

[Help](#)

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

#include "model.h"

#include <vector>

#ifndef model_heston_h_
#define model_heston_h_

//heston model class (without a variance reduction technique)
class model_heston: public model
{
public:

    //constructor
    //the parameters of heston model
    model_heston(double _alpha, double _beta, double _theta, double _nu, double _r
    {
        alpha = _alpha;
        beta = _beta;
        theta = _theta;
        nu = _nu;
        rho = _rho;
        K = _K;
        T = _T;
        x0 = _x0;
    };

    double alpha;
    double beta;
    double theta;
    double nu;
    double rho;

    //functions for a Ninomiya-Victoir schema
    virtual std::vector<double> exp_V0(double, std::vector<double>);
```

```

virtual std::vector<double> exp_V1(double, std::vector<double>);
virtual std::vector<double> exp_V2(double, std::vector<double>);
virtual std::vector<double> f_1(std::vector<double>, double, std::vector<double>);
virtual std::vector<double> f_2(std::vector<double>, double, std::vector<double>);

//functions for an Euler schema
virtual std::vector<double> f_b(std::vector<double>, double);
virtual std::vector<double> f_sigma(std::vector<double>, double);

//functions for a variance reduction technique
virtual double f_control(std::vector<double>)
{
    return 0.;
};
virtual double f_esp(double &)
{
    return 0.;
};

};

//heston model class
//with this class we apply a variance reduction technique
class model_heston_var_control: public model_heston
{
public:

    //constructors
    model_heston_var_control(double _alpha, double _beta, double _theta, double _n)
    {};
    model_heston_var_control(model_heston *_ptr): model_heston(_ptr->alpha, _ptr->beta, _ptr->theta, _ptr->n)
    {};

    std::vector<double> exp_V0(double, std::vector<double>);
    std::vector<double> exp_V1(double, std::vector<double>);
    std::vector<double> exp_V2(double, std::vector<double>);
    std::vector<double> f_b(std::vector<double>, double);
    std::vector<double> f_sigma(std::vector<double>, double);
    std::vector<double> f_1(std::vector<double>, double, std::vector<double>);

```

```

std::vector<double> f_2(std::vector<double>, double, std::vector<double>);

//control variable
double f_control(std::vector<double>);

//mean of a control variable
double f_esp(double &);

};

class rv_vector_heston: public rv_vector
{
public:
    rv_vector_heston(double _ncorr, int _ndim, int _generator, int _nred_var): rv_
    {
        ncorr = ((_ncorr <= 1.) & (_ncorr >= -1.)) ? _ncorr : 0.;
        generator = _generator;
        nred_var = _nred_var;
    };

    virtual std::vector<double> get_rv(void)
    {
        std::vector<double> nres(ndim_vector);
        nres[0] = pnl_rand_normal(generator);
        nres[1] = pnl_rand_normal(generator);
        nres[1] = sqrt(1. - ncorr * ncorr) * nres[1] + ncorr * nres[0];
        nres[2] = 0.;
        if (nred_var != 0)
        {
            nres[3] = nres[0];
            nres[4] = 0.;
        }
        return nres;
    };

private:
    double ncorr;
    int generator;

    int nred_var;
};

```

```
//asian option, payoff
double f_asian(std::vector<double> _x, model *_ptr_model)
{
    double epsilon = DBL_EPSILON;
    return (_x[2] / _ptr_model->T - _ptr_model->K > epsilon) ? _x[2] / _ptr_model->T : 0;
}

//asian option, delta
double f_asian_delta(std::vector<double> _x, model *_ptr_model)
{
    double epsilon = DBL_EPSILON;
    int nindicator = (_x[2] / _ptr_model->T - _ptr_model->K > epsilon) ? 1 : 0;
    double ndelta = 0.;

    if ((nindicator == 1) & (std::abs(_ptr_model->x0[0]) > epsilon))
        ndelta = _x[2] / (_ptr_model->T * _ptr_model->x0[0]);

    return ndelta;
}

#endif
#endif //PremiaCurrentVersion
```