

[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_Andersen_Heston)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_Andersen_Heston)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

int MCAndersen(double S0, NumFunc_1 *pf, double T, double r, double divid, doub
{
    double log_S0 = log(S0);
    double delta = T / N_t_grid;
    double unif;
    double *vol_path, *logstock_path;

    // USING CENTRAL DISCRETIZATION
    double K1 = 0.5 * delta * (K_heston * rho / sigma - 0.5) - rho / sigma;
    double K2 = 0.5 * delta * (K_heston * rho / sigma - 0.5) + rho / sigma;
    double K3 = 0.5 * delta * (1 - pow(rho, 2));
    double K4 = 0.5 * delta * (1 - pow(rho, 2));
    double K000 = 0.0;
    double A = rho / sigma * (1 + K_heston * 0.5 * delta) - 0.5 * 0.5 * delta * SQ

    double m, s2, psi;
    double b2, b, a;
    double p, beta;
    int i;
    long k;
    double g1, g2;
    double price_sample, delta_sample, mean_price, mean_delta, var_price, var_delt
    double alpha, z_alpha;
```

```

/* 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);
vol_path = malloc(sizeof(double) * (N_t_grid + 1));
logstock_path = malloc(sizeof(double) * (N_t_grid + 1));

vol_path[0] = v0;
logstock_path[0] = log_S0;

for (k = 0; k < N_sample; k++)
{
    // N_path Paths

    for (i = 0; i < N_t_grid; i++)
    {
        // for every path
        m = Theta + (vol_path[i] - Theta) * exp(-K_heston * delta);
        s2 = vol_path[i] * pow(sigma, 2) * exp(-K_heston * delta) * (1. - exp
        psi = s2 / pow(m, 2.);

        if (psi <= threshold)
        {
            b2 = 2 / psi - 1 + sqrt(2 / psi) * sqrt(2 / psi - 1);
            a = m / (1 + b2);
            b = sqrt(b2);
            g1 = pnl_rand_normal(generator);
            vol_path[i + 1] = a * pow(b + g1, 2.);
            K000 = -(A * b2 * a) / (1 - 2 * A * a) + 0.5 * log(1 - 2 * A * a)
        }
        else
        {
            p = (psi - 1) / (psi + 1);

```

```

        beta = 2 / (m * (psi + 1));
        unif = pnl_rand_uni(generator);
        if (unif <= p) vol_path[i + 1] = 0;
        else vol_path[i + 1] = 1 / beta * log((1 - p) / (1 - unif));
        K000 = -log(p + (beta * (1 - p)) / (beta - A)) - (K1 + 0.5 * K3) *
    }
    g2 = pnl_rand_normal(generator);
    logstock_path[i + 1] = logstock_path[i] + K000 + K1 * vol_path[i] + K2
}
/*Price*/
price_sample = (pf->Compute)(pf->Par, exp(logstock_path[N_t_grid]));

/* Delta */
if (price_sample > 0.0)
    delta_sample = (exp(logstock_path[N_t_grid]) / 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);

free(vol_path);
free(logstock_path);

return OK;
}

int CALC(MC_Andersen_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 MCAndersen(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_Andersen_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_Andersen_Heston) =
{
    "MC_Andersen",
    { {"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_Andersen_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_Andersen_Heston),
CHK_mc,
MET(Init)
};

```