

## [Help](#)

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

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(MC_AsianFunctionalQuantization_Heston)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_AsianFunctionalQuantization_Heston)(void *Opt, void *Mod, PricingMet
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else
/* Variables globales */
static double s0, interest_rate, dividend_rate, mean_reversion_rate, mean, beta

/* Paramètres Méthode */
static int N1, dim1;          /* 1ère grille */
static int N2, dim2;          /* 2ème grille */
static double DistorX = 0.035195; /* Distortion de la 1ère grille */
static double DistorY = 0.051276; /* Distortion de la 2ème grille */

/* defined in premia_obj.c */
extern char premia_data_dir[MAX_PATH_LEN];
extern char *path_sep;

static void RK4(double *tab_gauss, int Nbpoints, int nbdim, int n, double *tab_v
{
    double k1, k2, k3, k4;
    double h;
    int i1, i2, j, k;
    double aux0, aux12, aux1, tj0, tj12, tj1;
    double auxx0, auxx12, auxx1;
    double sqrt_0, sqrt_12, sqrt_1;
    double sqrtv0, sqrtv1, sqrtv2, sqrtv3, tabij, S, S1;
    double tab_aux0, tab_aux12, tab_aux1, sqrt_rho;
```

```

double *tab_brownien0, *tab_brownien12, *tab_brownien1;

tab_brownien0 = (double *)malloc(Nbpoints * (2 * n) * sizeof(double));
tab_brownien12 = (double *)malloc(Nbpoints * (2 * n) * sizeof(double));
tab_brownien1 = (double *)malloc(Nbpoints * (2 * n) * sizeof(double));

sqrt_rho = sqrt(1.0 - rho * rho);

/* Calcul des trajectoires K-L de la dérivée du Brownien en t(j), t(j+1/2) et
h = 0.5 * Maturity / (double)n; /* 1/2 pas de temps ! */

for (j = 0; j < 2 * n; j++)
{
    tj0 = (double)j * h / Maturity;
    tj12 = tj0 + 0.5 * h / Maturity;
    tj1 = tj0 + h / Maturity;

    for (i1 = 0; i1 < Nbpoints; i1++)
    {
        tab_brownien0[ i1 + j * Nbpoints] = 0.;
        tab_brownien12[i1 + j * Nbpoints] = 0.;
        tab_brownien1[ i1 + j * Nbpoints] = 0.;

        for (k = 1; k < nbdim + 1; k++)
        {
            tab_brownien0[ i1 + j * Nbpoints] += tab_gauss[i1 + k * Nbpoints]
            tab_brownien12[i1 + j * Nbpoints] += tab_gauss[i1 + k * Nbpoints]
            tab_brownien1[ i1 + j * Nbpoints] += tab_gauss[i1 + k * Nbpoints]
        }
        tab_brownien0[ i1 + j * Nbpoints] *= sqrt(2.0 / Maturity) * M_PI / Mat
        tab_brownien12[i1 + j * Nbpoints] *= sqrt(2.0 / Maturity) * M_PI / Mat
        tab_brownien1[ i1 + j * Nbpoints] *= sqrt(2.0 / Maturity) * M_PI / Mat
    }
}
/* Fin Calculs trajectoires Browniennes */

for (i1 = 0; i1 < Nbpoints; i1++) /* Boucle sur les trajectoires de la vol Hes

```

```

{
    tab_v[i1] = v0;

    h = 0.5 * Maturity / (double)n; /* 1/2 pas de temps !

    for (j = 0; j < 2 * n; j++) /* Boucle en temps pour t -> y_i1(t)
    {
        /* Calcul de la ième trajectoire dérivé du Brownien en t(j), t(j+1/2),

        tabij = tab_v[i1 + j * Nbpoints];

        sqrtv0 = sqrt(MAX(tabij, 0.0));
        k1 = mean_reversion_rate * mean - 0.25 * beta * beta - mean_reversion_
        k1 += beta * sqrtv0 * tab_brownien0[i1 + j * Nbpoints];

        sqrtv1 = sqrt(MAX(tabij + 0.5 * h * k1, 0.0));
        k2 = mean_reversion_rate * mean - 0.25 * beta * beta - mean_reversion_
        k2 += beta * sqrtv1 * tab_brownien12[i1 + j * Nbpoints];

        sqrtv2 = sqrt(MAX(tabij + 0.5 * h * k2, 0.0));
        k3 = mean_reversion_rate * mean - 0.25 * beta * beta - mean_reversion_
        k3 += beta * sqrtv2 * tab_brownien12[i1 + j * Nbpoints];

        sqrtv3 = sqrt(MAX(tabij + h * k3, 0.0));
        k4 = mean_reversion_rate * mean - 0.25 * beta * beta - mean_reversion_
        k4 += beta * sqrtv3 * tab_brownien1[i1 + j * Nbpoints];

        tab_v[i1 + (j + 1)*Nbpoints] = tabij + h * (k1 + 2.0 * k2 + 2.0 * k3 +

    } /* Fin boucle en j (temps) Pas de temps moitié */

    for (i2 = 0; i2 < Nbpoints; i2++) /* Boucle sur les trajectoires de l'acti
    {

        S = s0;
        tab_vbar[i2 + i1 * Nbpoints] = 0.0;

        h = Maturity / (double)n; /* Pas de temps normal

        for (j = 0; j < n; j++) /* Boucle en temps pour t -> S_{i1,i2}(t)
        {

```

```

/* Calcul de la i(ème) trajectoire dérivé du Brownien en t(j), t(j)
aux0 = tab_brownien0[i1 + 2 * j * Nbpnts]; /* en i1 */
aux12 = tab_brownien0[i1 + (2 * j + 1) * Nbpnts];
aux1 = tab_brownien1[i1 + (2 * j + 1) * Nbpnts];

auxx0 = tab_brownien0[i2 + 2 * j * Nbpnts]; /* en i2 */
auxx12 = tab_brownien0[i2 + (2 * j + 1) * Nbpnts];
auxx1 = tab_brownien1[i2 + (2 * j + 1) * Nbpnts];

tab_aux0 = tab_v[i1 + 2 * j * Nbpnts];
tab_aux12 = tab_v[i1 + (2 * j + 1) * Nbpnts];
tab_aux1 = tab_v[i1 + (2 * j + 2) * Nbpnts];

sqrt_0 = sqrt(tab_aux0);
sqrt_12 = sqrt(tab_aux12);
sqrt_1 = sqrt(tab_aux1);

/* printf("%f %f %f \ n",aux0,aux12,aux1); */

k1 = interest_rate - dividend_rate - 0.5 * tab_aux0 - 0.25 * rho
k2 = interest_rate - dividend_rate - 0.5 * tab_aux12 - 0.25 * rho
k3 = interest_rate - dividend_rate - 0.5 * tab_aux12 - 0.25 * rho
k4 = interest_rate - dividend_rate - 0.5 * tab_aux1 - 0.25 * rho

k1 *= S;
k2 *= S + 0.5 * h * k1;
k3 *= S + 0.5 * h * k2;
k4 *= S + h * k3;

S1 = S + h * (k1 + 2.0 * k2 + 2.0 * k3 + k4) / (double)6.0;

tab_vbar[i2 + i1 * Nbpnts] += h * S + h * h * (k1 + k2 + k3) / 6

S = S1;
} /* Fin boucle en temps j */

tab_vbar[i2 + i1 * Nbpnts] /= Maturity;

} /* Fin boucle i2 */
} /* Fin boucle i1 */

```

```

    free(tab_brownien0);
    free(tab_brownien12);
    free(tab_brownien1);
}

static int MCAAsianFunctionalQuantization(double x0,
    NumFunc_2 *p,
    double T,
    double r,
    double divid,
    double y0,
    double alpha,
    double theta,
    double _beta,
    double _rho,
    int flag_opti, int n_steps,
    double *ptprice,
    double *ptdelta)
{
    double K;

    double *tab_gauss1, *tab_gauss2;
    int d, i1, i2;
    char idfile1[MAX_PATH_LEN], idfile2[MAX_PATH_LEN];
    double *tab_v_1, *tab_v_2, *tab_vbar_1, *tab_vbar_2;

    double CallX, CallY, PutX, PutY, DeltaCX, DeltaCY, DeltaPX, DeltaPY;
    double Call_Romberg, Put_Romberg, Call_parite, Put_parite;
    double DeltaC_Romb, DeltaP_Romb, DeltaC_par, DeltaP_par;
    double Truemoneyness, Kmin, Kmax;
    double CallPrice, PutPrice, DeltaCall, DeltaPut;
    FILE *tab_gauss1_fp, *tab_gauss2_fp;

    /* 1ère grille */
    if (flag_opti == 0)
    {
        /* 1ère grille */
        N1 = 966;
        dim1 = 4;
        /* 2ème grille */
        N2 = 96;
    }
}

```

```

        dim2 = 3;
    }
else if (flag_opti == 1)
{
    /* 1ère grille */
    N1 = 400;
    dim1 = 6;
    /* 2ème grille */
    N2 = 100;
    dim2 = 4;
}

/* Initialisation des variables globales */
s0 = x0;
Maturity = T;
interest_rate = r;
dividend_rate = divid;
v0 = y0;
mean_reversion_rate = alpha;
mean = theta;
beta = _beta;
rho = _rho;
K = p->Par[0].Val.V_DOUBLE;

tab_gauss1 = (double *)malloc(N1 * (dim1 + 1) * sizeof(double));
tab_gauss2 = (double *)malloc(N2 * (dim2 + 1) * sizeof(double));

/* Lecture de la première grille N1 */
/* 400.txt or 966.txt in in Premia/data */
sprintf(idfile1, "%s%s%d.txt", premia_data_dir, path_sep, N1);
tab_gauss1_fp = fopen(idfile1, "r");

for (i1 = 0; i1 < N1; i1++)
{
    for (d = 0; d < dim1; d++)
    {
        fscanf(tab_gauss1_fp, "%lf", tab_gauss1 + d * N1 + i1);
    }
    fscanf(tab_gauss1_fp, "%lf", tab_gauss1 + dim1 * N1 + i1);
}

```

```

fclose(tab_gauss1_fp);

/* Lecture de la deuxième grille N2 */
/* 100.txt or 96.txt in Premia/data */
sprintf(idfile2, "%s%s%d.txt", premia_data_dir, path_sep, N2);
tab_gauss2_fp = fopen(idfile2, "r");

for (i2 = 0; i2 < N2; i2++)
{
    for (d = 0; d < dim2; d++)
    {
        fscanf(tab_gauss2_fp, "%lf", tab_gauss2 + d * N2 + i2);
    }
    fscanf(tab_gauss2_fp, "%lf", tab_gauss2 + dim2 * N2 + i2);
}

fclose(tab_gauss2_fp);

/* Allocations de tableaux pour le calcul des trajectoires de la vol Heston */
tab_v_1 = (double *)malloc(N1 * (2 * n_steps + 1) * sizeof(double));
tab_vbar_1 = (double *)malloc(N1 * N1 * sizeof(double));

tab_v_2 = (double *)malloc(N2 * (2 * n_steps + 1) * sizeof(double));
tab_vbar_2 = (double *)malloc(N2 * N2 * sizeof(double));

/* Calcul des trajectoires de la vol Heston + celles de l'actif */
RK4(tab_gauss1, N1, dim1, n_steps, tab_v_1, tab_vbar_1);
RK4(tab_gauss2, N2, dim2, n_steps, tab_v_2, tab_vbar_2);

/* Calcul du prix de l'option et du delta */
if (fabs(dividend_rate - interest_rate) < 0.0001)
{
    Truemoneyness = s0;
}
else
{
    Truemoneyness = s0 * (exp((interest_rate - dividend_rate) * T) - 1.0) / ((
}

Kmax = MAX(K, 1.05 * Truemoneyness);

```

```

Kmin = MIN(K, 0.95 * Truemoneyness);

/* Pour la 1ère grille */
CallX = 0.;
PutX = 0.;
DeltaCX = 0.;
DeltaPX = 0.;
for (i1 = 0; i1 < N1; i1++)
{
    for (i2 = 0; i2 < N1; i2++)
    {
        CallX += tab_gauss1[i1] * tab_gauss1[i2] * MAX(tab_vbar_1[i2 + i1 * N1]
        PutX  += tab_gauss1[i1] * tab_gauss1[i2] * MAX(K - tab_vbar_1[i2 + i1 *

        if (tab_vbar_1[i2 + i1 * N1] > K)
        {
            DeltaCX += tab_gauss1[i1] * tab_gauss1[i2] * tab_vbar_1[i2 + i1 *
        }
        else
        {
            DeltaPX += tab_gauss1[i1] * tab_gauss1[i2] * tab_vbar_1[i2 + i1 *
        }
    }
}

/* Pour la 2ème grille */
CallY = 0.;
PutY = 0.;
DeltaCY = 0.;
DeltaPY = 0.;
for (i1 = 0; i1 < N2; i1++)
{
    for (i2 = 0; i2 < N2; i2++)
    {
        CallY += tab_gauss2[i1] * tab_gauss2[i2] * MAX(tab_vbar_2[i2 + i1 * N2]
        PutY  += tab_gauss2[i1] * tab_gauss2[i2] * MAX(K - tab_vbar_2[i2 + i1 *

```



```

        if (tab_vbar_2[i2 + i1 * N2] > K)
        {
            DeltaCY += tab_gauss2[i1] * tab_gauss2[i2] * tab_vbar_2[i2 + i1 *
        }
    else
    {
        DeltaPY += tab_gauss2[i1] * tab_gauss2[i2] * tab_vbar_2[i2 + i1 *
    }
}

}

CallX *= exp(-interest_rate * Maturity);
PutX *= exp(-interest_rate * Maturity);
CallY *= exp(-interest_rate * Maturity);
PutY *= exp(-interest_rate * Maturity);

DeltaCX *= exp(-interest_rate * Maturity);
DeltaCY *= exp(-interest_rate * Maturity);
DeltaPX *= -exp(-interest_rate * Maturity);
DeltaPY *= -exp(-interest_rate * Maturity);

if (flag_opti == 1)
{
    /* Romberg log-extrapolation */
    Call_Romberg = (CallX * log((double)N1) - CallY * log((double)N2)) / (log(
    Put_Romberg = (PutX * log((double)N1) - PutY * log((double)N2)) / (log(
    DeltaC_Romb = (DeltaCX * log((double)N1) - DeltaCY * log((double)N2)) / (
    DeltaP_Romb = (DeltaPX * log((double)N1) - DeltaPY * log((double)N2)) / (
}
else /*if(flag_opti==0)*/
{
    /* Romberg distorsion-extrapolation */
    Call_Romberg = (CallX / DistorX - CallY / DistorY) / (1.0 / DistorX - 1.0 /
    Put_Romberg = (PutX / DistorX - PutY / DistorY) / (1.0 / DistorX - 1.0 /
    DeltaC_Romb = (DeltaCX / DistorX - DeltaCY / DistorY) / (1.0 / DistorX -
    DeltaP_Romb = (DeltaPX / DistorX - DeltaPY / DistorY) / (1.0 / DistorX -
}

if (r != divid)
{

```

```

        Call_parite = Put_Romberg + s0 * exp(-dividend_rate * Maturity) * (1.0 -
        DeltaC_par = DeltaP_Romb + exp(-dividend_rate * Maturity) * (1.0 -
        Put_parite = Call_Romberg - s0 * exp(-dividend_rate * Maturity) * (1.0 -
        DeltaP_par = DeltaC_Romb - exp(-dividend_rate * Maturity) * (1.0 -
    }
else /*if(r==divid)*/
{
    Call_parite = Put_Romberg + s0 * exp(-dividend_rate * Maturity) - K * e
    DeltaC_par = DeltaP_Romb + exp(-dividend_rate * Maturity);
    Put_parite = Call_Romberg - s0 * exp(-dividend_rate * Maturity) + K * e
    DeltaP_par = DeltaC_Romb - exp(-dividend_rate * Maturity);
}

/* K-interpolation linéaire */
CallPrice = ((K - Kmin) * Call_Romberg + (Kmax - K) * Call_parite) / (Kmax - Kmin)
PutPrice = ((K - Kmin) * Put_parite + (Kmax - K) * Put_Romberg) / (Kmax - Kmin)

DeltaCall = ((K - Kmin) * DeltaC_Romb + (Kmax - K) * DeltaC_par) / (Kmax - Kmin)
DeltaPut = ((K - Kmin) * DeltaP_par + (Kmax - K) * DeltaP_Romb) / (Kmax - Kmin)

if ((p->Compute) == &Call_OverSpot2)
{
    /* Price estimator */
    *ptprice = CallPrice;
    /* Delta estimator */
    *ptdelta = DeltaCall;
}
else
{
    /* Price estimator */
    *ptprice = PutPrice;
    /* Delta estimator */
    *ptdelta = DeltaPut;
}

free(tab_gauss1);
free(tab_gauss2);
free(tab_v_1);
free(tab_v_2);
free(tab_vbar_1);

```

```

    free(tab_vbar_2);
    return 0;
}

int CALC(MC_AasianFunctionalQuantization_Heston)(void *Opt, void *Mod, PricingMet
{
    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 MCAsianFunctionalQuantization(ptMod->S0.Val.V_PDOUBLE,
                                         ptOpt->PayOff.Val.V_NUMFUNC_2,
                                         ptOpt->Maturity.Val.V_DATE - ptMod->T.Val
                                         ptMod->LongRunVariance.Val.V_PDOUBLE,
                                         ptMod->Sigma.Val.V_PDOUBLE,
                                         ptMod->Rho.Val.V_PDOUBLE,
                                         Met->Par[0].Val.V_ENUM.value,
                                         Met->Par[1].Val.V_INT,
                                         &(Met->Res[0].Val.V_DOUBLE),
                                         &(Met->Res[1].Val.V_DOUBLE));
}

static int CHK_OPT(MC_AasianFunctionalQuantization_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 PremiaEnumMember OptFlagMembers[] =
{
    { "Product Quantizer", 0 },

```

```

    { "optimal Quantizer", 1 },
    { NULL, NULLINT }
};

static DEFINE_ENUM(OptFlag, OptFlagMembers);

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;

        Met->Par[0].Val.V_ENUM.value = 1;
        Met->Par[0].Val.V_ENUM.members = &OptFlag;
        Met->Par[1].Val.V_INT = 16;
    }

    return OK;
}

PricingMethod MET(MC_AasianFunctionalQuantization_Heston) =
{
    "MC_AasianFC_Hes",
    {
        {"Optimal Flag", ENUM, {100}, ALLOW},
        {"TimeStepNumber", LONG, {100}, ALLOW},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_AasianFunctionalQuantization_Heston),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(MC_AasianFunctionalQuantization_Heston),
    CHK_ok,
    MET(Init)
};

```