

[Help](#)

```

/*
  Author: Syoiti Ninomiya
  Tokyo Institute of Technology
  Implementation of Ninomyia-Victoir paper "Weak approximation of stochastic dif
*/

#include "hes1d_pad.h"

/*****
/* */
/* */
*****/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
static int CHK_OPT(MC_AsianKusuoka_Heston)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(MC_AsianKusuoka_Heston)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else
static double *expV0(double s, double *initial, double *destination,
                    double mu, double rho, double alpha, double beta,
                    double theta)
{
    double J, A;

    J = theta - 0.25 * beta * beta / alpha;
    A = mu - 0.25 * rho * beta - 0.5 * initial[1];
    destination[0] =
        initial[0] * exp((mu - rho * beta * 0.25 - 0.5 * J) * s
            + (initial[1] - J) * 0.5 / alpha * (exp(-alpha * s) - 1.0))
    destination[1] = J + (initial[1] - J) * exp(-alpha * s);

```

```

    destination[2] = initial[2] + initial[0] * (exp(A * s) - 1.0) / A;
    return destination;
}

```

```

static double *expV1(double s, double *initial, double *destination,
                    double mu, double rho, double alpha, double beta,
                    double theta)
{
    double X, sqrt_y2;

    sqrt_y2 = sqrt(initial[1]);
    X = 0.5 * rho * beta * s + sqrt_y2;
    destination[0] =
        initial[0] * exp(s * (0.25 * rho * beta * s + sqrt_y2));
    destination[1] = X * X;
    destination[2] = initial[2];
    return destination;
}

```

```

static double *expV2(double s, double *initial, double *destination,
                    double mu, double rho, double alpha, double beta,
                    double theta)
{
    double X;

    X = 0.5 * sqrt(1.0 - rho * rho) * beta * s + sqrt(initial[1]);
    destination[0] = initial[0];
    destination[1] = X * X;
    destination[2] = initial[2];
    return destination;
}

```

```

static int MCAAsianKusuoka(double x0, NumFunc_2 *p, double T, double r, double d
{
    double K, mu, dt, sq_dt;
    double *u_seq1, *u_seq2, *n_seq1, *n_seq2, *ber_seq;

    K = p->Par[0].Val.V_DOUBLE;
    mu = r - divid;
    dt = T / (double)n_steps;

```

```

sq_dt = sqrt(dt);

u_seq1 = (double *)calloc(3 * n_steps, sizeof(double));
u_seq2 = u_seq1 + n_steps;
ber_seq = u_seq1 + 2 * n_steps;
n_seq1 = (double *)calloc(2 * n_steps, sizeof(double));
n_seq2 = n_seq1 + n_steps;

{
    double sum, x[4][3], dsum, dx[4][3];
    double *last = NULL, *dlast = NULL;
    long int i;
    int j;

    for (dsum = sum = 0.0, i = 0; i < niter; i++)
    {
        b2_g_sobol_seq("G_SOBOL_1", 3 * n_steps, u_seq1);
        {
            int k;
            for (k = 0; k < n_steps; k++)
            {
                n_seq1[k] = sqrt(-2.0 * log(u_seq1[k])) * cos(2.0 * M_PI * u_seq2[k]);
                n_seq2[k] = sqrt(-2.0 * log(u_seq1[k])) * sin(2.0 * M_PI * u_seq2[k]);
            } /** for (k) **/
        }
        for (x[0][0] = x0, dx[0][1] = x[0][1] = y0, dx[0][2] = x[0][2] = 0.0,
            dx[0][0] = x0 * (1.0 + inc), j = 0;
            j < n_steps; j++)
        {
            /* int k;*/
            if (ber_seq[j] > 0.5)
            {
                last =
                    expV0(0.5 * dt,
                        expV1(sq_dt * n_seq1[j],
                            expV2(sq_dt * n_seq2[j],
                                expV0(0.5 * dt,
                                    x[0], x[1], mu, rho, alpha, beta, theta),
                                    x[2], mu, rho, alpha, beta, theta),
                                    x[3], mu, rho, alpha, beta, theta),
                                x[0], mu, rho, alpha, beta, theta);
            }
        }
    }
}

```

```

        dlast =
            expV0(0.5 * dt,
                expV1(sq_dt * n_seq1[j],
                    expV2(sq_dt * n_seq2[j],
                        expV0(0.5 * dt,
                            dx[0], dx[1], mu, rho, alpha, beta, theta),
                            dx[2], mu, rho, alpha, beta, theta),
                            dx[3], mu, rho, alpha, beta, theta),
                        dx[0], mu, rho, alpha, beta, theta);
    }
else    /** ber_seq[j] <= 0.5 */
{
    last =
        expV0(0.5 * dt,
            expV2(sq_dt * n_seq1[j],
                expV1(sq_dt * n_seq2[j],
                    expV0(0.5 * dt,
                        x[0], x[1], mu, rho, alpha, beta, theta),
                        x[2], mu, rho, alpha, beta, theta),
                        x[3], mu, rho, alpha, beta, theta),
                    x[0], mu, rho, alpha, beta, theta);
    dlast =
        expV0(0.5 * dt,
            expV2(sq_dt * n_seq1[j],
                expV1(sq_dt * n_seq2[j],
                    expV0(0.5 * dt,
                        dx[0], dx[1], mu, rho, alpha, beta, theta),
                        dx[2], mu, rho, alpha, beta, theta),
                        dx[3], mu, rho, alpha, beta, theta),
                    dx[0], mu, rho, alpha, beta, theta);
    }
} /** for (j) */
if ((p->Compute) == &Call_OverSpot2)
{
    sum += (last[2] / (double)T - K > 0) ? last[2] / (double)T - K : 0;
    dsum += (dlast[2] / (double)T - K > 0) ? dlast[2] / (double)T - K : 0;
}
else
{
    if ((p->Compute) == &Put_OverSpot2)
    {

```

```

        sum += (K - last[2] / (double)T > 0) ? K - last[2] / (double)T : 0;
        dsum += (K - dlast[2] / (double)T > 0) ? K - dlast[2] / (double)T : 0;
    }
}

} /** for (i) **/

*ptprice = exp(-r * T) * sum / (double)niter;
*ptdelta = exp(-r * T) * (dsum - sum) / (double)niter / inc / x0;
}

free(u_seq1);
free(n_seq1);
b2_g_sobol_free();

return OK;
}

int CALC(MC_AasianKusuoka_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 MCAsianKusuoka(ptMod->S0.Val.V_PDOUBLE,
        ptOpt->PayOff.Val.V_NUMFUNC_2,
        ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE, r, divid,
        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_DOUBLE,
        &(Met->Res[0].Val.V_DOUBLE),
        &(Met->Res[1].Val.V_DOUBLE));
}

```

```

static int CHK_OPT(MC_AsianKusuoka_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)
{
    if (Met->init == 0)
    {
        Met->init = 1;

        Met->Par[0].Val.V_LONG = 10000;
        Met->Par[1].Val.V_INT = 100;
        Met->Par[2].Val.V_PDOUBLE = 0.001;

    }

    return OK;
}

PricingMethod MET(MC_AsianKusuoka_Heston) =
{
    "MC_Asian_NV_Hes",
    { {"N iterations", LONG, {100}, ALLOW},
      {"TimeStepNumber", LONG, {100}, ALLOW},
      {"Delta Increment Rel", PDOUBLE, {100}, ALLOW},
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(MC_AsianKusuoka_Heston),
    { {"Price", DOUBLE, {100}, FORBID},
      {"Delta", DOUBLE, {100}, FORBID} ,
      {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },

```

```
CHK_OPT(MC_AsianKusuoka_Heston),  
CHK_mc,  
MET(Init)  
};
```