

[Help](#)

```
#include "
href../../../../mod/doublehes1d/doublehes1d_vol/doublehes1d_vol_h_src.pdfhes1d_vol.
#include"pnl/pnl_vector.h"
#include"pnl/pnl_random.h"
#include "pnl/pnl_mathtools.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2013+2) //The "#els
static int CHK_OPT(MC_Timer)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_Timer)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

//Simulating trajectories of Variance process and the stopping time tau under He
static void getSpotPathEuler(double kappa, double theta, double sigma, double v0)
{
    int i, j;
    double Z, x, x1, H;
    double dt;

    dt = V / (double)m;

    for (j = 0; j < n; j++)
    {
        x1 = v0;
        H = dt / v0;

        for (i = 1; i < m; i++)
        {
            Z = pnl_rand_normal(generator);

            x = x1 + kappa * (theta / x1 - 1.0) * dt + sigma * sqrt(dt) * Z;
            H += dt / x;
            x1 = x;
        }
    }
}
```

```

    }
    pnl_mat_set(VT, j, 0, x);
    pnl_mat_set(VT, j, 1, H);
}

return;

}

int MCtimer(double s0, NumFunc_1 *p, double V, double r, double q, double v0, do
{
    int init_mc, simulation_dim;
    PnlMat *VTau;
    double c, d1, d2, var, tau;
    double K, error, mean_price, price;
    int i;

    K = p->Par[0].Val.V_PDOUBLE;
    /* Size of the random vector we need in the simulation */
    simulation_dim = m;

    /* MC sampling */
    init_mc = pnl_rand_init(generator, simulation_dim, n);
    /* Test after initialization for the generator */
    if (init_mc == OK)
    {
        mean_price = 0.0;
        error = 0.0;
        VTau = pnl_mat_create(n, 2);
        getSpotPathEuler(kappa, theta, sigma, v0, V, generator, n, m, VTau);
        //pnl_mat_print(VTau);

        for (i = 0; i < n; i++)
        {
            var = pnl_mat_get(VTau, i, 0);
            tau = pnl_mat_get(VTau, i, 1);

            c = (var - v0 - kappa * theta * tau + kappa * V) / sigma;

            d1 = (log(s0 / K) + (r - q) * tau + rho * c + (0.5 - SQR(rho)) * V) /

```

```

        d2 = d1 - sqrt((1.0 - SQR(rho)) * V);

        price = s0 * exp(-q * tau) * exp(rho * c - 0.5 * SQR(rho) * V) * cdf_n
        mean_price += price;
        error += SQR(price);
    }
    *ptprice = mean_price / (double) n;
    *inf_price = *ptprice - 1.96 * sqrt((error / ((double) n) - (*ptprice) * (
    *sup_price = *ptprice + 1.96 * sqrt((error / ((double) n) - (*ptprice) * (
    }
    pnl_mat_free(&VTau);

    return init_mc;
}

int CALC(MC_Timer)(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 MCTimer(ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOff.Val.V_NUMFUNC_1,
        ptOpt->VarianceBudget.Val.V_PDOUBLE,
        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->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE),
        &(Met->Res[2].Val.V_DOUBLE));
}

static int CHK_OPT(MC_Timer)(void *Opt, void *Mod)

```

```

{

    if ((strcmp(((Option *)Opt)->Name, "Timer") == 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->HelpFilenameHint = "mc_timer";

        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;
    }
    return OK;
}

PricingMethod MET(MC_Timer) =
{
    "MC_Timer",
    { {"N iterations", LONG, {100}, ALLOW},
      {"TimeStepNumber", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_Timer),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Inf Price", DOUBLE, {100}, FORBID},
      {"Sup Price", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_Timer),
    CHK_mc,

```

```
    MET(Init)  
};
```