

[Help](#)

```
#include <stdlib.h>
#include "
href../../mod/hes4over2/hes4over2_std/hes4over2_std_h_src.pdfhes4over2_std.h"
#include "
href../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2016+2) //The "#els
static int CHK_OPT(MC_Alfonsi_Heston4over2)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_Alfonsi_Heston4over2)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double Phi(double x, double t, double w, double k, double theta, double s
{
    double Psi_k_t2;
    double phi;
    double aa;
    Psi_k_t2 = (1 - exp( - k * t / 2)) / k;
    aa = k * theta;

    phi = (aa - pow(sigma, 2) / 4) * Psi_k_t2 + exp(- k * t / 2) * pow(sigma / 2 *

    return phi;
}

// We calculate K2(t):

static double K2(double t, double k, double theta, double sigma)
{
    double K2;
    double Psi_k_t2;
    double aa;
    K2 = 0;
    Psi_k_t2 = (1 - exp( - k * t / 2)) / k;
```

```

aa = k * theta;

if (pow(sigma, 2) > 4 * aa)
{
    K2 = exp(k * t / 2) * ((pow(sigma, 2) / 4 - aa) * Psi_k_t2 + pow((sigma /
}
return K2;
}

// u1 and u2: We calculate the first and second moments of CIR process departing

static int Moments_CIR(double *u1, double *u2, double t, double x, double k, dou
{
    double Psi_k_t = (1 - exp(- k * t)) / k;
    double aa = k * theta;
    *u1 = x * exp(- k * t) + aa * Psi_k_t;
    *u2 = pow(*u1, 2) + pow(sigma, 2) * Psi_k_t * (aa * Psi_k_t / 2 + x * exp(- k
    return 0;
}

// We calculate the value of CIR at time t, departing from x using the third ord

static int CIR_02(double *x1, double t, double k, double theta, double sigma,int
{
    double U = pnl_rand_uni(generator);
    double Y, G;
    double u1, u2, pi;
    Y = sqrt(3) * (U < 1. / 6) - sqrt(3) * (U > 5. / 6);
    G = pnl_rand_normal(generator);

    if (*x1 > K2(t, k, theta, sigma))
    {
        *x1 = Phi(*x1, t, sqrt(t) * G, k, theta, sigma);
    }
    else
    {
        Moments_CIR(&u1, &u2, t, *x1, k, theta, sigma);

        pi = (1 - sqrt(1 - pow(u1, 2) / u2)) / 2;

```

```

    if (U < pi)
    {
        *x1 = u1 / (2 * pi);
    }
    else
    {
        *x1 = u1 / (2 * (1 - pi));
    }
}

return 0;
}

// The scheme associate to operator L2

static int HW2(double *x1, double *x2, double a, double b, double rho, double t,
{

    double G01 = pnl_rand_normal(generator);

    *x2 = *x2 * exp(sqrt((1 - pow(rho, 2)) * t) * (a * sqrt(*x1) + b / sqrt(*x1))

    return 0;
}

// The scheme associated to operator L1

static int HW1(double *x1, double *x2, double *x3, double *x4, double *x5, doubl
{
    double dx1, dinvx1, dlogx1, factorexp;

    dx1 = - pow(a, 2) * *x1; // delta x1

    dinvx1 = - pow(b, 2) / *x1; // delta (1 / x1)

    dlogx1 = - log(*x1); // delta (log(x1))

    // We calculate the CIR after time t departing from x1:

    CIR_02(x1, t, k, theta, sigma,generator);

```

```

dx1 = dx1 + pow(a, 2) * *x1;

dinvx1 = dinvx1 + pow(b, 2) / *x1;

dlogx1 = dlogx1 + log(*x1);

// We use trapezoidal rule to calculate the integrals for X3, X4 and X5:

*x3 = *x3 + (pow(a, 2) * *x1 - 0.5 * dx1) * t;

*x4 = *x4 + (pow(b, 2) / *x1 - 0.5 * dinvx1) * t;

*x5 = *x5 + 0.5 * *x2 * t;

// We calculate the exponential factor to multiply be multiplied by X2

if (a == 0)
{
    factorex = exp((r - divid + b * rho * k / sigma) * t + b * rho / sigma *
}
else
{
    if (b == 0)
{
    factorex = exp((r - divid - a * rho * k * theta / sigma) * t + rho / (sigma *
}
    else
{
    factorex = exp((r - divid - a * b - a * rho * k * theta / sigma + b * rho * k
}
    }

;
*x2 = *x2 * factorex;

*x5 = *x5 + 0.5 * *x2 * t;

return 0;
}

```



```

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;

/* Size of the random vector we need in the simulation */
simulation_dim = N_time;

/* MC sampling */
init_mc = pnl_rand_init(generator,N_time,N_mc);
/* Test after initialization for the generator */
if (init_mc == OK)
{
    for (ipath = 1; ipath <= N_mc; ipath++)
    {

x1 = V0;
x2 = S0;
x3 = 0;
x4 = 0;
x5 = 0;

    for (i = 0; i < N_time; i++)
{
    Moments_CIR(&u1, &u2, t, x1, k, theta, sigma);

    //At each time, we calculate the composition of p1 and p2 (or in the inverse o

    Path(&x1, &x2, &x3, &x4, &x5, a, b, V0, r, divid, rho, t, k, theta, sigma, gene
}

        /*Price*/
S_T=x2;
        price_sample = (p->Compute)(p->Par,S_T);

```

```

        /* Delta */
        if (price_sample > 0.0)
            delta_sample = (S_T / 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_mc);
*pterror_price = exp(-r * T) * sqrt(var_price / (double)N_mc - 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_mc);
if ((p->Compute) == &Put)
    *ptdelta *= (-1);
*pterror_delta = sqrt(exp(-2.0 * r * T) * (var_delta / (double)N_mc - SQR(*ptdelta)));

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

/*Memory desallocation*/

return init_mc;
}

```

```

int CALC(MC_Alfonsi_Heston4over2)(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 MCAlfonsi4over2(ptMod->S0.Val.V_PDOUBLE,
                           ptOpt->PayOff.Val.V_NUMFUNC_1,
                           ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                           r,
                           divid, ptMod->a.Val.V_PDOUBLE, ptMod->b.Val.V_PDOUBLE, ptMod->Sigma0.Val.V_PDOU
                           , 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_Alfonsi_Heston4over2)(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_DOUBLE = 0.95;
    }

    return OK;
}

PricingMethod MET(MC_Alfonsi_Heston4over2) =
{
    "MC_Alfonsi",
    { {"N iterations", LONG, {100}, ALLOW},
      {"TimeStepNumber", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"Confidence Value", DOUBLE, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_Alfonsi_Heston4over2),
    { {"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_Alfonsi_Heston4over2),
    CHK_mc,
    MET(Init)
};

```