

[Help](#)

```
#include "hes1d_std.h"
#include "enums.h"

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

int MCLord(double S0, NumFunc_1 *pf, double T, double r, double divid, double v
{
    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, Vpos;
    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);

    /* 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_sample 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);

        Vpos = MAX(V, 0.);
        V += KD * (Theta - Vpos) + SD * sqrt(Vpos) * g1;
        log_S += (-0.5 * Vpos * delta + sqrt(Vpos) * (rho * g1 + sq_rho * g2))
    }

    /*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(

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

return OK;
}

int CALC(MC_Lord_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 MCLord(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_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_Lord_Heston)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt)->Name, "PutEuro") == 0))
        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_DOUBLE = 0.95;
    }

    return OK;
}

PricingMethod MET(MC_Lord_Heston) =
{
    "MC_Lord",

```

```

{ {"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_Lord_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_Lord_Heston),
CHK_mc,
MET(Init)
};

```