Algorithmica
Research AB

Heston Stochastic Volatility Model

Demo workspace walkthrough

Last update: 2021-03-02

Table of Contents

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

$$\frac{dS(t)}{S(t)} = \mu(t)dt + \sqrt{V(t)}dW_1(t)$$

$$dV(t) = \kappa\big(\theta - V(t)\big)dt + \sigma\sqrt{V(t)}dW_2(t),$$

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

$$< dW_1, dW_2 > = \rho dt$$

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), $v_0 = V(0)$.

θ = The long term mean for the variance process V(t).

ρ = The instantaneous correlation between the Brownian motions for the asset and variance process.

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

σ = 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.

θ

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.

ρ

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.

κ

A larger κ 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.

σ

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 κ and σ both overlap a bit. We can make the smile flatter by either increasing κ or by decreasing σ. This makes the calibration a bit numerically problematic since a large κ combined with a large σ can produce a very similar looking smile compared to a small κ combined with a small σ.
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

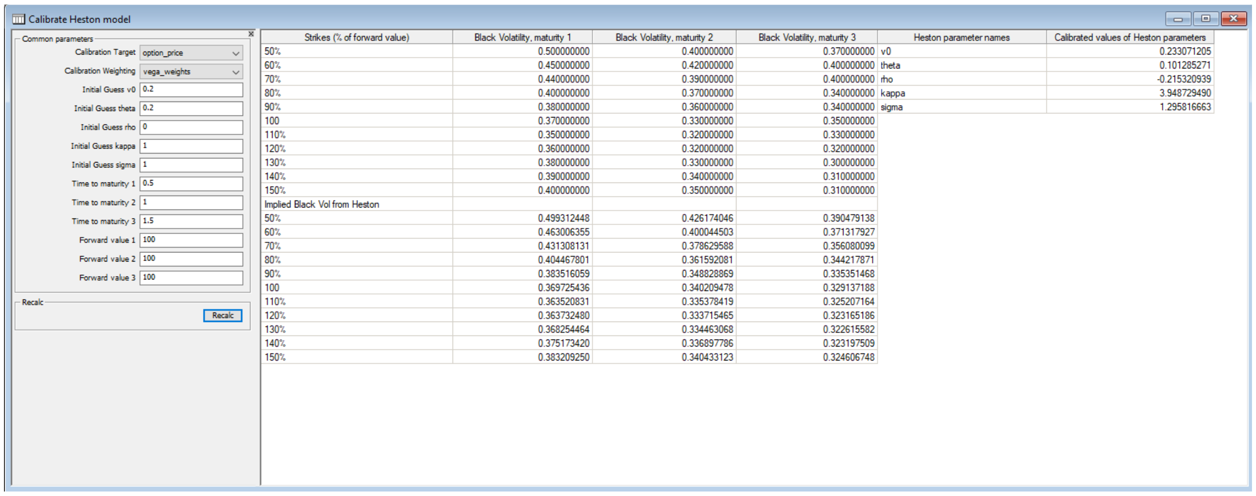
$$2\kappa\theta > \sigma^2$$

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 be 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).



Common parameters

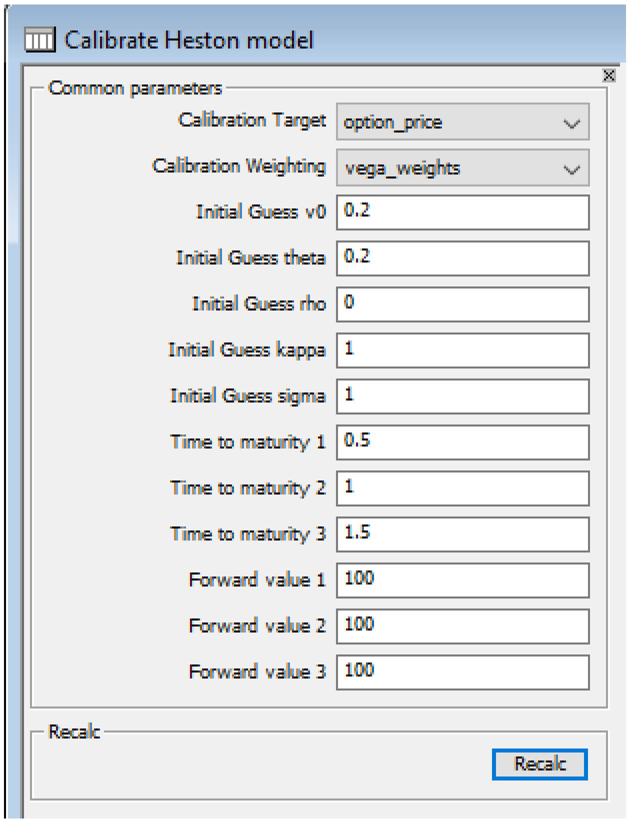
Calibration Target	option_price
Calibration Weighting	vega_weights
Initial Guess v0	0.2
Initial Guess theta	0.2
Initial Guess rho	0
Initial Guess kappa	1
Initial Guess sigma	1
Time to maturity 1	0.5
Time to maturity 2	1
Time to maturity 3	1.5
Forward value 1	100
Forward value 2	100
Forward value 3	100

Strikes (% of forward value)	Black Volatility, maturity 1	Black Volatility, maturity 2	Black Volatility, maturity 3	Heston parameter names	Calibrated values of Heston parameters
50%	0.500000000	0.400000000	0.370000000	v0	0.233071205
60%	0.450000000	0.420000000	0.400000000	theta	0.101285271
70%	0.440000000	0.390000000	0.400000000	rho	-0.215320939
80%	0.400000000	0.370000000	0.340000000	kappa	3.948729490
90%	0.380000000	0.360000000	0.340000000	sigma	1.295816663
100	0.370000000	0.330000000	0.350000000		
110%	0.350000000	0.320000000	0.330000000		
120%	0.360000000	0.320000000	0.320000000		
130%	0.380000000	0.330000000	0.300000000		
140%	0.390000000	0.340000000	0.310000000		
150%	0.400000000	0.350000000	0.310000000		
Implied Black Vol from Heston					
50%	0.499312448	0.426174046	0.390479138		
60%	0.463006355	0.400044503	0.371317927		
70%	0.431308131	0.378629588	0.356080099		
80%	0.404467801	0.361592081	0.344217871		
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:



Calibrate Heston model

Common parameters

Calibration Target	option_price
Calibration Weighting	vega_weights
Initial Guess v0	0.2
Initial Guess theta	0.2
Initial Guess rho	0
Initial Guess kappa	1
Initial Guess sigma	1
Time to maturity 1	0.5
Time to maturity 2	1
Time to maturity 3	1.5
Forward value 1	100
Forward value 2	100
Forward value 3	100

Recalc

Calibration Target :

This specifies an enum:

```
// Enum to decribe 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 choosen 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 preferrably by 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 = 0.00000000001,
                        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);
```

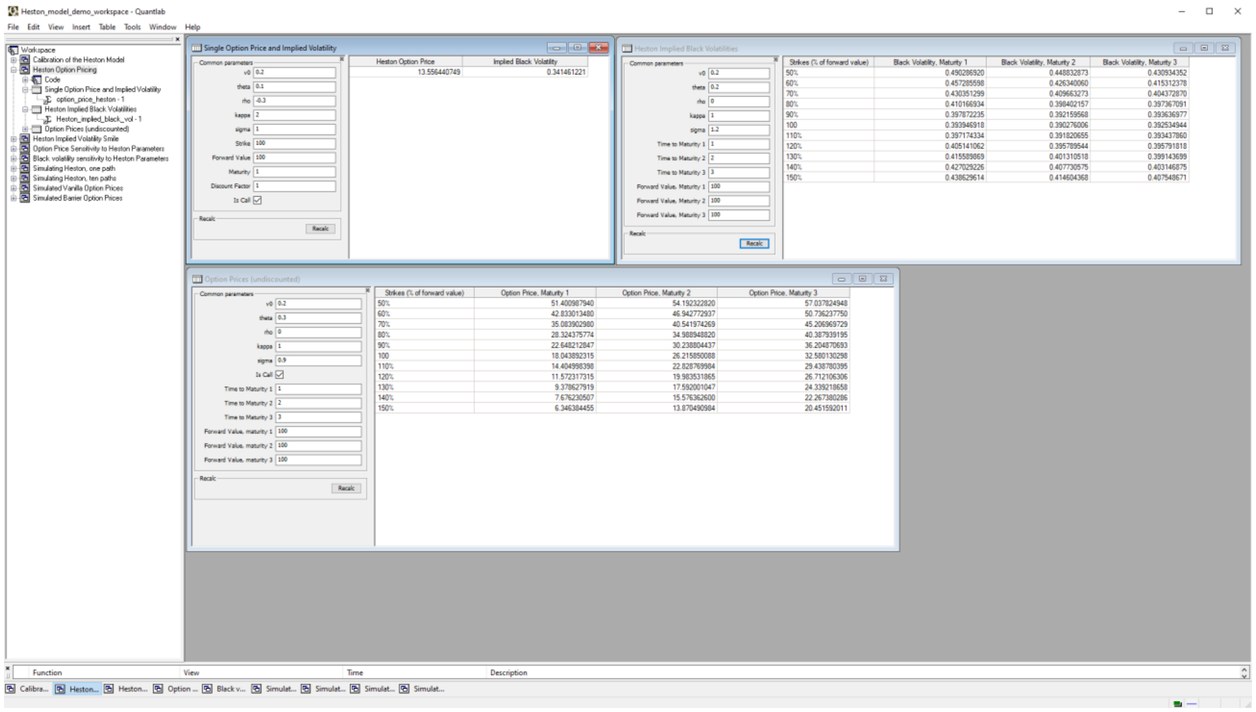
We get a Heston_params object as a result. This contains our calibrated parameters. We can ask for the calibrated parameters by using its member functions :

```
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.


Heston_model_demo_workspace - Quantlab
File Edit View Insert Table Tools Window Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Code
Single Option Price and Implied Volatility
option_price_heston - 1
Heston Implied Black Volatilities
Heston_implied_black_vol - 1
Option Prices (undiscounted)
Heston Implied Volatility Smile
Option Price Sensitivity to Heston Parameters
Black volatility sensitivity to Heston Parameters
Simulating Heston, one path
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Simulated Barrier Option Prices
Recalc
Function
View
Time
Description
Calibra...
Heston...
Heston...
Option ...
Black v...
Simulat...
Simulat...
Simulat...
Simulat...

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	✓

Heston Option Price	Implied Black Volatility
13.556440749	0.341461221

Heston Implied Black Volatilities

Common parameters

v0	0.2
theta	0.2
rho	0
kappa	1
sigma	1.2
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

Strikes (% of forward value)	Black Volatility, Maturity 1	Black Volatility, Maturity 2	Black Volatility, Maturity 3
50%	0.490286920	0.448832873	0.430934352
60%	0.457285598	0.426340060	0.415312378
70%	0.430351299	0.409663273	0.404372870
80%	0.410166934	0.398402157	0.397367091
90%	0.397872235	0.392159568	0.393636977
100	0.393946918	0.390276006	0.392534944
110%	0.397174334	0.391820655	0.393437860
120%	0.405141062	0.395789544	0.395791818
130%	0.415589869	0.401310518	0.399143699
140%	0.427029226	0.407730575	0.403146875
150%	0.438629614	0.414604368	0.407548671

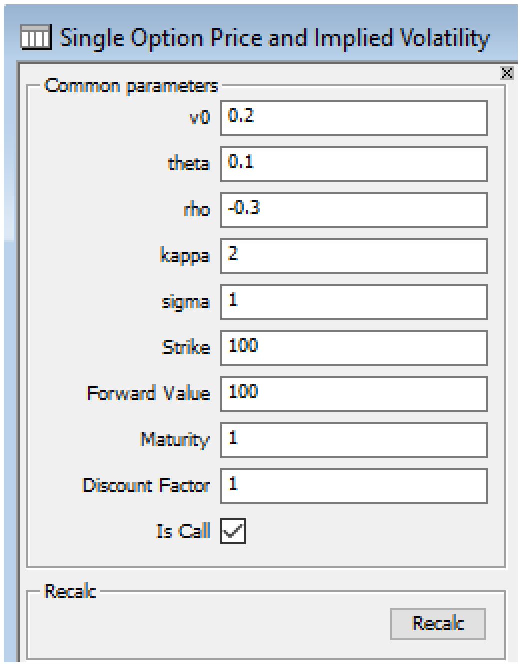
Option Prices (undiscounted)

Common parameters

v0	0.2
theta	0.3
rho	0
kappa	1
sigma	0.9
Is Call	✓
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

Strikes (% of forward value)	Option Price, Maturity 1	Option Price, Maturity 2	Option Price, Maturity 3
50%	51.400987940	54.192322820	57.037824948
60%	42.833013480	46.942772937	50.736237750
70%	35.083902980	40.541974269	45.206969729
80%	28.324375774	34.988948820	40.387939195
90%	22.648212847	30.238804437	36.204870693
100	18.043892315	26.215850088	32.580130298
110%	14.404998398	22.828769984	29.438780395
120%	11.572317315	19.983531865	26.712106306
130%	9.378627919	17.592001047	24.339218658
140%	7.676230507	15.576362600	22.267380286
150%	6.346384455	13.870490984	20.451592011

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



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	✓

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.

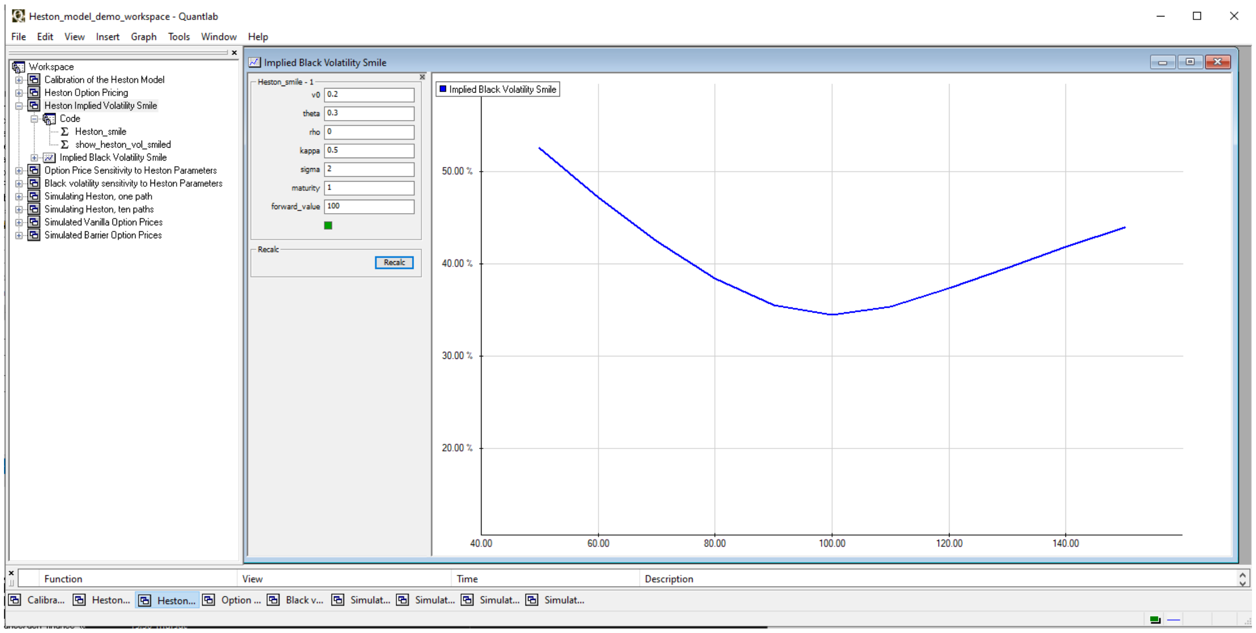
Let us say we want to price a call option in the Heston model where v0=0.2, theta = 0.3, rho = -0.2, kappa = 1, sigma = 0.7, strike = 100, forward_value = 95, maturity = 1 (year) and discount factor = 0.98. Then we simply call the pricing function:

```
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);
```

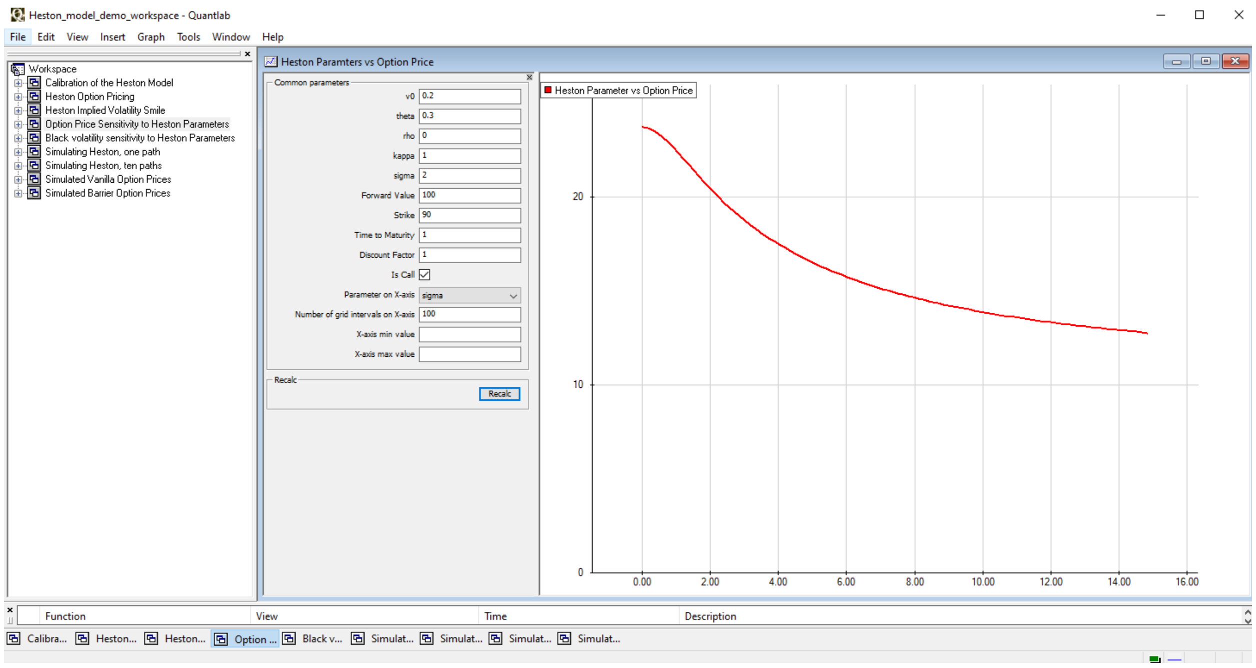
Workspace Tab “Heston Implied Volatility Smile”


Heston_model_demo_workspace - Quantlab
File Edit View Insert Graph Tools Window Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Heston Implied Volatility Smile
Code
Heston_smile
show_heston_vol_smiled
Implied Black Volatility Smile
Option Price Sensitivity to Heston Parameters
Black volatility sensitivity to Heston Parameters
Simulating Heston, one path
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Simulated Barrier Option Prices
Implied Black Volatility Smile
Heston_smile - 1
Recalc
Recalc
Implied Black Volatility Smile
50.00 %
40.00 %
30.00 %
20.00 %
40.00
60.00
80.00
100.00
120.00
140.00
Function
View
Time
Description
Calibra...
Heston...
Heston...
Option ...
Black v...
Simulat...
Simulat...
Simulat...
Simulat...

v0	0.2
theta	0.3
rho	0
kappa	0.5
sigma	2
maturity	1
forward_value	100

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”


Heston_model_demo_workspace - Quantlab
File
Edit
View
Insert
Graph
Tools
Window
Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Heston Implied Volatility Smile
Option Price Sensitivity to Heston Parameters
Black volatility sensitivity to Heston Parameters
Simulating Heston, one path
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Simulated Barrier Option Prices
Heston Paramters vs Option Price
Common parameters
Recalc
Recalc
Heston Parameter vs Option Price
20
10
0
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
Function
View
Time
Description
Calibra...
Heston...
Heston...
Option ...
Black v...
Simulat...
Simulat...
Simulat...
Simulat...

v0	0.2
theta	0.3
rho	0
kappa	1
sigma	2
Forward Value	100
Strike	90
Time to Maturity	1
Discount Factor	1
Is Call	✓
Parameter on X-axis	sigma
Number of grid intervals on X-axis	100
X-axis min value	
X-axis max value	

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.

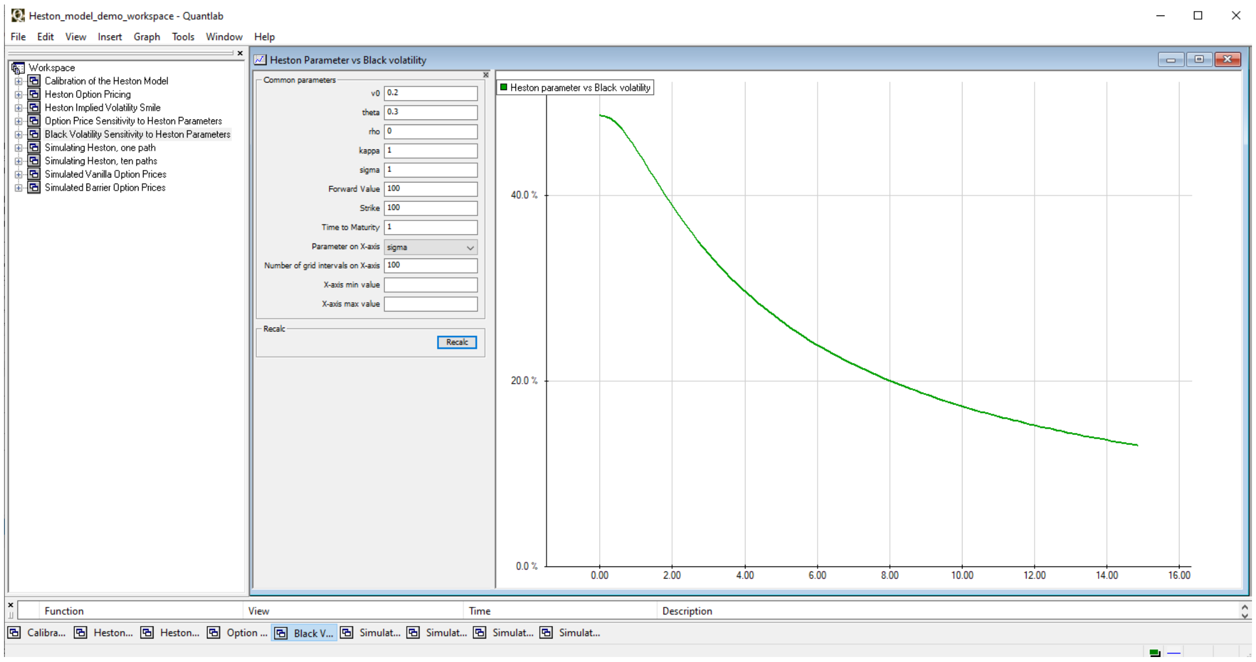
The graph shows the option price as a function of a chosen input parameter. All input paramters will be fixed, as given by the user, except one, which is the one we put on the x-axis.
So if we choose “**Parameter on X-axis**” as sigma. The graph shows how the option price depends on sigma (The value 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”


Heston_model_demo_workspace - Quantlab
File
Edit
View
Insert
Graph
Tools
Window
Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Heston Implied Volatility Smile
Option Price Sensitivity to Heston Parameters
Black Volatility Sensitivity to Heston Parameters
Simulating Heston, one path
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Simulated Barrier Option Prices
Heston Parameter vs Black volatility
Heston parameter vs Black volatility
40.0 %
20.0 %
0.0 %
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
Recalc
Function
View
Time
Description
Calibra...
Heston...
Heston...
Option ...
Black V...
Simulat...
Simulat...
Simulat...
Simulat...

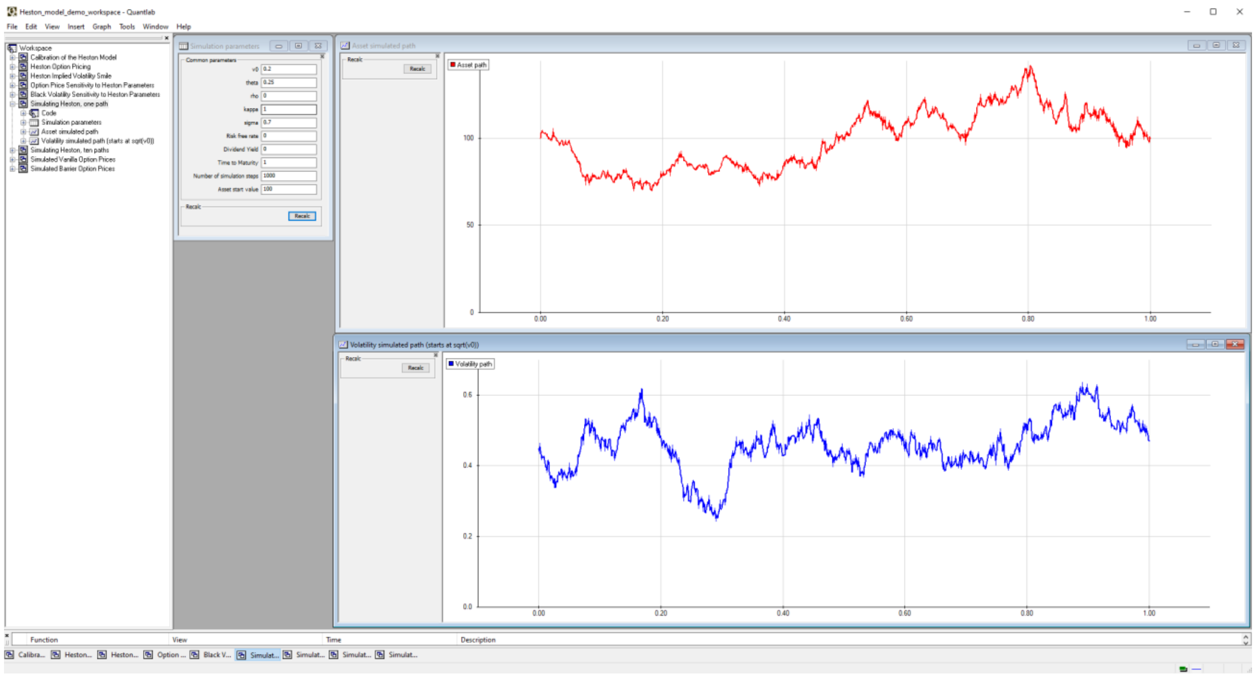
Common parameters

v0	0.2
theta	0.3
rho	0
kappa	1
sigma	1
Forward Value	100
Strike	100
Time to Maturity	1
Parameter on X-axis	sigma
Number of grid intervals on X-axis	100
X-axis min value	
X-axis max value	

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”


Heston_model_demo_workspace - Quantlab
File Edit View Insert Graph Tools Window Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Heston Implied Volatility Smile
Option Price Sensitivity to Heston Parameters
Black Volatility Sensitivity to Heston Parameters
Simulating Heston, one path
Code
Simulation parameters
Asset simulated path
Volatility simulated path (starts at sqrt(v0))
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Simulated Barrier Option Prices
Simulation parameters
Asset simulated path
Asset path
Volatility simulated path (starts at sqrt(v0))
Volatility path
Recalc
Function
View
Time
Description

Common parameters

v0	0.2
theta	0.25
rho	0
kappa	1
sigma	0.7
Risk free rate	0
Dividend Yield	0
Time to Maturity	1
Number of simulation steps	1000
Asset start value	100

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 v0=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, inital_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.

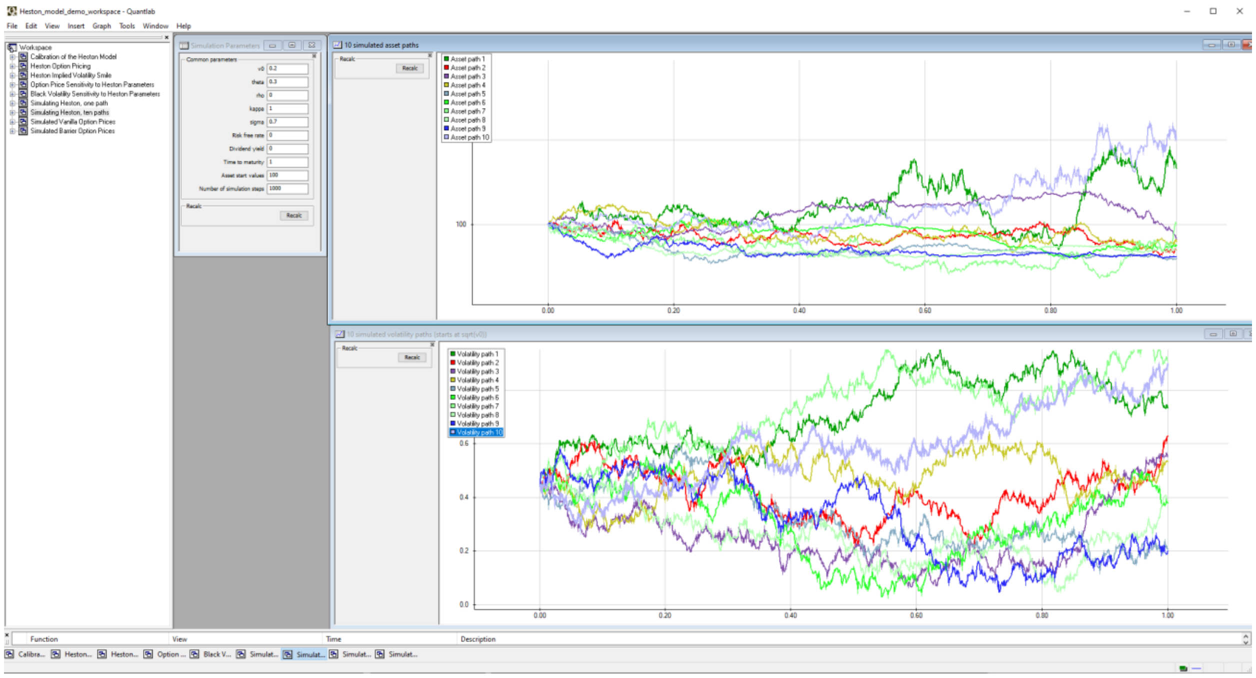
volatility_values() Returns the simulated volatilites (square root of the variances).

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”



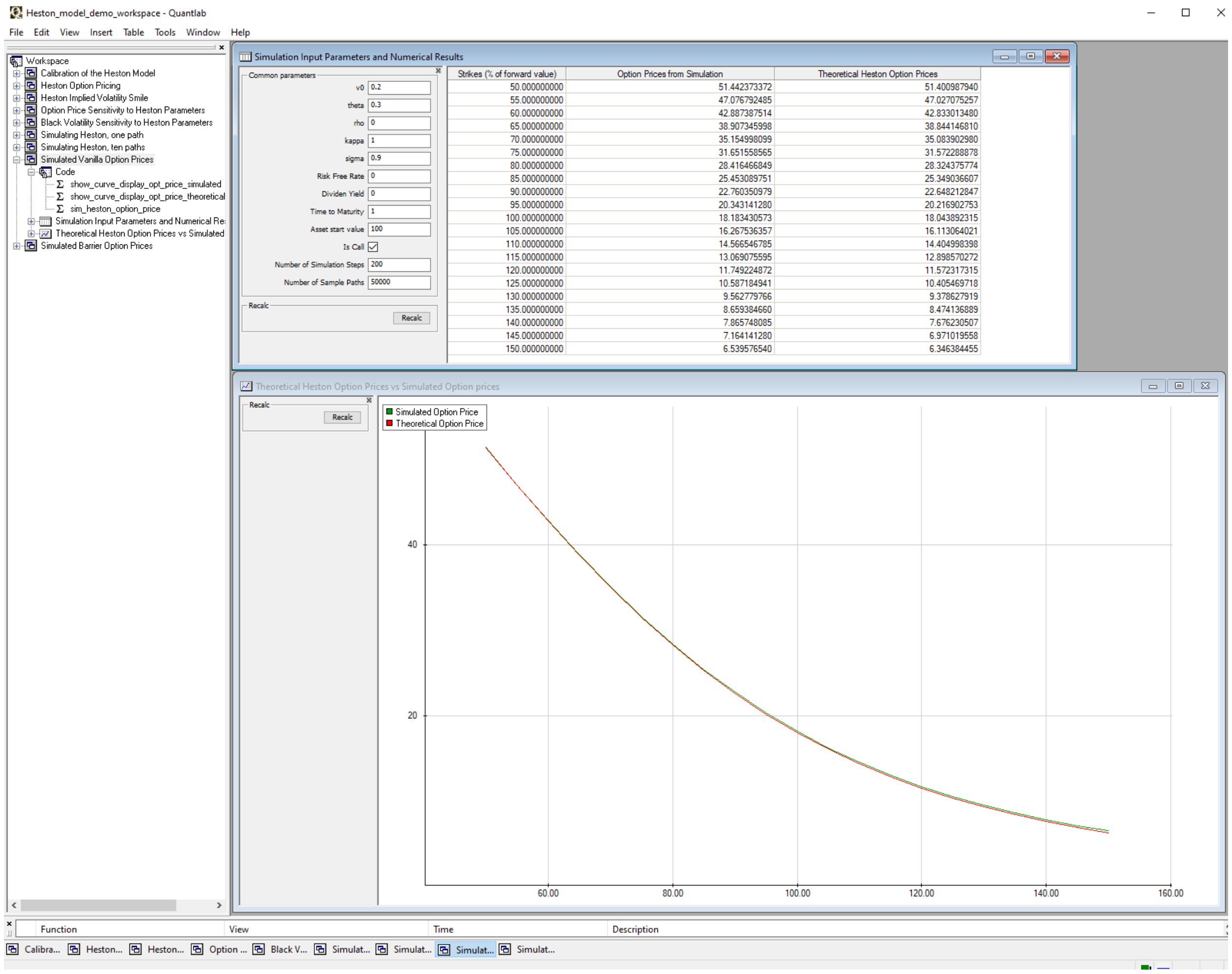
Common parameters

v0	0.2
theta	0.3
rho	0
kappa	1
sigma	0.7
Risk free rate	0
Dividend yield	0
Time to maturity	1
Asset start values	100
Number of simulation steps	1000

Recalc

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


Heston_model_demo_workspace - Quantlab
File Edit View Insert Table Tools Window Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Heston Implied Volatility Smile
Option Price Sensitivity to Heston Parameters
Black Volatility Sensitivity to Heston Parameters
Simulating Heston, one path
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Code
show_curve_display_opt_price_simulated
show_curve_display_opt_price_theoretical
sim_heston_option_price
Simulation Input Parameters and Numerical Re
Theoretical Heston Option Prices vs Simulated
Simulated Barrier Option Prices
Theoretical Heston Option Prices vs Simulated Option prices
Recalc
Simulated Option Price
Theoretical Option Price
40
20
60.00
80.00
100.00
120.00
140.00
160.00
Function
View
Time
Description

Simulation Input Parameters and Numerical Results

Common parameters

v0	0.2
theta	0.3
rho	0
kappa	1
sigma	0.9
Risk Free Rate	0
Dividen Yield	0
Time to Maturity	1
Asset start value	100
Is Call	✓
Number of Simulation Steps	200
Number of Sample Paths	50000

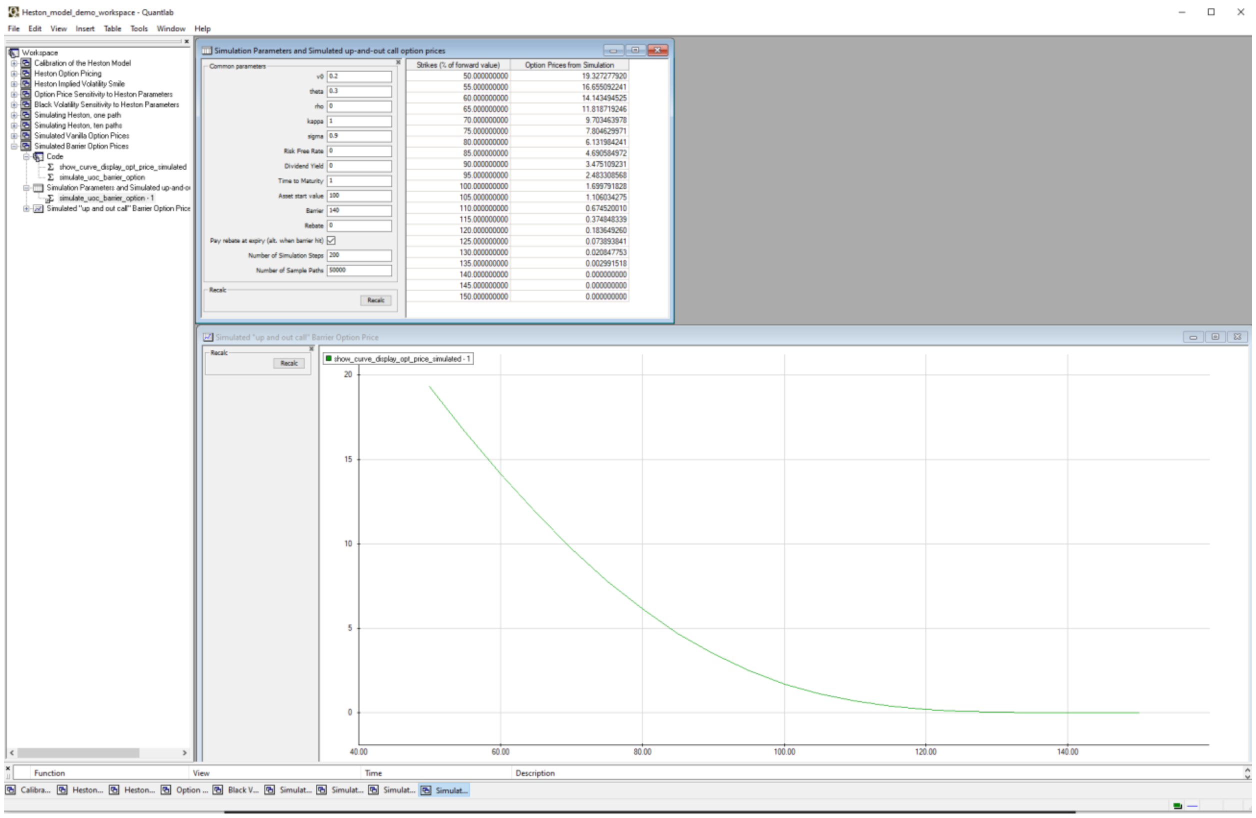
Strikes (% of forward value)	Option Prices from Simulation	Theoretical Heston Option Prices
50.000000000	51.442373372	51.400987940
55.000000000	47.076792485	47.027075257
60.000000000	42.887387514	42.833013480
65.000000000	38.907345998	38.844146810
70.000000000	35.154998099	35.083902980
75.000000000	31.651558565	31.572288878
80.000000000	28.416466849	28.324375774
85.000000000	25.453089751	25.349036607
90.000000000	22.760350979	22.648212847
95.000000000	20.343141280	20.216902753
100.000000000	18.183430573	18.043892315
105.000000000	16.267536357	16.113064021
110.000000000	14.566546785	14.404998398
115.000000000	13.069075595	12.898570272
120.000000000	11.749224872	11.572317315
125.000000000	10.587184941	10.405469718
130.000000000	9.562779766	9.378627919
135.000000000	8.659384660	8.474136889
140.000000000	7.865748085	7.676230507
145.000000000	7.164141280	6.971019558
150.000000000	6.539576540	6.346384455

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”


Heston_model_demo_workspace - Quantlab
File Edit View Insert Table Tools Window Help
Workspace
Calibration of the Heston Model
Heston Option Pricing
Heston Implied Volatility Smile
Option Price Sensitivity to Heston Parameters
Black Volatility Sensitivity to Heston Parameters
Simulating Heston, one path
Simulating Heston, ten paths
Simulated Vanilla Option Prices
Simulated Barrier Option Prices
Code
show_curve_display_opt_price_simulated
simulate_uoc_barrier_option
Simulation Parameters and Simulated up-and-o
simulate_uoc_barrier_option - 1
Simulated "up and out call" Barrier Option Price
Simulation Parameters and Simulated up-and-out call option prices
Recalc
Simulated "up and out call" Barrier Option Price
show_curve_display_opt_price_simulated - 1
Function
View
Time
Description

Common parameters

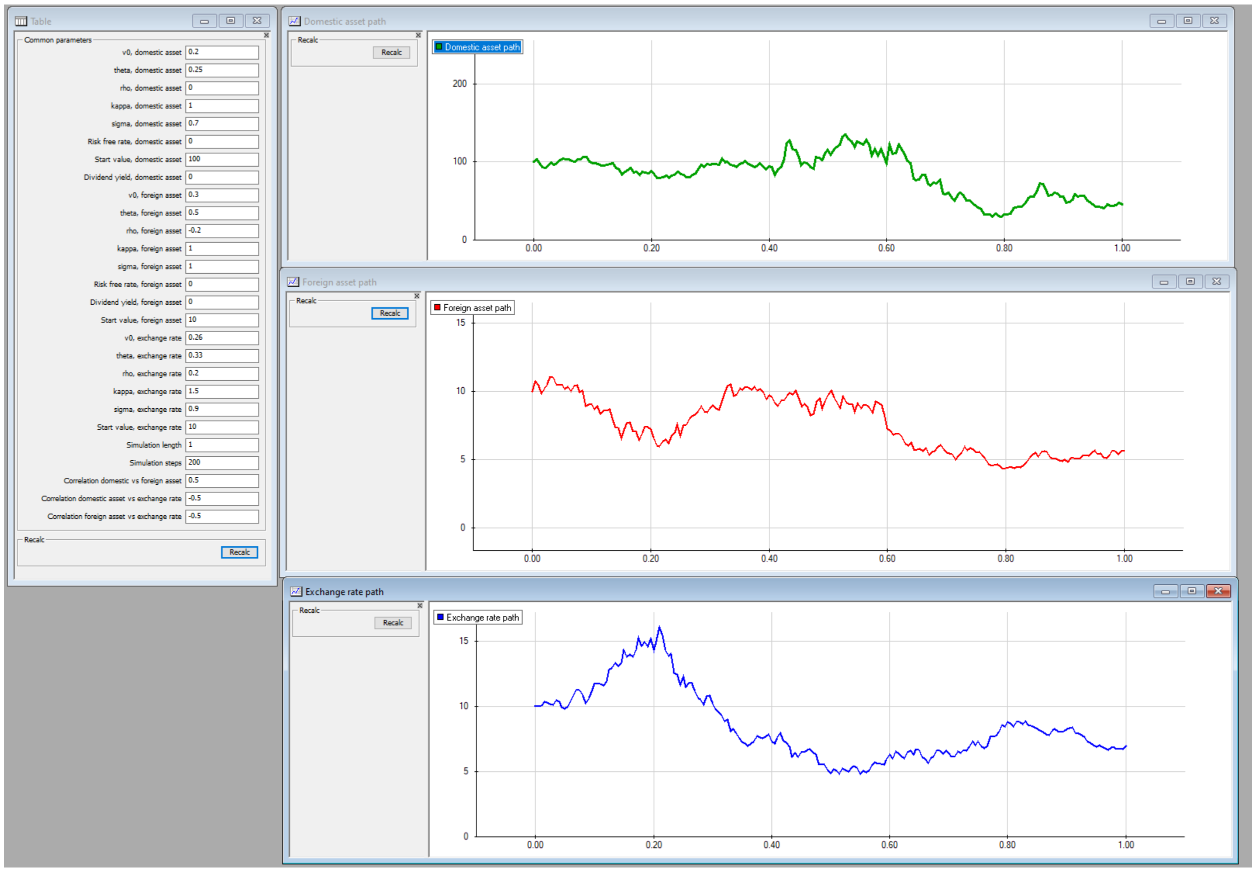
v0	0.2
theta	0.3
rho	0
kappa	1
sigma	0.9
Risk Free Rate	0
Dividend Yield	0
Time to Maturity	1
Asset start value	100
Barrier	140
Rebate	0
Pay rebate at expiry (alt. when barrier hit)	✓
Number of Simulation Steps	200
Number of Sample Paths	50000

Strikes (% of forward value)	Option Prices from Simulation
50.000000000	19.327277920
55.000000000	16.655092241
60.000000000	14.143494525
65.000000000	11.818719246
70.000000000	9.703463978
75.000000000	7.804629971
80.000000000	6.131984241
85.000000000	4.690584972
90.000000000	3.475109231
95.000000000	2.483308568
100.000000000	1.699791828
105.000000000	1.106034275
110.000000000	0.674520010
115.000000000	0.374848339
120.000000000	0.183649260
125.000000000	0.073893841
130.000000000	0.020847753
135.000000000	0.002991518
140.000000000	0.000000000
145.000000000	0.000000000
150.000000000	0.000000000

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 be 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.

Workspace Tab “Simulating Multi-Asset Heston”



Common parameters

v0, domestic asset	0.2
theta, domestic asset	0.25
rho, domestic asset	0
kappa, domestic asset	1
sigma, domestic asset	0.7
Risk free rate, domestic asset	0
Start value, domestic asset	100
Dividend yield, domestic asset	0
v0, foreign asset	0.3
theta, foreign asset	0.5
rho, foreign asset	-0.2
kappa, foreign asset	1
sigma, foreign asset	1
Risk free rate, foreign asset	0
Dividend yield, foreign asset	0
Start value, foreign asset	10
v0, exchange rate	0.26
theta, exchange rate	0.33
rho, exchange rate	0.2
kappa, exchange rate	1.5
sigma, exchange rate	0.9
Start value, exchange rate	10
Simulation length	1
Simulation steps	200
Correlation domestic vs foreign asset	0.5
Correlation domestic asset vs exchange rate	-0.5
Correlation foreign asset vs exchange rate	-0.5

Recalc

Recalc

The point of this tab is to illustrate the Multi-Asset Heston model in a simple example.
We do a simulation of three factors. First we have an asset in the domestic base market, let us say the “SEK” market.
Then we add another “foreign” market, say the “USD” market. When doing so we simulate the exchange rate for USD/SEK in the Heston model.
Then we add an asset on foreign market and simulate this together with the rest.
And finally we also can add correlations between the simulated assets and exchange rate.
The input is of the same type as for the single asset Heston, but repeated, except that we now can give the correlations between the factors too.
So let’s continue to describe the usage of the code behind this.

The multi-asset Heston simulation class.

The Multi-asset Heston simulation class has a constructor that looks like this

```
multi_asset_heston_simulation(integer n_paths, string domestic_currency, disc_func domestic_yield_curve);
```

Here we create the simulator object and also sets up the domestic market for the simulation.

n_paths is the number of sample paths to use in the simuation.

domestic_currency is the currency of the domestic market, like “SEK”. It is just a label and can be choosen freely, so it does not have to match any existing currency.

domestic_yield_curve is the risk free yield curve on the domestic market represented as a *disc_func* object.

Example:

To create a Multi-asset Heston simulator with “SEK” as domestic market and currency, using 1000 sample paths and using a risk free domestic yield curve with nodes at 1,2 and 3 year with yields 0.01 (1%), 0.011 (1.1%) and 0.006 (0.6%), we would write

```
integer n_paths = 1000;

string domestic_currency = "SEK";

vector(number) maturities      = [1,2,3];
vector(number) domestic_yields = [0.01, 0.011, 0.006];

interpolator ip = ip_linear(0,0); /
disc_func    domestic_yield_curve = disc_func_interp(maturities, exp(- domestic_yields .* maturities), ip);

multi_asset_heston_simulation sim = multi_asset_heston_simulation(n_paths, domestic_currency, domestic_yield_curve);
```

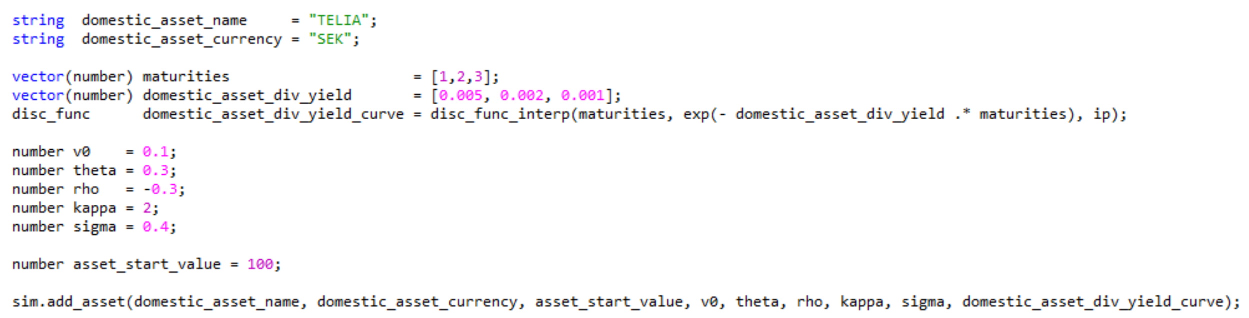
The interpolator object tells us that we should use linear interpolation and constant extrapolation (derivatives 0) before the first point and after the last point.

Let us now add an asset to the given domestic market. To do this we use the member function with signature

```
void add_asset(string    asset_name,
               string    asset_currency,
               number    initial_value,
               number    heston_initial_variance,
               number    heston_long_term_variance,
               number    heston_correlation,
               number    heston_mean_reversion,
               number    heston_vol_vol,
               disc_func option(nullable) dividend_yield = null<disc_func>);
```

Example:

We call the asset “TELIA” and its currency should be “SEK”. We choose/construct a dividend yield curve the same way we construct a yield curve. This is not mandatory, though.
We need to provide the assets starting value, which we choose to be 100 SEK and also the five Heston model parameters, which we choose a bit arbitrary in this example.
To do all this, we write:



```
string  domestic_asset_name     = "TELIA";
string  domestic_asset_currency = "SEK";

vector(number) maturities                   = [1,2,3];
vector(number) domestic_asset_div_yield       = [0.005, 0.002, 0.001];
disc_func      domestic_asset_div_yield_curve = disc_func_interp(maturities, exp(- domestic_asset_div_yield .* maturities), ip);

number v0    = 0.1;
number theta = 0.3;
number rho   = -0.3;
number kappa = 2;
number sigma = 0.4;

number asset_start_value = 100;

sim.add_asset(domestic_asset_name, domestic_asset_currency, asset_start_value, v0, theta, rho, kappa, sigma, domestic_asset_div_yield_curve);
```

So far so good. We can, so far, only add assets to the domestic market, since we have not introduced any foreign markets yet.
Let us now assume we want to add the “USD” market/currency. This can be done in several ways. We have three different ways of adding a foreign market:

1. We can choose to simulate the exchange rate to USD in SEK using a Heston simulation. Then we need to provide the Heston parameters.

2. We can also choose not to simulate the exchange rate, since we might not have any use for it. Then it is assumed that the exchange rate is modelled using a standard Black-Scholes model.
Even if the process is not simulated, its volatility and correlations to the assets on the same foreign market are needed to calculate the so called quanto-adjustments for the drift terms of the foreign assets.

3. Alternative 3. Is a lazy version of alternative 2, but here we don’t even require the volatility of the exchange rate. This is then assumed to be zero, which will make the quanto adjustments zero as well. Since the quanto-adjustments typically are small compared to the other terms, ignoring them will not have a huge effect on the simulation.

The three member functions to use in all those cases have signatures

```
void add_foreign_market(string    foreign_currency,
                        disc_func foreign_yield_curve);

void add_foreign_market(string    foreign_currency,
                        disc_func foreign_yield_curve,
                        number    fx_black_vol);

void add_foreign_market(string    foreign_currency,
                        disc_func foreign_yield_curve,
                        number    fx_initial_value,
                        number    fx_heston_initial_variance,
                        number    fx_heston_long_term_variance,
                        number    fx_heston_correlation,
                        number    fx_heston_mean_reversion,
                        number    fx_heston_vol_vol);
```

The input is pretty straightforward. We need a label/currency for the market, a given risk free yield curve for the market. If we choose to simulate the exchange rate, we need an initial value and the Heston parameters.
So adding a “USD” foreign market with the “USD” exchange rate simulated using Heston (using some arbitrary data) we would write

```
number v0    = 0.1;
number theta = 0.3;
number rho   = -0.3;
number kappa = 2;
number sigma = 0.4;

number fx_start_value = 10;

string         foreign_currency    = "USD";
vector(number) maturities          = [1,2,3];
vector(number) foreign_yields      = [0.01, 0.015, 0.02];
disc_func      foreign_yield_curve = disc_func_interp(maturities, exp(- foreign_yields .* maturities), ip);

sim.add_foreign_market(foreign_currency, foreign_yield_curve, fx_start_value, v0, theta, rho, kappa, sigma);
```

Let us finally assume we want to add an asset to the foreign “USD” market. Note that the market must have been added before the asset can be added. This is done using the same function to add assets to the domestic market.
Example:

```
number v0    = 0.1;
number theta = 0.3;
number rho   = -0.3;
number kappa = 2;
number sigma = 0.4;

number foreign_asset_start_value = 10;

string         foreign_asset_name           = "TESLA";
string         foreign_currency             = "USD";
vector(number) maturities                    = [1,2,3];
vector(number) foreign_asset_div_yield       = [0.006, 0.003, 0.001];
disc_func      foreign_asset_div_yield_curve = disc_func_interp(maturities, exp(- foreign_asset_div_yield .* maturities), ip);

sim.add_asset(foreign_asset_name, foreign_currency, foreign_asset_start_value, v0, theta, rho, kappa, sigma, foreign_asset_div_yield_curve);
```

So we now have three simulated factors in our simulator: The domestic asset “TELIA”, the foreign asset “TESLA” and the exchange rate “USD” (in SEK).
We can choose to add correlations (for the log returns) between any of those. This can be done using the member function with signature

```
void add_correlation(string asset_or_fx_name_1, string asset_or_fx_name_2, number correlation);
```

Example:

```
sim.add_correlation("TELIA", "TESLA", 0.5);
sim.add_correlation("TELIA", "USD", -0.4);
sim.add_correlation("TESLA", "USD", -0.3);
```

When this is done the simulator is ready to start simulating (unless we want to add more factors).

This we do using the simulate member function with signature

```
void simulate(number  simulation_length,
              integer n_simulation_steps,
              rng     my_rng,
              logical use_martingale_correction = true);
```

We need to give the length of the simulation (in years), the number of time steps to use, a random number generator and lastly an optional technical parameter which tells the simulator to do a so call martingale correction or not (this is a method to remove model bias due to too large time steps).

When we want to retrieve values from the simulator, we can use the getter functions

```
vector(number) asset_values(string name);
vector(number) log_asset_values(string name);

vector(number) fx_values(string name);
vector(number) log_fx_values(string name);

vector(number) volatility_values(string asset_or_fx_name);
vector(number) variance_values(string asset_or_fx_name);
```

We have separated getters for the simulated exchange rates and the assets for clarity.