlJackel_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 MCKahlJackel(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_INT,
                    Met->Par[2].Val.V_ENUM.value,
                    Met->Par[3].Val.V_RGDOUBLE12,
                    Met->Par[4].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),
                    &(Met->Res[6].Val.V_DOUBLE),
                    &(Met->Res[7].Val.V_DOUBLE));

}

static int CHK_OPT(MC_KahlJackel_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->Par[0].Val.V_LONG = 15000;
    Met->Par[1].Val.V_INT = 100;
    Met->Par[2].Val.V_ENUM.value = 0;
    Met->Par[2].Val.V_ENUM.members = &PremiaEnumMCRNGs;
    Met->Par[3].Val.V_RGDOUBLE12 = 1.5;
    Met->Par[4].Val.V_DOUBLE = 0.95;
}

return OK;
}

PricingMethod MET(MC_KahlJackel_Heston) =
{
    "MC_KahlJackel",
    { {"N iterations", LONG, {100}, ALLOW},
      {"TimeStepNumber", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"THRESHOLD", DOUBLE, {100}, ALLOW},
      {"Confidence Value", DOUBLE, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_KahlJackel_Heston),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {"Error Price", DOUBLE, {100}, FORBID},
      {"Error 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_KahlJackel_Heston),
    CHK_mc,
    MET(Init)
};

```