

[Help](#)

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
#else

#ifndef heston_kusuoka_h_
#define heston_kusuoka_h_

#include "
href../../mod/hes1d/hes1d_pad/function_heston2_h_src.pdffunction_heston2.h"
#include "
href../../mod/hes1d/hes1d_pad/montecarlo2_h_src.pdfmontecarlo2.h"

using namespace std;

//_alpha mean reversion
//_beta volatility of volatility
//_theta log-run variance
//_nu annual interest rate
//_rho correlation coefficient
//_K strike
//_T maturity
//_x0 spot
//_y0 current variance
//_r discount factor
//_steps number of partition
//_niter number of trajectory

//_flag_var_control =0 We don't use the method of variance reduction
// =1 We use the method of variance reduction

//_flag_schema =0 Euler Scheme
// =1 Romberg Extrapolation for Euler Scheme
// =2 Ninomiya-Victoir Scheme

//_ndelta delta (derive of the option price)
//_nerror error of the result
//_nerror_delta delta error

int heston_kusuoka(int _generator, double _alpha, double _beta, double _theta, d
int _flag_var_control, int _flag_schema, double &nprice, dou
```

```

{

double epsilon = DBL_EPSILON;

//error of date
if (_T < epsilon || 2.*_alpha * _theta - _beta * _beta < -epsilon)
    return 0;

//we cannot apply Ninomiya-Victoir scheme for brownian motion correlated
//so in this case we set _flag_scheme=Euler Scheme
if ((abs(_rho) > epsilon) & (_flag_scheme == 2))
    _flag_scheme = 0;

//random variable initialization
// rnd_init();

//model dimension (the number of the equations differential stochastic)
int ndim_vector = (_flag_var_control == 0) ? 3 : 5;

//x0 - vector initial
vector<double> x0(ndim_vector);

x0[0] = _x0;
x0[1] = _y0;
x0[2] = 0.;

if (_flag_var_control != 0)
{
    x0[3] = _x0;
    x0[4] = 0.;
}

//model (heston or heston with variance reduction)
model *ptr_model = (_flag_var_control == 0) ? (model *) new model_heston(_alpha, _beta, _theta)
    : (model *) new model_heston_var_control(_alpha, _beta, _theta);

//random variable vector
rv_vector *ptr_rv_vector = new rv_vector_heston(_rho, ndim_vector, _generator,

```

```

double nres = 0., ncorr = 0., nres2 = 0., nerror2 = 0.;
_nerror = 0.;
_ndelta = 0.;
_nerror_delta = 0.;

//discretisation scheme (0 - Euler Scheme, 1 - Euler Extrapolation, 2 - Ninom
typedef vector<double> (*ptr_scheme)(rv_vector *, model *, int, int);
ptr_scheme scheme = ((_flag_scheme == 0) || (_flag_scheme == 1)) ? scheme_eule

//without variance reduction
if (_flag_var_control == 0)
{
    nres = monte_carlo(_steps, _niter, ptr_rv_vector, f_asian, f_asian_delta,
}
//variance reduction
else
{
    nres = monte_carlo2(_steps, _niter, ptr_rv_vector, f_asian, f_asian_delta,
}

//if we have choose extrapolation then we must have two results for construct
//(first - Euler Scheme; second - Euler Scheme with double step (2*steps))
if (_flag_scheme == 1)
{
    if (_flag_var_control == 0)
        nres2 = monte_carlo(2 * _steps, _niter, ptr_rv_vector, f_asian, f_asian_delta,
    else
        nres2 = monte_carlo2(2 * _steps, _niter, ptr_rv_vector, f_asian, f_asian_delta,

    nres = 2.*nres2 - nres;
    _nerror = 4.*nerror2 + _nerror;
}

_nprice = exp(-_r * _T) * nres;
_nerror = exp(-2.*_r * _T) * _nerror;
_ndelta = exp(-_r * _T) * _ndelta;
_nerror_delta = exp(-2.*_r * _T) * _nerror_delta;

// cout<<"res="<<nres<<" error="<< _nerror<<" delta="<<_ndelta<<" nerror_delta="<<_nerror_delta<<endl;

```

```
//free memory
delete ptr_rv_vector;
delete ptr_model;

return OK;
}

#endif
#endif //PremiaCurrentVersion
```