

[Help](#)

```
#include "
href../../../../mod/merhes1d/merhes1d_pad/merhes1d_pad_h_src.pdfhes1d_pad.h"
#include "
href../../../../common/math/alfonsi_h_src.pdfmath/alfonsi.h"
#include "
href../../../../common/enums_h_src.pdfenums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
static int CHK_OPT(MC_AsianAlfonsi_Heston)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_AsianAlfonsi_Heston)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static int MCAsianAlfonsi(double S0, NumFunc_2 *p, double t, double r, double d
{
    long i, ipath;
    double price_sample, delta_sample, mean_price, mean_delta, var_price, var_delt
    int init_mc;
    int simulation_dim;
    double alpha, z_alpha;
    double S_T, A_T, g1, g2;
    double h = t / (double)M;
    double sqrt_h = sqrt(h);
    double *X1a, *X2a, *X3a, *X4a;
    double w_t_1, w_t_2;
    double aaa = k * theta;
    double Kseuil, aux;
    double mu = r - divid;

    if (flag_cir == 1)
        Kseuil = MAX((0.25 * SQR(sigma) - aaa) * psik(h * 0.5, k), 0.);
    else
    {
        if (k == 0)
```

```

        Kseuil = 1;
    else Kseuil = (exp(k * h) - 1) / (h * k);
    if (sigma * sigma <= 4 * k * theta / 3)
    {

        Kseuil = Kseuil * sigma * sqrt(k * theta - sigma * sigma / 4) / sqrt(2)
    }
    if (sigma * sigma > 4 * k * theta / 3 && sigma * sigma <= 4 * k * theta)
    {
        aux = (0.5 * sigma * sqrt(3 + sqrt(6)) + sqrt(sigma * sigma / 4 - k *
        Kseuil = Kseuil * SQR(aux);
    }
    if (sigma * sigma > 4 * k * theta)
    {
        aux = 0.5 * sigma * sqrt(3 + sqrt(6)) + sqrt(sigma * sqrt(sigma * sigma
        Kseuil = Kseuil * (sigma * sigma / 4 - k * theta + SQR(aux));
    }
    if (sigma * sigma == 4 * k * theta) Kseuil = 0;
}

/*Memory allocation*/
X1a = malloc(sizeof(double) * (M + 1));
X2a = malloc(sizeof(double) * (M + 1));
X3a = malloc(sizeof(double) * (M + 1));
X4a = malloc(sizeof(double) * (M + 1));

/* 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;

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

/* MC sampling */
init_mc = pnl_rand_init(generator, simulation_dim, nb);

```

```

/* Test after initialization for the generator */
if (init_mc == OK)
{

    for (ipath = 1; ipath <= nb; ipath++)
    {
        /* Begin of the N iterations */
        X1a[0] = V0;
        X2a[0] = 0;
        X3a[0] = S0;
        X4a[0] = 0;
        for (i = 1 ; i <= M ; i++)
        {
            /*Discrete law obtained by matching of first
            five moments of a gaussian r.v.*/
            if (flag_cir == 1)
                g1 = DiscLawMatch5(generator);
            else
                g1 = DiscLawMatch7(generator);
            w_t_1 = sqrt_h * g1;

            g2 = pnl_rand_normal(generator);
            w_t_2 = sqrt_h * g2;

            X1a[i] = X1a[i - 1];
            X2a[i] = X2a[i - 1];
            X3a[i] = X3a[i - 1];
            X4a[i] = X4a[i - 1];
            fct_Heston(&X1a[i], &X2a[i], &X3a[i], &X4a[i],
                h, w_t_1, w_t_2, aaa, k, sigma, mu, rho, Kseuil, genera

        }
        /*Price*/
        A_T = 1. / t * X4a[M];
        S_T = X3a[M];
        price_sample = (p->Compute)(p->Par, S_T, A_T);

        /* Delta */
        if (price_sample > 0.0)
            delta_sample = (A_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)nb);
    *pterror_price = exp(-r * t) * sqrt(var_price / (double)nb - 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)nb);
    if ((p->Compute) == &Put_OverSpot2)
        *ptdelta *= (-1);
    *pterror_delta = sqrt(exp(-2.0 * r * t) * (var_delta / (double)nb - SQR(*ptdelta)));

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

/*Memory desallocation*/
free(X1a);
free(X2a);
free(X3a);
free(X4a);

return init_mc;
}

```

```

int CALC(MC_AsianAlfonsi_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 MCAAsianAlfonsi(ptMod->S0.Val.V_PDOUBLE,
                           ptOpt->PayOff.Val.V_NUMFUNC_2,
                           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->Par[4].Val.V_ENUM.value,
                           &(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_AsianAlfonsi_Heston)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0)
        || (strcmp(((Option *)Opt)->Name, "AsianPutFixedEuro") == 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;
        Met->Par[4].Val.V_ENUM.value = 2;
        Met->Par[4].Val.V_ENUM.members = &PremiaEnumCirOrder;
    }

    return OK;
}

PricingMethod MET(MC_AsianAlfonsi_Heston) =
{
    "MC_Alfonsi_Asian",
    { {"N iterations", LONG, {100}, ALLOW},
      {"TimeStepNumber", LONG, {100}, ALLOW},
      {"RandomGenerator", ENUM, {100}, ALLOW},
      {"Confidence Value", DOUBLE, {100}, ALLOW},
      {"Cir Order", ENUM, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_AsianAlfonsi_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_AsianAlfonsi_Heston),
    CHK_mc,
    MET(Init)
};

```