

# Heston Stochastic Volatility Model

## Demo workspace walkthrough

## Table of Contents

The Heston Stochastic Volatility Model .....	2
The five parameters of the Heston Model.....	3
The role of the different parameters.....	3
$\nu_0$ .....	3
$\theta$ .....	3
$\rho$ .....	3
$\kappa$ .....	4
$\sigma$ .....	4
The Feller Condition .....	4
Workspace tab “Calibration of the Heston Model” .....	5
The Calibration Class.....	8
Workspace Tab “Heston Option Pricing” .....	10
The Option Pricing and Implied Volatility calculation functions.....	11
Workspace Tab “Heston Implied Volatility Smile” .....	13
Workspace Tab “Option Price Sensitivity to Heston Parameters” .....	14
Workspace Tab “Black Volatility Sensitivity to Heston Parameters” .....	15
Workspace Tab “Simulating Heston, one path” .....	16
The Simulation Class .....	16
Workspace Tab “Simulating Heston, ten paths” .....	19
Workspace Tab “Simulated Vanilla Option Prices .....	20
Workspace Tab “Simulated Barrier Option Prices” .....	21

## The Heston Stochastic Volatility Model

The Heston Stochastic volatility model was introduced by Steven Heston in 1993 in an attempt to provide a more realistic model for assets than the well known Black-Scholes (BS) model.

Simply put, it replaces the constant volatility of the BS model by a stochastic volatility which is modelled by a separate process.

The processes are given by

$$\begin{aligned}\frac{dS(t)}{S(t)} &= \mu(t)dt + \sqrt{V(t)}dW_1(t) \\ dV(t) &= \kappa(\theta - V(t))dt + \sigma\sqrt{V(t)}dW_2(t),\end{aligned}$$

where  $W_1(t)$  and  $W_2(t)$  are standard Brownian motions with instantaneous correlation given by

$$\langle dW_1, dW_2 \rangle = \rho dt$$

and  $\mu(t)$  is the drift term, typically the risk free rate (with any dividend yield subtracted).

The Heston model depends on five free parameters:

#### The five parameters of the Heston Model

$v_0$  = The initial value for the variance process  $V(t)$ ,  $v_0 = V(0)$ .

$\theta$  = The long term mean for the variance process  $V(t)$ .

$\rho$  = The instantaneous correlation between the Brownian motions for the asset and variance process.

$\kappa$  = The mean reversion speed. How fast the variance process tends to its long term mean.

$\sigma$  = The volatility of the variance process. Often called vol-vol.

Those parameters have to be calibrated from available market data (in the form of Black volatilities for different maturities and strikes).

#### The role of the different parameters

##### $v_0$

Controls the initial vertical position of the implied Black volatility smile. Larger value means larger initial volatilities.

Allowed values: All positive values are theoretically allowed, even though values too close to zero is not recommended. Typically this value is less than 1.

##### $\theta$

Controls the vertical position of the long term implied Black volatility smile. Larger value means larger long term volatilities.

Allowed values: All positive values are theoretically allowed, even though values too close to zero is not recommended. Typically this value is less than 1.

##### $\rho$

Controls the skew of the smiles. The value 0 gives the most symmetrical smile. Negative and positive values skews the smile in different directions.

Allowed values: All values (strictly) between -1 and 1 theoretically allowed, even though values too close to -1 or 1 are not recommended. In real application this correlation is typically negative.

## $\kappa$

A larger  $\kappa$  means a flatter smile since the variance process then tends to follow its mean more closely.

Allowed values: All positive values are theoretically allowed, even though values too close to zero is not recommended. Also too large values of this parameter is not recommended. By large we here mean around 10 or higher.

The reason is that the model in general becomes very insensitive to changes in this parameter for larger values.

## $\sigma$

Controls the kurtosis of the smile, or more simply put the “happiness” of the smile. A larger sigma gives a more pronounced smile. A smaller sigma gives a flatter smile.

Allowed values: All positive values are theoretically allowed, even though values too close to zero is not recommended. Typically this value is less than 3.

### **Remark:**

We may note that the effect of the parameters  $\kappa$  and  $\sigma$  both overlap a bit. We can make the smile flatter by either increasing  $\kappa$  or by decreasing  $\sigma$ . This makes the calibration a bit numerically problematic since a large  $\kappa$  combined with a large  $\sigma$  can produce a very similar looking smile compared to a small  $\kappa$  combined with a small  $\sigma$ .

This is not a huge problem in practice though, since we are not really interested in the Heston parameters as such, but rather how well the implied volatility smile fits the market data.

## The Feller Condition

The variance process can hit zero if the so called Feller condition

$$2\kappa\theta > \sigma^2$$

is not satisfied. It is not an absorbing state though, so the process don't stick to zero in that case.

The Feller condition is often fulfilled in real application, but sometimes it is not.

This condition is not a requirement in any way, but in general the simulation of the Heston model will typically become less accurate if this condition is violated by a large margin.

## Workspace tab “Calibration of the Heston Model”

This is a screenshot of the contents of the table in first tab in the Workspace. The purpose of this tab is to demonstrate the calibration of the Heston model using some simplified input.

Remark: When using the workspace for the first time. Press recalc twice (after providing the input).

Calibrate Heston model						
Common parameters		Strikes (% of forward value)	Black Volatility, maturity 1	Black Volatility, maturity 2	Black Volatility, maturity 3	Heston parameter names
Calibration Target	option_price	50%	0.500000000	0.400000000	0.370000000	v0
Calibration Weighting	vega_weights	60%	0.450000000	0.420000000	0.400000000	theta
Initial Guess v0	0.2	70%	0.440000000	0.390000000	0.400000000	rho
Initial Guess theta	0.2	80%	0.400000000	0.370000000	0.340000000	kappa
Initial Guess rho	0	90%	0.380000000	0.360000000	0.340000000	sigma
Initial Guess kappa	1	100%	0.370000000	0.330000000	0.350000000	
Initial Guess sigma	1	110%	0.350000000	0.320000000	0.330000000	
Time to maturity 1	0.5	120%	0.360000000	0.320000000	0.320000000	
Time to maturity 2	1	130%	0.360000000	0.330000000	0.300000000	
Time to maturity 3	1.5	140%	0.390000000	0.340000000	0.310000000	
Forward value 1	100	150%	0.400000000	0.350000000	0.310000000	
Forward value 2	100	Implied Black Vol from Heston				
Forward value 3	100	50%	0.499312448	0.426174046	0.390479138	
		60%	0.463006355	0.400044503	0.371317927	
		70%	0.431308131	0.378529588	0.360800999	
		80%	0.404467801	0.361592081	0.344217671	
		90%	0.383516059	0.348828869	0.335351468	
		100%	0.369725436	0.340209478	0.329137188	
		110%	0.363520831	0.335378419	0.325207164	
		120%	0.363732480	0.333715465	0.323165186	
		130%	0.368254464	0.334463068	0.322615582	
		140%	0.375173420	0.336897786	0.323197509	
		150%	0.383209250	0.340433123	0.324606748	

The calibrated Heston parameters are seen in the two columns on the far right.

Column 2 to 4 contains Black volatilities (expressed as absolute numbers, that is 0.45 for 45% volatility and so on) used as input for the calibration. The upper part of the table of these three columns have editable entries where the user can change the values.

The lower part of the same columns contains the calculated implied Black volatilities using the calibrated parameters in the last column. The point of this is to provide a reference to compare with the upper part of the table. The better match the better the calibrations.

Since the model only have five parameters to calibrate we can of course not expect a very good match to market data here.

The input parameters expected from the user(except the Black volatilities) are on the left side of the table:

### Calibration Target :

This specifies an enum:

```
// Enum to describe what target quantity should be matched in the Heston calibration.
// We can compare option prices or implied Black volatilities.
enum calibration_target option(category : "Finance/Heston-Model") { OPTION_PRICE option(str : "option_price"), VOLATILITY option(str : "volatility"); }
```

Simply put, in our calibration procedure we can choose to compare market and model option prices or implied volatilities.

### Calibration weighting :

This specifies an enum:

```
// Enum to describe how we choose the weights.
// If custom weights are chosen, all those those custom weights needs to be provided.
enum calibration_weights option(category : "Finance/Heston-Model") { CUSTOM option(str : "custom"),
..... EQUAL_WEIGHTS option(str : "equal_weights"),
..... EQUAL_TOTAL_WEIGHT_PER_MATURITY option(str : "equal_total_weight_per_maturity"),
..... VEGA_WEIGHTS option(str : "vega_weights"); }
```

Internally, we use a least squares optimizer where we minimize the sum of square errors of model option prices to market option prices (or volatilities if that option is chosen).

We can choose to put weights in front of these square terms. The choices are the following:

CUSTOM = The user can specify a vector of weights for each option freely when adding market data to the calibration class. This is not used in this workspace though, so this option has been excluded here.

EQUAL\_WEIGHTS = No weights/all weights equal to 1.

EQUAL\_TOTAL\_WEIGHT\_PER\_MATURITY = The sum of all weights for all terms belonging to options for a specific maturity adds up to the same number for all maturities. This can be useful if we have a different number of options for each maturity, but we want each maturity be equally important.

VEGA\_WEIGHTS = Here we construct weights based on the option Vega. This is only used when we compare option prices. Using these weights emulates the situation if we would compare volatilities with equal weights instead.

This is typically the desired choice as it gives a very good fit generally.

**Technical remark:**

More precisely. For each option define

$$v_i = \min\left(\frac{1}{V_i}, \frac{1000}{F_i}\right)$$
$$w_i = \frac{v_i^2}{\sum_j v_j^2}$$

where the sum is over all options (numbered by j) and  $V_i$  is the vega of option i and  $F_i$  is the forward value of the asset for the maturity of option i.

$w_i$  is now the weight we use for option i.

**Initial guess v0, Initial guess theta, Initial guess rho, Initial guess kappa, Initial guess sigma :**

The optimizing procedure for the Heston model takes an initial guess for the Heston parameters as starting point. It can happen that the calibration procedure “get lost” if this guess is chosen badly. The above entries are the initial guesses for all the Heston variables.

The above starting values are quite typical and typically there is no need to change them.

A few loose guidelines for choosing “good” values are:

The initial v0, theta, kappa or sigma should not be chosen too close to 0 (the end point of their domain of definition).

Rho must be between -1 and 1, but the starting guess should not be too close to the endpoints.

v0, theta, kappa or sigma should not be chosen too large as starting guess. v0, theta and sigma should preferably be chosen less than 1 and kappa less than say 5.

**Time to maturity 1, Time to maturity 2, Time to maturity 3 :**

The table has editable cells where the user can enter the Black volatilities used as input for the calibration. We use three columns here, each one representing data for one maturity.

Those three parameters tells the calibration procedure which maturities the black volatilities of the three columns represent.

#### **Forward value 1, Forward value 2, Forward value 3 :**

Here the forward values for the corresponding maturities are specified.

#### **The Calibration Class**

We now aim to describe the code used behind the scenes to create this tab and to calibrate the Heston model in general.

Create an instance of the calibration class:

```
heston_calibration h = heston_calibration();
```

Now assume we have some market data at the maturity  $T=1$ . Assume the forward value for the underlying to maturity  $T$  is  $F=100$ . Assume we have strikes at 80, 100 and 120 and the corresponding Black volatilities 45%, 40%, 41%.

This we can add to our calibration class:

```
number T = 1;
number F = 100;
vector(number) strikes = [70, 100, 120];
vector(number) black_vol = [0.45, 0.40, 0.41];

h.add_data_for_maturity(T, F, strikes, black_vol);
```

This is then repeated for all maturities  $T$ , where we have market data.

At least two maturities, and preferably more, should be used. This is needed to reflect the time behavior of the model and its parameters.

We also have an alternative version of the above function. We can decide how much weight we should put in front of each squared error term in the least squares procedure. In this way we can explicitly tell the class that some option data are more important than others to match.

Take the same example as before, but want the options to be weighted by 0.5, 1 and 2 respectively.

Then we specify:

```
number T = 1;
number F = 100;
vector(number) strikes = [70, 100, 120];
vector(number) black_vol = [0.45, 0.40, 0.41];
vector(number) weights = [0.5, 1, 2];

h.add_data_for_maturity(T, F, strikes, black_vol, weights);
```

If such custom weights are used for one maturity it is required to use custom weights for all maturities.



When all data has been added we start the calibration we simply calling the function “calibrate”. There are two versions of this. One which defaults all arguments of the other to suitable values. The signatures of the functions are

```
heston_params calibrate();  
heston_params calibrate(calibration_target target,  
                        calibration_weights weight_type,  
                        vector(number) start_guess,  
                        number option_val_rel_tol = 0.0000000001,  
                        number alpha_search_abs_tol = 0.000001);
```

The first two arguments are exactly those enums just described for the workspace. If custom weights have been used to fill the class with market data, the input argument **target** must equal CUSTOM. **start\_guess** = [v0, theta, rho, kappa, sigma] is a vector containing the start guess of the Heston parameters as described earlier, in the given order.

**option\_val\_rel\_tol** is a relative error tolerance for the internal option price calculation procedure. This is defaulted if not given.

**alpha\_search\_abs\_tol** is another internal absolute error tolerance when searching for a dampening parameter alpha used in the option pricing. This is defaulted and should probably be left that way.

So let us choose the initial guesses as v0=0.2, theta = 0.3, rho=0, kappa = 1 and sigma = 0.8. Let us assume we want to compare option prices using vega weighting :

```
vector(number) start_guess = [0.2, 0.3, 0, 1, 0.8];  
heston_params result = h.calibrate(OPTION_PRICE, VEGA_WEIGHTS, start_guess);
```

We get a Heston\_params object as a result. This contains our calibrated parameters. We can ask for the calibrated parameters by using its member functions :

```
number v0 = result.v0();  
number theta = result.theta();  
number rho = result.rho();  
number kappa = result.kappa();  
number sigma = result.sigma();
```

## Workspace Tab “Heston Option Pricing”

This is a screenshot of the second tab which is used to demonstrate option pricing in the Heston model.

There are three tables. One table for pricing a single option and calculate its implied Black volatility, one table showing a grid of option prices for different strikes and maturities and one table showing a grid of implied volatilities for different strikes and maturities.

The screenshot displays the Heston Option Pricing workspace with three main tables and a parameter input section.

**Single Option Price and Implied Volatility**

Common parameters	Heston Option Price	Implied Black Volatility
$v_0$ 0.2 $\theta$ 0.3 $\rho$ -0.3 $\kappa$ 2 $\sigma$ 1 Strike 100 Forward Value 100 Maturity 1 Discount Factor 1 Is Cal <input checked="" type="checkbox"/>	13.556440749	0.341461221

**Heston Implied Black Volatilities**

Common parameters	Strikes (% of forward value)	Black Volatility, Maturity 1	Black Volatility, Maturity 2	Black Volatility, Maturity 3
$v_0$ 0.2 $\theta$ 0.3 $\rho$ -0.3 $\kappa$ 2 $\sigma$ 1 Time to Maturity 1 1 Time to Maturity 2 2 Time to Maturity 3 3 Forward Value, maturity 1 100 Forward Value, maturity 2 100 Forward Value, maturity 3 100	50%	0.450269320	0.44032373	0.43034352
	60%	0.45720358	0.44334090	0.43131238
	70%	0.430351299	0.40965273	0.40437370
	80%	0.41016034	0.39440217	0.39787591
	90%	0.39772226	0.38116568	0.38363877
	100	0.39344818	0.39027006	0.39254944
	110%	0.39777834	0.39116269	0.39347380
	120%	0.40914102	0.39878944	0.39979181
	130%	0.41933969	0.40131051	0.39943699
	140%	0.42762626	0.40773857	0.40146876
150%	0.43062914	0.41404568	0.40746871	

**Option Prices (undiscounted)**

Common parameters	Strikes (% of forward value)	Option Price, Maturity 1	Option Price, Maturity 2	Option Price, Maturity 3
$v_0$ 0.2 $\theta$ 0.3 $\rho$ -0.3 $\kappa$ 2 $\sigma$ 0.8 Is Cal <input checked="" type="checkbox"/> Time to Maturity 1 1 Time to Maturity 2 2 Time to Maturity 3 3 Forward Value, maturity 1 100 Forward Value, maturity 2 100 Forward Value, maturity 3 100	50%	51.400967940	54.19232820	57.037624948
	60%	42.833013480	46.942772937	50.786277750
	70%	35.63392080	40.54174269	45.26690729
	80%	28.324375774	34.988948320	40.387939195
	90%	22.645212847	30.23804437	36.204870693
	100	18.943803115	26.21480008	32.585110096
	110%	14.40498398	22.82878984	29.438702395
	120%	11.5721715	19.98351085	26.712106306
	130%	9.378627919	17.93001047	24.338218658
	140%	7.67628507	15.57636200	22.287380286
150%	6.34638455	13.87049084	20.451592011	

The input to all of the tables is similar. We describe the single option table first. The input entries look like this:

**Single Option Price and Implied Volatility**

Common parameters

v0: 0.2

theta: 0.1

rho: -0.3

kappa: 2

sigma: 1

Strike: 100

Forward Value: 100

Maturity: 1

Discount Factor: 1

Is Call: ☒

Recalc

Recalc

The first five entries are simply the parameters for the Heston model as described before.

The other entries are also self explanatory: We have the strike of the option, the forward value of the underlying, the maturity of the option, the discount factor from maturity and finally we can choose to price a call or put option.

The input for the other two tables is almost the same, but we specify three maturities and the corresponding forward values at those maturities.

### The Option Pricing and Implied Volatility calculation functions

Here we describe the functions used for this workspace tab.

Let us say we want to price a call option in the Heston model where  $v_0=0.2$ ,  $\theta = 0.3$ ,  $\rho = -0.2$ ,  $\kappa = 1$ ,  $\sigma = 0.7$ ,  $\text{strike} = 100$ ,  $\text{forward\_value} = 95$ ,  $\text{maturity} = 1$  (year) and  $\text{discount factor} = 0.98$ . Then we simply call the pricing function:

```
number v0      = 0.2;
number theta   = 0.3;
number rho     = -0.2;
number kappa   = 1;
number sigma   = 0.7;

number strike  = 100;
number forward_value = 95;
number maturity = 1;
number df      = 0.98;
logical is_call = true;

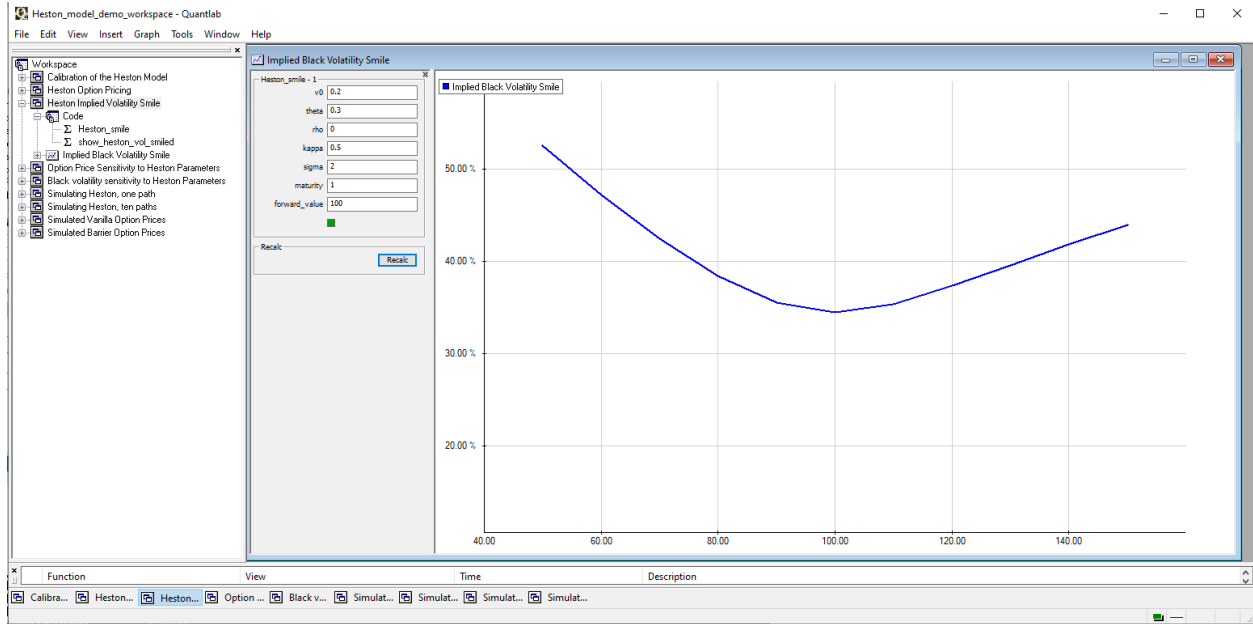
number option_price = heston_option_price(v0, theta, rho, kappa, sigma, strike, forward_value, maturity, df, is_call)
```

If we want to calculate the implied Black volatility we call the function doing that (using the same values of the parameters as above):

```
number implied_black_vol = heston_implied_black_volatility(v0, theta, rho, kappa, sigma, strike, forward_value, maturity);
```

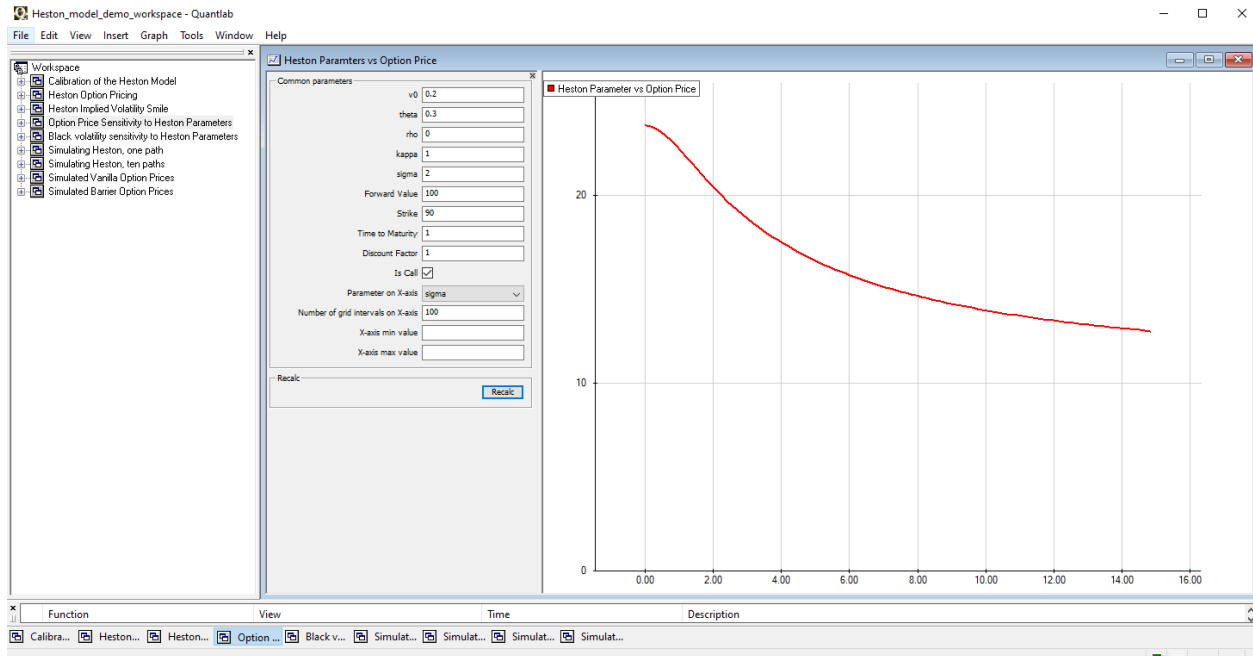


## Workspace Tab “Heston Implied Volatility Smile”



The purpose of this is to show the implied Black volatility smile given the five Heston parameters together with the time to maturity and the forward value of the underlying. The same code as in the last tab is used, so nothing new code-wise here.

## Workspace Tab “Option Price Sensitivity to Heston Parameters”



This tab can give a lot of insight in how the different parameters of the Heston model influence the option price. The first input entries in this table is the, by now, well known Heston parameters, the forward value, time to maturity, discount factor and a flag `is_call` for switching between call and put options.

The graph shows the option price as a function of a chosen input parameter. All input parameters will be fixed, as given by the user, except one, which is the one we put on the x-axis.

So if we choose “**Parameter on X-axis**” as sigma. The graph shows how the option price depends on sigma (The value of sigma given by the user in the “sigma” box is then ignored).

We can put all Heston parameters on the X-axis, but also the strike, forward value and time to maturity.

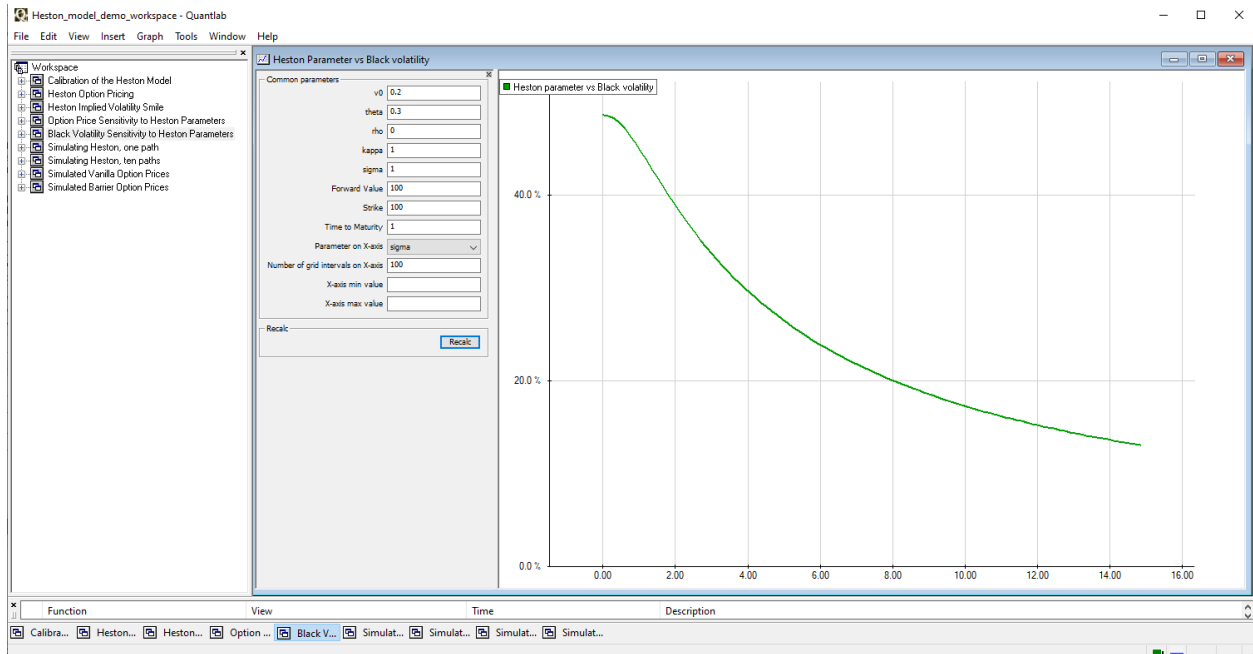
The entry “**Number of grid intervals on X-axis**” determines how many intervals and grid points we will have on the X-axis. Typically around 100 gives a smooth enough graph.

Then we have two optional entries “**X-axis min value**” and “**X-axis max value**”.

Those specify the left and right endpoints in the X-axis. If not given they will be defaulted.

Those are useful if we want to zoom in on a particular part of interest of the graph.

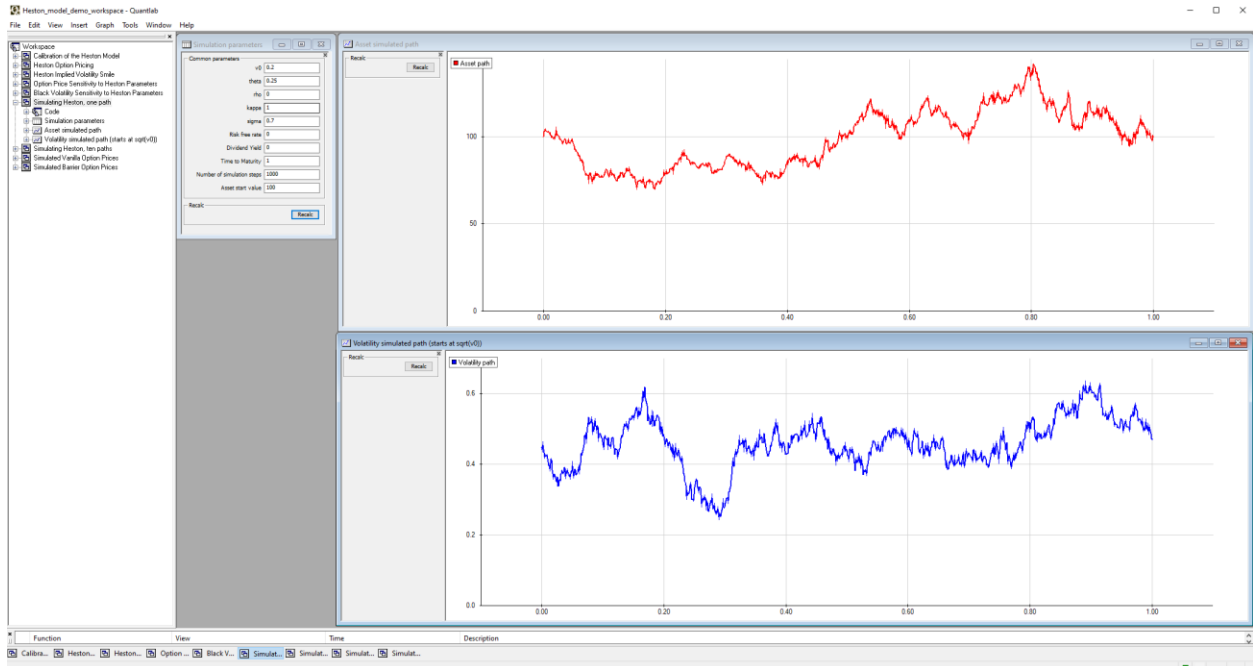
## Workspace Tab “Black Volatility Sensitivity to Heston Parameters”



The input to this table is identical to the table last tab: “Option Price Sensitivity to Heston Parameters”.

The only difference is that the graph shows Black implied volatility instead of option price.

## Workspace Tab “Simulating Heston, one path”



This tab simulates the Heston model using one sample path and present the result in two graphs. The first graph is the asset itself and the second the volatility process (square root of the variance process).

The input is the usual five Heston parameters, but also a constant risk free rate and dividend yield used for the drift of the asset. We also choose a time to maturity here, meaning where the simulation will stop.

We also need to provide a start value for the asset and the number of simulation time steps to use.

One may note the dependence of the asset on the volatility graphically. When the volatility graph goes down, the asset tend to be less volatile, as expected.

### The Simulation Class

Let us say we want to do a 1 year long simulation of an asset with initial value = 100 using Heston parameters  $v_0=0.2$ ,  $\theta = 0.3$ ,  $\rho = -0.2$ ,  $\kappa = 1$ ,  $\sigma = 0.7$  using 1000 sample paths and 100 time steps. Let us assume the risk free rate is 0.02 and the dividend yield is 0.01.

The code for creating a simulator and doing the simulation would then look like this.



```

number v0      = 0.2;
number theta   = 0.23;
number rho     = -0.2;
number kappa   = 1;
number sigma   = 0.7;

integer n_paths      = 1000;
number initial_asset_value = 100;

heston_simulation s = heston_simulation(n_paths, initial_asset_value, v0, theta, rho, kappa, sigma, QE);

number simulation_length = 1;
integer n_time_steps     = 100;
number rate              = 0.02;
number div_yield         = 0.01;
number drift             = rate - div_yield;

integer seed = millisecond(now());
rng my_rng = rng(seed);

s.simulate(simulation_length, n_time_steps, drift, my_rng);

```

Note that we needed to create a provide a random number generator to the simulation function.

The last argument QE in the constructor represents the internal simulation model to use. This last argument is an enum describing different simulation models.

```
enum heston_sim_model option(category : "Finance/Heston Model") {TV option(str : "tv"), QE option(str : "qe")};
```

TV stands for “Transformed Volatility” scheme. It is a simple and very fast simulation scheme. However, it seems somewhat biased in some situations.

QE stand for “Quadratic Exponential” Scheme. This is the recommended scheme which is the default.

This scheme performs well, especially when using so called martingale corrections.

The last argument could be skipped and safely be defaulted to the QE scheme.

The simulation class has a few member function to obtain the simulated samples:

```

vector(number) asset_values();
vector(number) log_asset_values();

vector(number) volatility_values();
vector(number) variance_values();

number current_time();

```

**asset\_values()** Returns the simulated values for the asset.

**log\_asset\_values()** Returns the logarithm of the simulated values for the asset.

**volatility\_values()** Returns the simulated volatilities (square root of the variances).

**variance\_values()** Returns the simulated variances (squared volatilities).

**current\_time()** Returns where in time the simulation is at the moment.

The point in having the four getter functions instead of two for the asset and its variance process is to avoid doing unnecessary exponentials, logarithms, square roots and squares if it is not necessary. If we call the simulate function many times this can be costly.

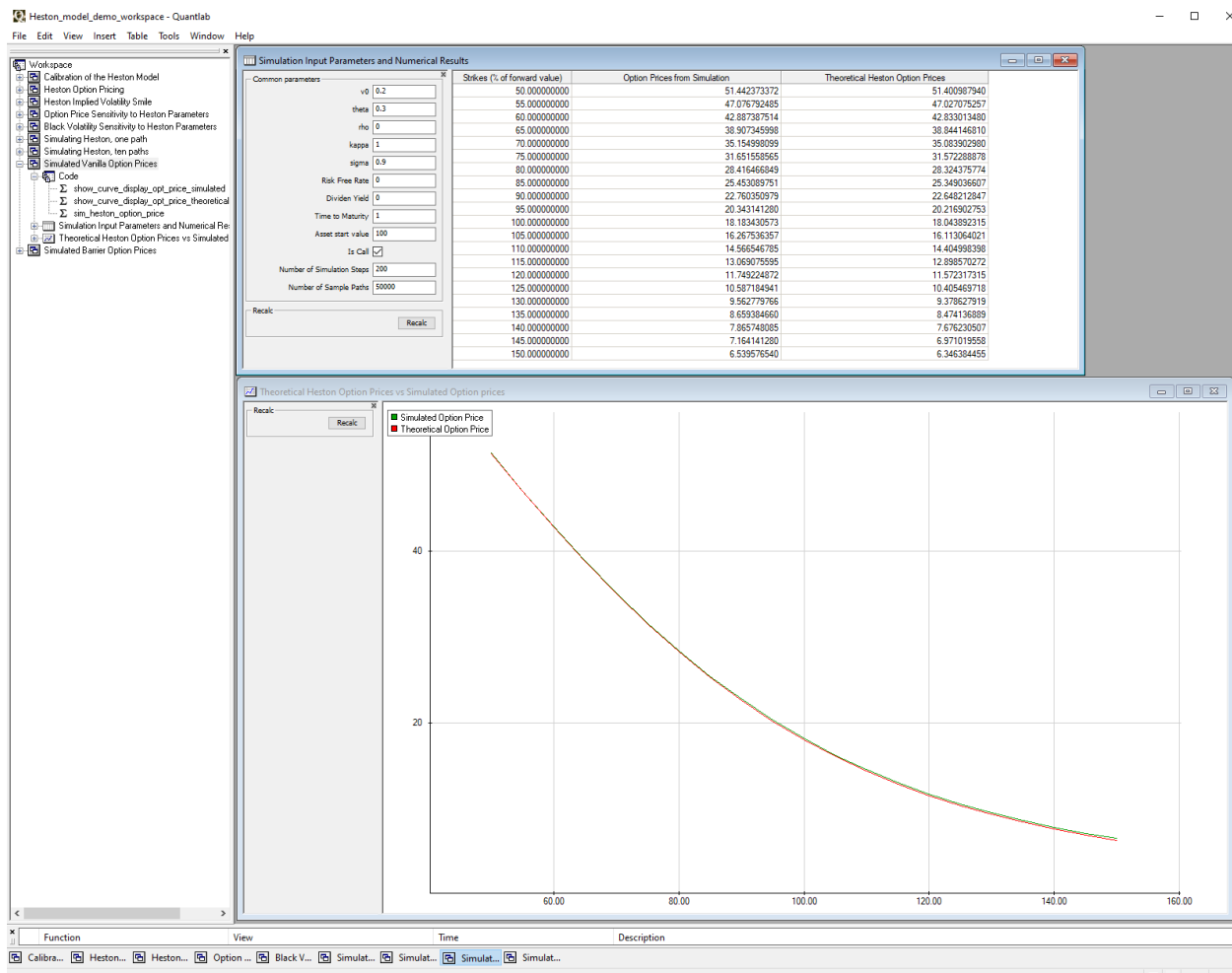
## Workspace Tab “Simulating Heston, ten paths”



This tab is identical to the previous tab, except that we simulate ten paths for both the asset and volatility.

The point is to get a bit of understanding how much different paths differ from each other.

## Workspace Tab “Simulated Vanilla Option Prices



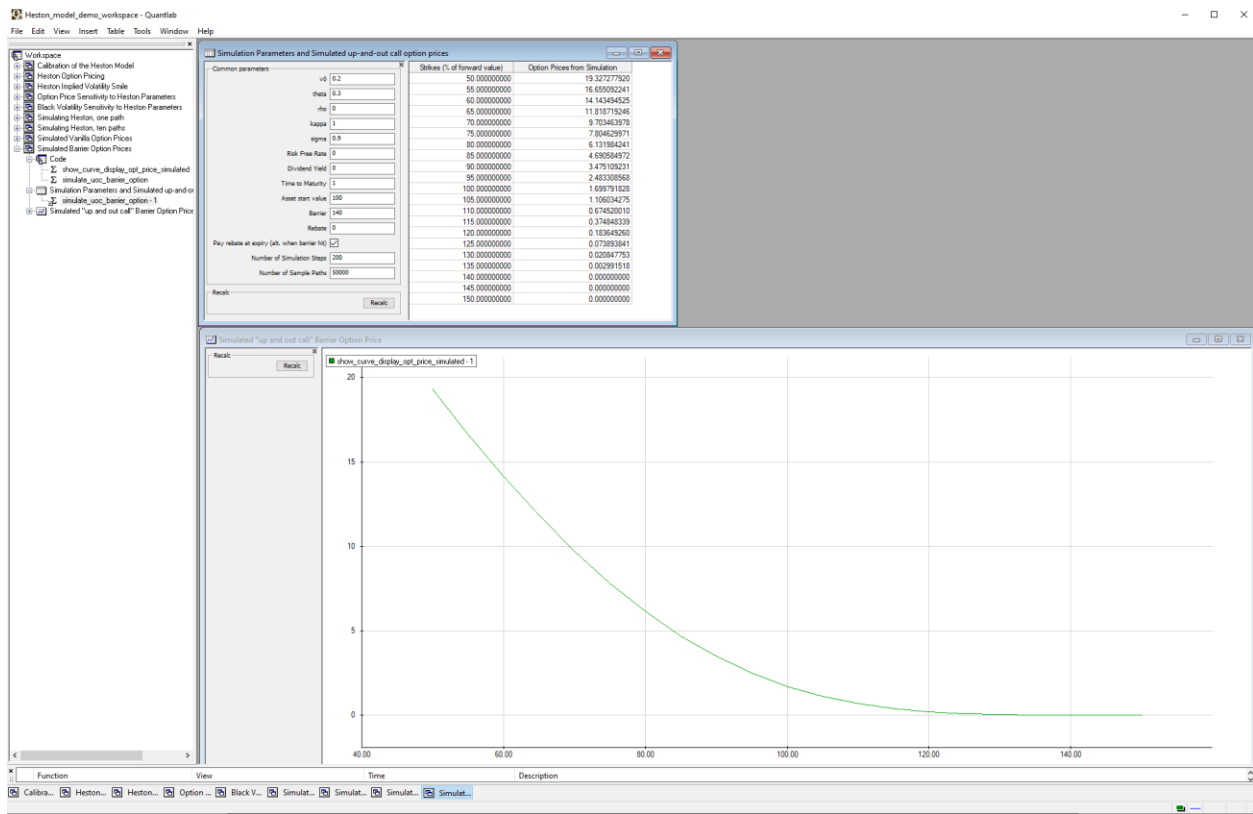
The point of this tab is to price vanilla call/put options in the Heston model and to compare those to the theoretical option prices.

The theoretical option prices are the correct prices in the model. So we can study the quality of the simulation here.

The input is the same as in the “Simulated Heston, one path” and “Simulated Heston, ten paths” tabs. The output is the simulated and theoretical option prices in the Heston model, both in numerical and in the form of graphs.

The simulated option price is computed the usual Monte Carlo way by averaging the payoff over all samples.

## Workspace Tab “Simulated Barrier Option Prices”



The point of this tab is to show how the Heston simulation class can be used to price something slightly more complicate, a barrier up-and-out call option with rebate.

This is an ordinary vanilla European call option, but with the extra feature that option expires and becomes worthless if the simulated asset ever goes above the barrier level. If that happens a rebate is paid (which can be 0).

The rebate can be paid either when the barrier is hit or at expiry.

The input here is the usual one, except for the obvious new entries where we can set the barrier, the rebate and if it is paid on hit or expiry.

This is an example where we need to split the simulation interval in many sub steps and check if the barrier has been hit in each of those steps.

Here there we have the obvious problem that the barrier might have been hit between steps.

However, one can calculate that probability and adapt the calculations to that.

We don't go into details of the math here.