Algorithmica
Research AB

The Heston Stochastic Local Volatility (SLV) Model

Demo workspace walkthrough

Last update: 2021-02-12

Table of Contents

The basic Heston Stochastic Volatility Model

This section is a short repetition of the basic Heston Stochastic volatility model.
More details can be found in the document “Heston Stochastic Volatility Model” describing that model and the demo workspace for that 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 Local Volatility Model

The local volatility model is a generalization of the standard Black-Scholes model with constant volatility σ:

$$\frac{dS(t)}{S(t)} = \mu(t)dt + \sigma \, dW(t),$$

where the constant volatility is replaced be a so called *local volatility* function, $\sigma_{LV}(t, S)$, depending on both time and the current level of the asset:

$$\frac{dS(t)}{S(t)} = \mu(t)dt + \sigma_{LV}(t,S)\, dW(t)$$

This model has the huge advantage of replicating market prices exactly (if the local volatility function has been calibrated correctly).
However, despite this, this model should not be used in practice for pricing something more exotic path dependent options. The reason for this is that the stochastic behavior of this model is not realistic. The main issue is that the local volatility function is calibrated at market data at time zero and remains fixed no matter how the process develops.
So in general the basic Heston model is a much better choice for pricing of exotic options, even though the fit to market vanilla prices is much worse than the local volatility model.

I would be nice if we could somehow combine the good aspects of both the Local volatility model and the Heston model. This is the purpose of the so called Heston Stochastic local Volatility (SLV) model we will describe in the next section.

The Heston Stochastic Local Volatility (SLV) Model

The Heston SLV model is given by the same two processes as the standard Heston model, but where we have an extra volatility factor $\sigma(t, S)$ for the asset term:

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

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

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).

So the volatility term for the asset is a deterministic local volatility type function $\sigma(t, S)$ multiplied by the stochastic volatility factor $\sqrt{V(t)}$.

The we need to be able to compute the function $\sigma(t, S)$. We will not go into details in the rather involved derivations of this, but one can show that we need to have:

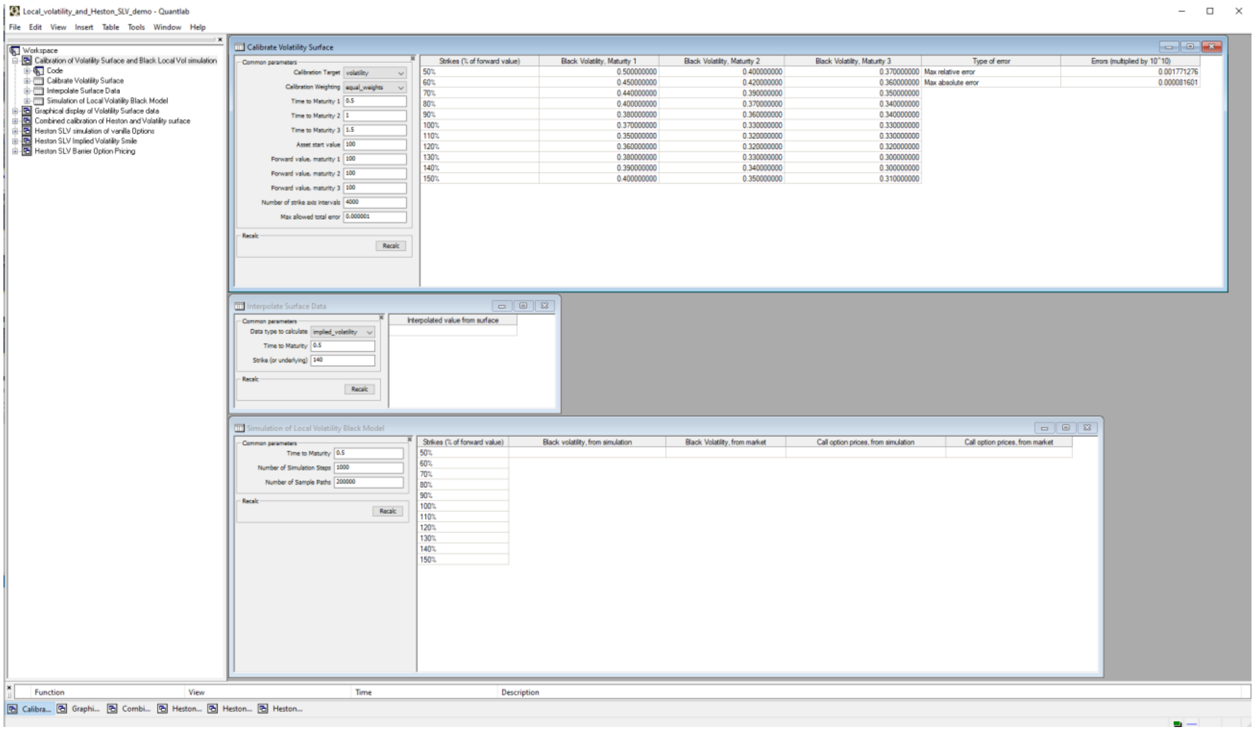
$$\sigma^2(t,K) = \frac{\sigma_{LV}^2(t,K)}{E[V(t)|S(t)=K]}\,.$$

The nominator is the local volatility function squared. The denominator is an expected value of the variance given the value of the underlying. This is not possible to calculate in any explicit analytic form.
In our implementation we use a method of calculating the empirical expected value by sorting the range of S(t) in a number of “bins” where the value of S(t) is approximately constant in each bin.
Then we interpolate the midpoints of those bins either using linear interpolation or splines. That function is then out approximation for the expected value above.
Then we of course have the problem of choosing an appropriate number of bins. The number of sample paths should be much larger than the number of bins so that each bin contains a decent number of samples.
Empirically around 80 bins seems like a good choice if the number of paths is like 100K.

Workspace tab “Calibration of Volatility Surface and Black Local Vol simulation”

This is a screenshot of the contents of the first tab in the Workspace. The purpose of this tab is to demonstrate the calibration of the volatility surface (which later will be used to get the local volatility) using some simplified input. This tab also contains a table where we can simulate the Local volatility model using the local volatility obtained from the calibrated volatility surface.

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


Local_volatility_and_Heston_SLV_demo - Quantlab
File Edit View Insert Table Tools Window Help
Workspace
Calibration of Volatility Surface and Black Local Vol simulation
Code
Calibrate Volatility Surface
Interpolate Surface Data
Simulation of Local Volatility Black Model
Graphical display of Volatility Surface data
Combined calibration of Heston and Volatility surface
Heston SLV simulation of vanilla Options
Heston SLV Implied Volatility Smile
Heston SLV Barrier Option Pricing
Function
View
Time
Description
Calibra...
Graphi...
Combi...
Heston...
Heston...
Heston...

Calibrate Volatility Surface

Common parameters

Calibration Target: volatility
Calibration Weighting: equal_weights
Time to Maturity 1: 0.5
Time to Maturity 2: 1
Time to Maturity 3: 1.5
Asset start value: 100
Forward value, maturity 1: 100
Forward value, maturity 2: 100
Forward value, maturity 3: 100
Number of strike axis intervals: 4000
Max allowed total error: 0.000001

Recalc

Strikes (% of forward value)	Black Volatility, Maturity 1	Black Volatility, Maturity 2	Black Volatility, Maturity 3	Type of error	Errors (multiplied by 10^10)
50%	0.500000000	0.400000000	0.370000000	Max relative error	0.001771276
60%	0.450000000	0.420000000	0.360000000	Max absolute error	0.000081601
70%	0.440000000	0.390000000	0.350000000		
80%	0.400000000	0.370000000	0.340000000		
90%	0.380000000	0.360000000	0.340000000		
100%	0.370000000	0.330000000	0.330000000		
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	0.300000000		
150%	0.400000000	0.350000000	0.310000000		

Interpolate Surface Data

Common parameters

Data type to calculate: implied_volatility
Time to Maturity: 0.5
Strike (or underlying): 140

Recalc

Interpolated value from surface

Simulation of Local Volatility Black Model

Common parameters

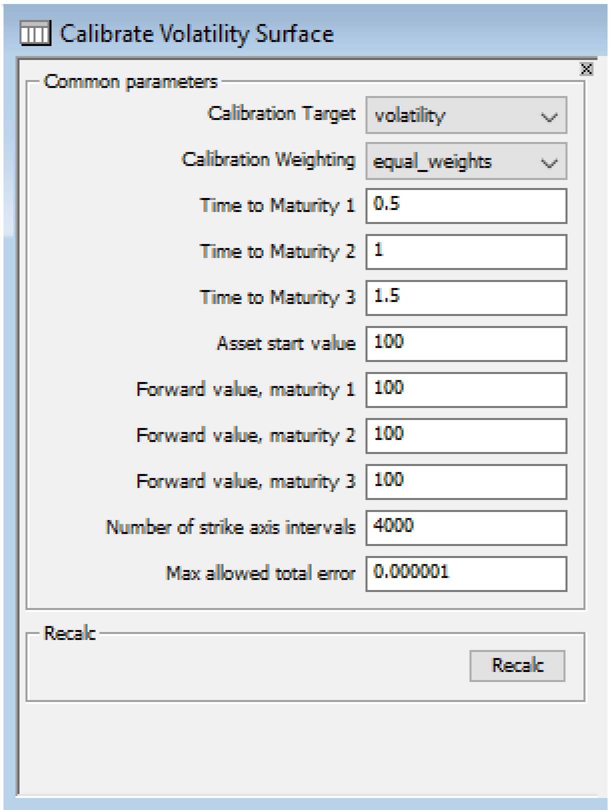
Time to Maturity: 0.5
Number of Simulation Steps: 1000
Number of Sample Paths: 200000

Recalc

Strikes (% of forward value)	Black volatility, from simulation	Black Volatility, from market	Call option prices, from simulation	Call option prices, from market
50%				
60%				
70%				
80%				
90%				
100%				
110%				
120%				
130%				
140%				
150%				

The errors (times 10^10) of the volatility surface calibration 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 input parameters for the volatility calibration expected from the user(except the Black volatilities) are on the left side of the upper table:


Calibrate Volatility Surface

Common parameters

Calibration Target: volatility

Calibration Weighting: equal_weights

Time to Maturity 1: 0.5

Time to Maturity 2: 1

Time to Maturity 3: 1.5

Asset start value: 100

Forward value, maturity 1: 100

Forward value, maturity 2: 100

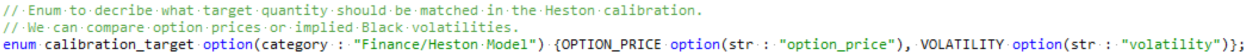
Forward value, maturity 3: 100

Number of strike axis intervals: 4000

Max allowed total error: 0.000001

Recalc

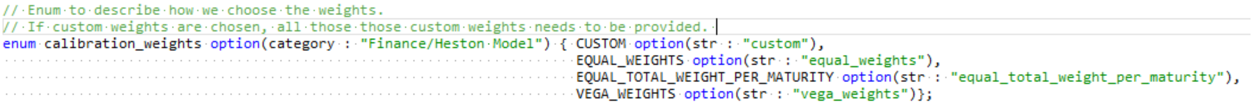
Calibration Target :
This specifies an enum:


```
// 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.

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.

Asset start value, Forward value 1, Forward value 2, Forward value 3 :

Here the forward values for the corresponding maturities are specified (and the asset start value at time zero).

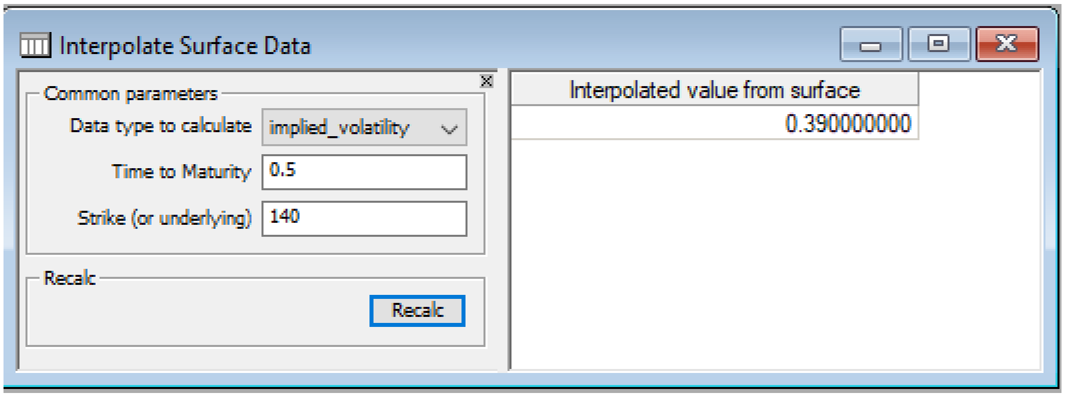
Number of strike axis intervals

We calibrate the volatility surface using a finite difference grid in the time and strike direction. This is how many grid intervals we use in the spatial strike direction. Do not use to few. 3000 or 4000 seems to be a nice choice.

Max allowed total error

This concerns the least squares error norm for the given maturity. The norm is the square root of the squared differences between model and market values. If the tolerance here is breached, an error will be thrown.

Then we have the table



Data type to calculate: implied_volatility

Time to Maturity: 0.5

Strike (or underlying): 140

Interpolated value from surface
0.390000000

This table can only be used after the volatility surface has been successfully calibrated using the previously mentioned table. The purpose of this is to inspect the values we get from the calibrated volatility surface.

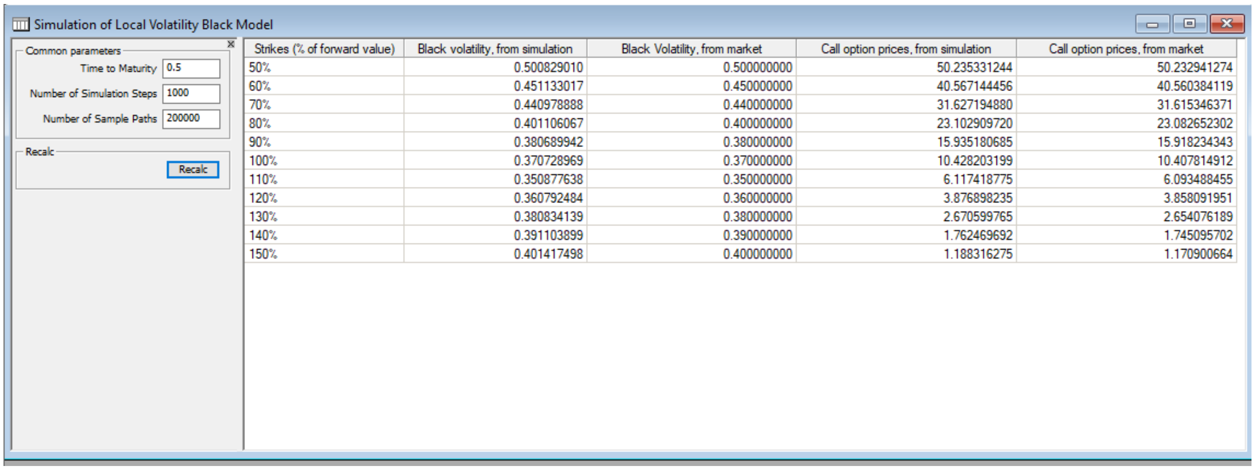
This is simple enough: In the first drop down menu, choose what kind of data we want from the surface. Choose between call option values, put option values, implied Black volatility or local volatility.

The time to maturity and strikes is precisely that, choose the maturity and strike to inspect.

Of special interest is of course the maturities and strikes we used to calibrate the surface. We should get back our used market values to high precision.

The local volatility surface we get is very irregular and has not been smoothed in any way. Typically there will be discontinuities when we pass the market maturities. This is not as bad as it might seem. The purpose of the local volatilities is to “add up” together in the correct way to match the implied volatilities.

Then we have the Local vol model simulation table:



Time to Maturity: 0.5

Number of Simulation Steps: 1000

Number of Sample Paths: 200000

Strikes (% of forward value)	Black volatility, from simulation	Black Volatility, from market	Call option prices, from simulation	Call option prices, from market
50%	0.500829010	0.500000000	50.235331244	50.232941274
60%	0.451133017	0.450000000	40.567144456	40.560384119
70%	0.440978888	0.440000000	31.627194880	31.615346371
80%	0.401106067	0.400000000	23.102909720	23.082652302
90%	0.380689942	0.380000000	15.935180685	15.918234343
100%	0.370728969	0.370000000	10.428203199	10.407814912
110%	0.350877638	0.350000000	6.117418775	6.093488455
120%	0.360792484	0.360000000	3.876898235	3.858091951
130%	0.380834139	0.380000000	2.670599765	2.654076189
140%	0.391103899	0.390000000	1.762469692	1.745095702
150%	0.401417498	0.400000000	1.188316275	1.170900664

Like the previous table, this can only be used after a volatility surface has been calibrated using the first table. The input is rather self explanatory: We give the length of the simulation as the time to maturity. We choose the number of simulation steps and the number of sample paths.

Monte Carlo simulations converge poorly so one has to use many paths to converge. Too few steps and

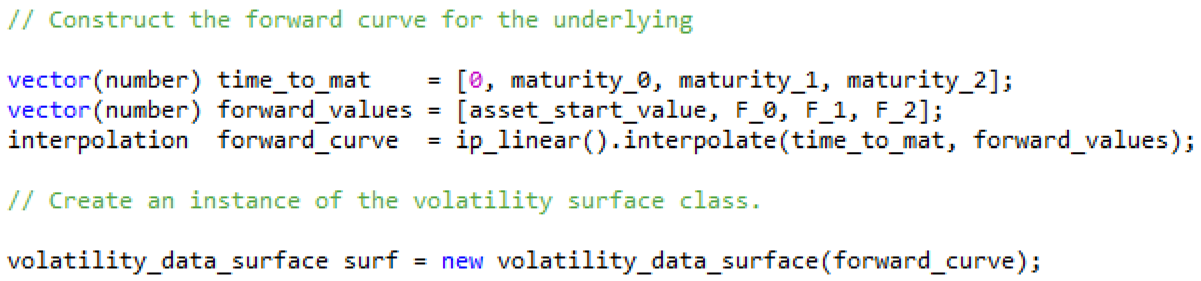
too few sample paths gives us two different types of errors. Too few steps gives a discretization error resulting in bias that can’t be overcome no matter how many sample paths we use.
Too few sample path causes problem with too few samples to convergence in the Central limit theorem that is the basis of Monte Carlo.
So one should balance the two parameters in a reasonable way. Folk lore says that the number of steps should be proportional to the square root of the number of sample paths for a good balance.

The output is just the market and simulated Black vol and option prices compared.

The Volatility surface data Class

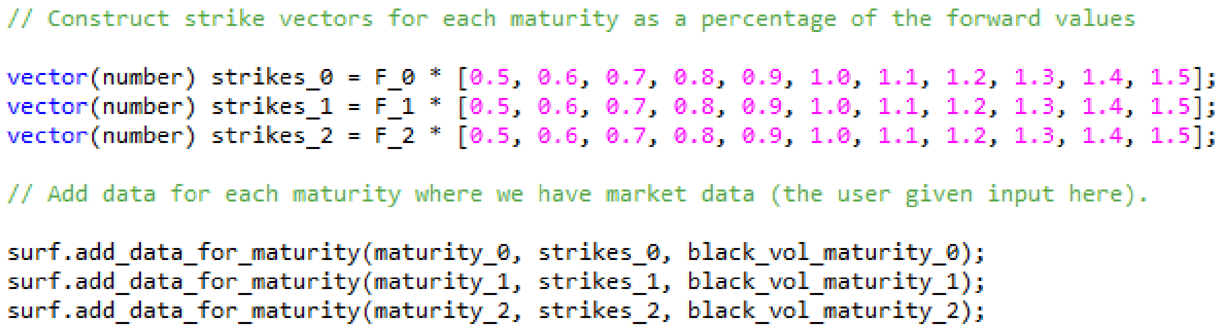
Create an instance of the volatility data surface class. To do this we need a forward curve describing the forward values of the asset for each maturity, including the start value of the asset.

Here maturity_0, maturity_1, and maturity_2 are the time to maturities given by the user and F_0, F_1 and F_2 the corresponding forward values. We create an interpolation object which will use linear interpolation (Alternatively there is a constructor taking one vector of maturities and one vector of forward values. Linear interpolation will then be used internally in the class).



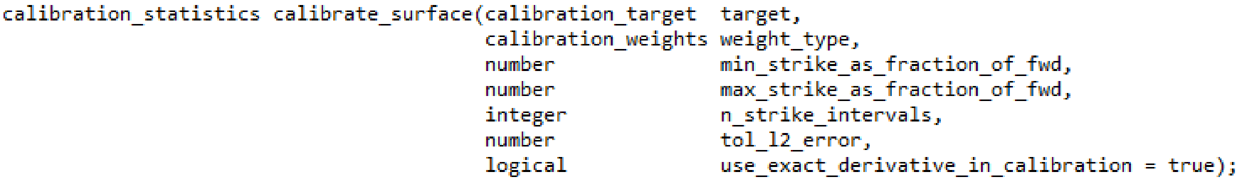
```
// Construct the forward curve for the underlying

vector(number) time_to_mat    = [0, maturity_0, maturity_1, maturity_2];
vector(number) forward_values = [asset_start_value, F_0, F_1, F_2];
interpolation  forward_curve  = ip_linear().interpolate(time_to_mat, forward_values);

// Create an instance of the volatility surface class.

volatility_data_surface surf = new volatility_data_surface(forward_curve);
```

We then construct three strike vectors, one for each maturity, based on percentages of the forward values. Then we add the strikes and black maturities for each maturity to the class:



```
// Construct strike vectors for each maturity as a percentage of the forward values

vector(number) strikes_0 = F_0 * [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5];
vector(number) strikes_1 = F_1 * [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5];
vector(number) strikes_2 = F_2 * [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5];

// Add data for each maturity where we have market data (the user given input here).

surf.add_data_for_maturity(maturity_0, strikes_0, black_vol_maturity_0);
surf.add_data_for_maturity(maturity_1, strikes_1, black_vol_maturity_1);
surf.add_data_for_maturity(maturity_2, strikes_2, black_vol_maturity_2);
```

Then we call the calibration function that will we used to calibrate the volatility surface. This function has signature:



```
calibration_statistics calibrate_surface(calibration_target  target,
                                         calibration_weights weight_type,
                                         number              min_strike_as_fraction_of_fwd,
                                         number              max_strike_as_fraction_of_fwd,
                                         integer             n_strike_intervals,
                                         number              tol_l2_error,
                                         logical             use_exact_derivative_in_calibration = true);
```

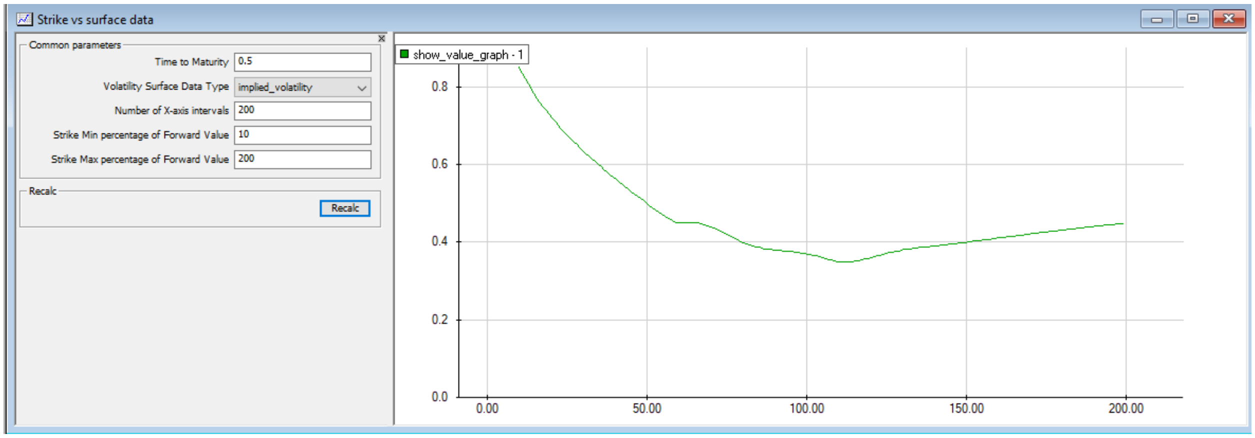
The first two argument was already described above. The arguments **min_strike_as_fraction_of_fwd** and **max_strike_as_fraction_of_fwd** tells the calibration the outer limits of the strike grid expressed as a fraction of the forward value. The forward value used by the grid internally is always 1, which then will be rescaled in a clever way. So if for example min_strike_as_fraction_of_fwd = 0.5, the minimum value in the internal grid will be 0.5*1 = 0.5.
Empirical studies shows that it is a very bad idea to choose lower and upper limits too tight to the market data. The technical reason for this is that the finite difference grid has boundary conditions saying that the second derivative of the call price there should be zero. This condition becomes a bit incompatible with the rest of the grid depending on the market data.
Therefore the **min_strike_as_fraction_of_fwd** should be close to 0, say 0.01 and **max_strike_as_fraction_of_fwd** should be large, like say 5.

The **tol_l2_error** is the maximum error norm allowed, mentioned earlier.

The last variable is a technical one deciding if we should use exact or numerical derivatives in the calibration. Both choices should give very similar results.

The function returns a kind of statistics object that can be ignored, but contains some good debug data for the calibration.

Workspace Tab “Graphical display of Volatility Surface data”



Strike vs surface data

Common parameters

Time to Maturity: 0.5

Volatility Surface Data Type: implied_volatility

Number of X-axis intervals: 200

Strike Min percentage of Forward Value: 10

Strike Max percentage of Forward Value: 200

Recalc

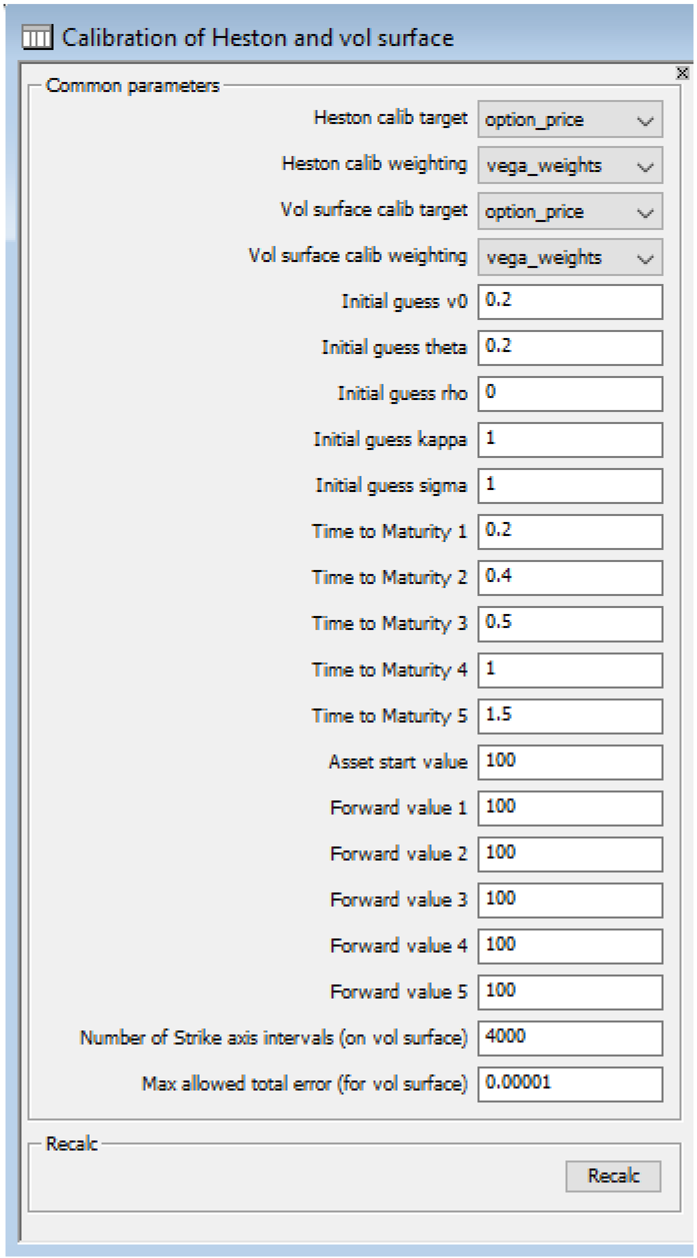
This tab is used to graphically display the volatility surface data. We choose the time to maturity to inspect, what kind of data to inspect (call option prices, put option prices, implied volatility or local volatility), the number of intervals to use on the x-axis, the min and max value on the x-axis expressed as a percentage of the forward value (so 20 represents 20% of the forward value)

Workspace Tab “Combined calibration of Heston and Volatility surface”

The purpose of this tab is to calibrate both the Heston model and the volatility surface data class based on the data given here. NOTE: The results of this tab and table are used in the tabs below. Running this table is a requirement for the tabs below.

The entries in the upper part of the table, are modifiable, representing market black volatilities for different strikes and maturities. In the right columns we get the calibrated Heston parameters and the numerical errors for the volatility surface calibration.

The input to the left should be familiar by now:



Calibration of Heston and vol surface

Common parameters

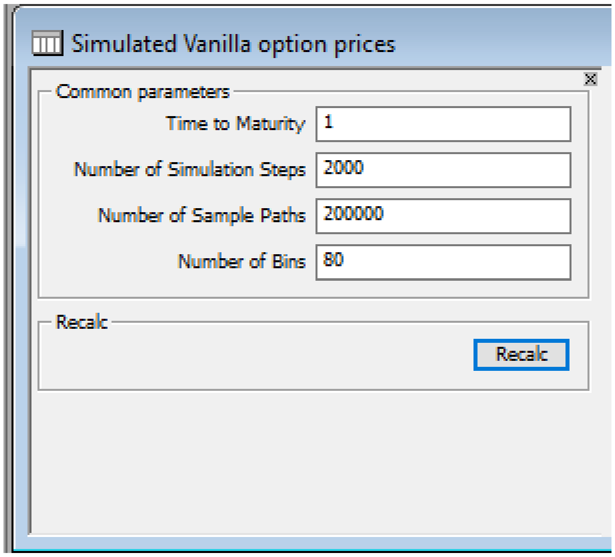
Heston calib target	option_price
Heston calib weighting	vega_weights
Vol surface calib target	option_price
Vol surface calib weighting	vega_weights
Initial guess v0	0.2
Initial guess theta	0.2
Initial guess rho	0
Initial guess kappa	1
Initial guess sigma	1
Time to Maturity 1	0.2
Time to Maturity 2	0.4
Time to Maturity 3	0.5
Time to Maturity 4	1
Time to Maturity 5	1.5
Asset start value	100
Forward value 1	100
Forward value 2	100
Forward value 3	100
Forward value 4	100
Forward value 5	100
Number of Strike axis intervals (on vol surface)	4000
Max allowed total error (for vol surface)	0.00001

We choose the calibration target and weighting for the Heston and the volatility surface calibration, the initial guesses for the parameters in the Heston model, market maturities, market forward values (including the asset start value), the number of grid intervals to use on the strike axis for the volatility surface, and the maximal error norm allowed on for the vol calibration.

We use five different maturities to get slightly more realistic input data than the three maturities we used before.

Workspace Tab “Heston SLV simulation of vanilla Options”

This is the main tab for simulation of the Heston Stochastic Local Volatility (SLV) model. This tab requires that a Heston model and a volatility surface has been calibrated in the previous tab.
We don’t need much input data here since most of the input are retrieved from the calibrated Heston model and the calibrated volatility surface :



Common parameters

Time to Maturity: 1

Number of Simulation Steps: 2000

Number of Sample Paths: 200000

Number of Bins: 80

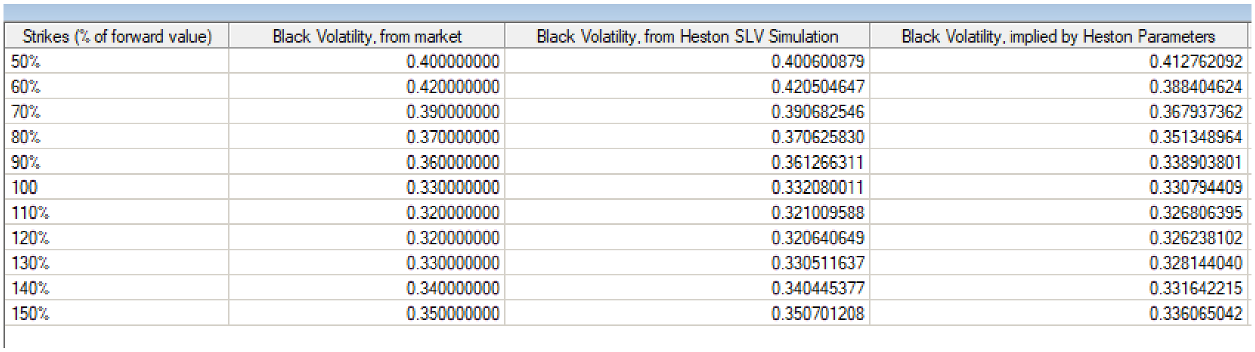
We need to provide a time to maturity telling the simulate how far to simulate (in years).

We need to provide the simulator with the number of simulation steps and the number of sample paths. We also need a more technical parameter for the specific Heston SLV model we use, the number of bins used when computing the empirical expected value mentioned earlier. 80 seems to be a reasonable choice here.

We then produce a number of columns to see how well the simulation performs, both compared to ordinary Heston model and compared to market data.

1. We have the Black volatility from market used as input. The closer we are to this, the better
2. Then we have Black volatility implied theoretically by the Heston model parameters. This is the benchmark for a pure Heston simulation. A perfect simulation would get a perfect match to these values.
3. Then we have the Black volatilities for the Heston SLV simulation. Those should, in general, be much closer to the market volatilities compared to those theoretical pure Heston volatilities mentioned in 2, indicating that the Heston SLV in general is superior to the standard Heston.

This is an example on what we get using the input above: (running it again will give different output due to different seed)



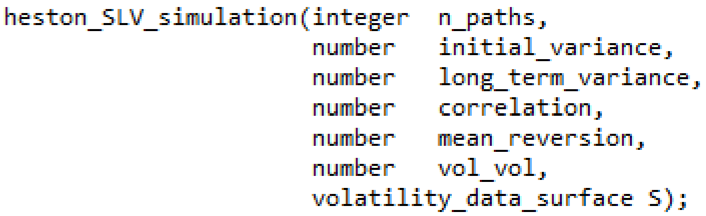
Strikes (% of forward value)	Black Volatility, from market	Black Volatility, from Heston SLV Simulation	Black Volatility, implied by Heston Parameters
50%	0.400000000	0.400600879	0.412762092
60%	0.420000000	0.420504647	0.388404624
70%	0.390000000	0.390682546	0.367937362
80%	0.370000000	0.370625830	0.351348964
90%	0.360000000	0.361266311	0.338903801
100	0.330000000	0.332080011	0.330794409
110%	0.320000000	0.321009588	0.326806395
120%	0.320000000	0.320640649	0.326238102
130%	0.330000000	0.330511637	0.328144040
140%	0.340000000	0.340445377	0.331642215
150%	0.350000000	0.350701208	0.336065042

Then we have the corresponding three columns for the option prices.

Then we have a graph below the table displaying the option price graphs for the three cases mentioned (market vs pure Heston theoretical prices, vs Simulated Heston SLV prices).

The Heston SLV simulation class

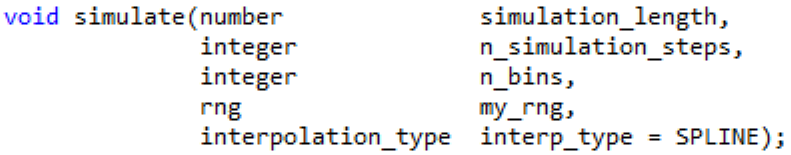
The constructor for the Heston SLV class looks like this



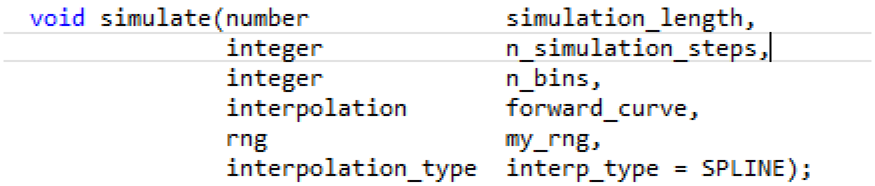
```
heston_SLV_simulation(integer  n_paths,
                      number   initial_variance,
                      number   long_term_variance,
                      number   correlation,
                      number   mean_reversion,
                      number   vol_vol,
                      volatility_data_surface S);
```

The first argument is the number of sample paths. The following five arguments are the standard Heston parameters, v0, theta, rho, kappa and sigma (just renamed a bit differently here).
The last argument should be an instance of the calibrated volatility surface class. Note that a forward curve for the underlying is given implicitly by that surface class.

To simulate we call one of the simulation member functions:



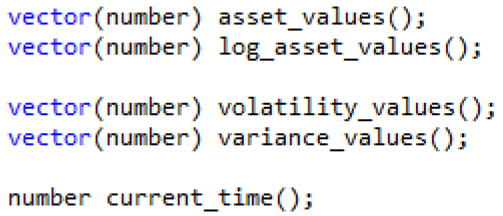
```
void simulate(number              simulation_length,
              integer             n_simulation_steps,
              integer             n_bins,
              rng                 my_rng,
              interpolation_type  interp_type = SPLINE);
```



```
void simulate(number              simulation_length,
              integer             n_simulation_steps,
              integer             n_bins,
              interpolation       forward_curve,
              rng                 my_rng,
              interpolation_type  interp_type = SPLINE);
```

The difference in those two lies in that one of them use the forward curve given by the volatility surface while the other one uses an external user given curve.
Otherwise most input arguments are pretty straightforward: First we have the simulation length in years. Then we have the number of simulation steps in the given period. Then we have the number of

bins used to calculate the internal expectation mentioned earlier. Then the optional forward curve, followed by the random number generator and the choice of interpolation techniques when interpolating the bins mentioned.
The class also have a few getters:



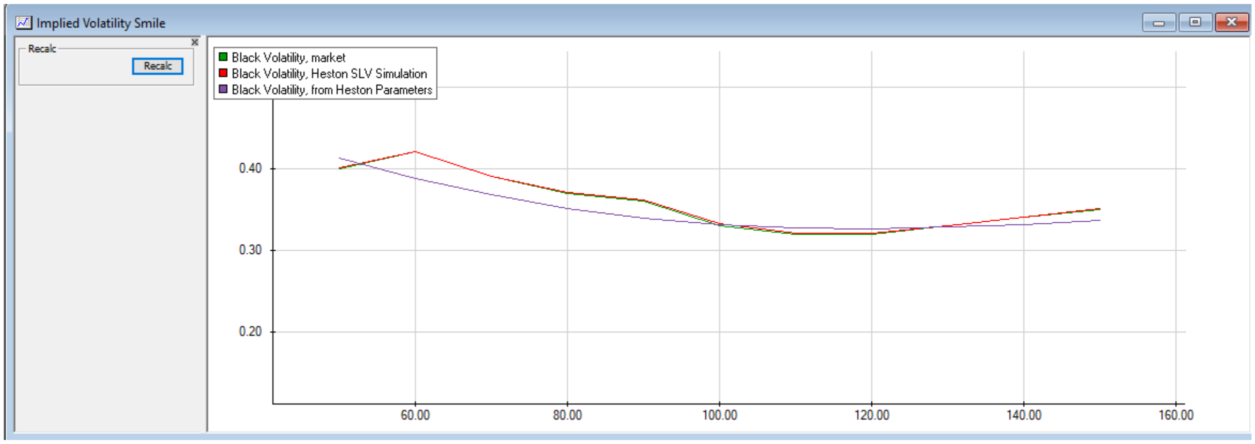
```
vector(number) asset_values();
vector(number) log_asset_values();

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

number current_time();
```

Gives the simulated asset values, the log asset values, the volatility values (square root of variance values) and variance values. And lastly a function giving how far in time the simulation has simulated at the moment (Not any real clock time, but as a measurement of the time axis in the simulation).

Workspace Tab “Heston SLV Implied Volatility Smile”

This is a tab that requires the Heston Simulation table to be run before the graph can be shown.

It shows the implied volatility smile for the Heston SLV simulation and compare it to the market smile and the pure Heston theoretically implied smile. Typically the market and Heston SLV smile are very close and the pure Heston smile is a bit off and more like a smooth average.
This picture is typical:

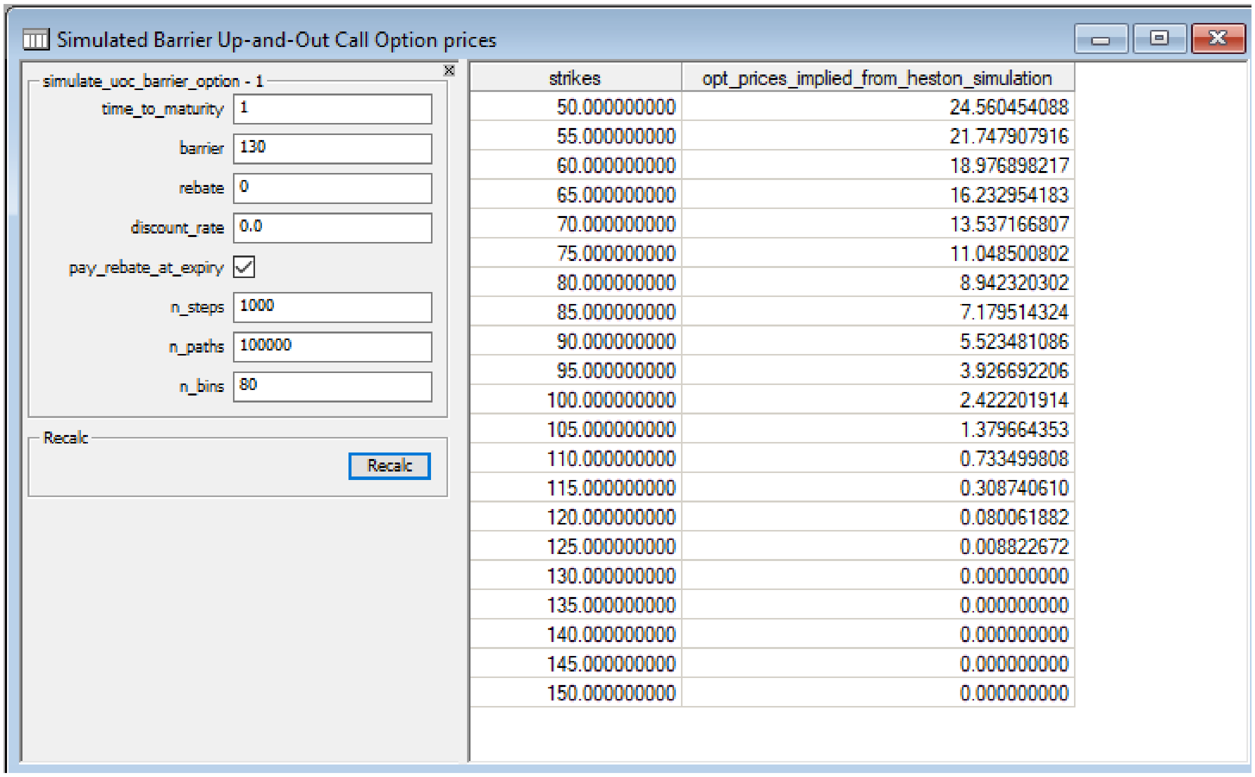

Implied Volatility Smile
Recalc
Recalc
🟩 Black Volatility, market
🟥 Black Volatility, Heston SLV Simulation
🟪 Black Volatility, from Heston Parameters
0.40
0.30
0.20
60.00
80.00
100.00
120.00
140.00
160.00

Workspace tab “Heston SLV Barrier Option pricing”

The point of this tab is to show how the Heston SLV 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.

We get a result table showing the option prices for different maturities:



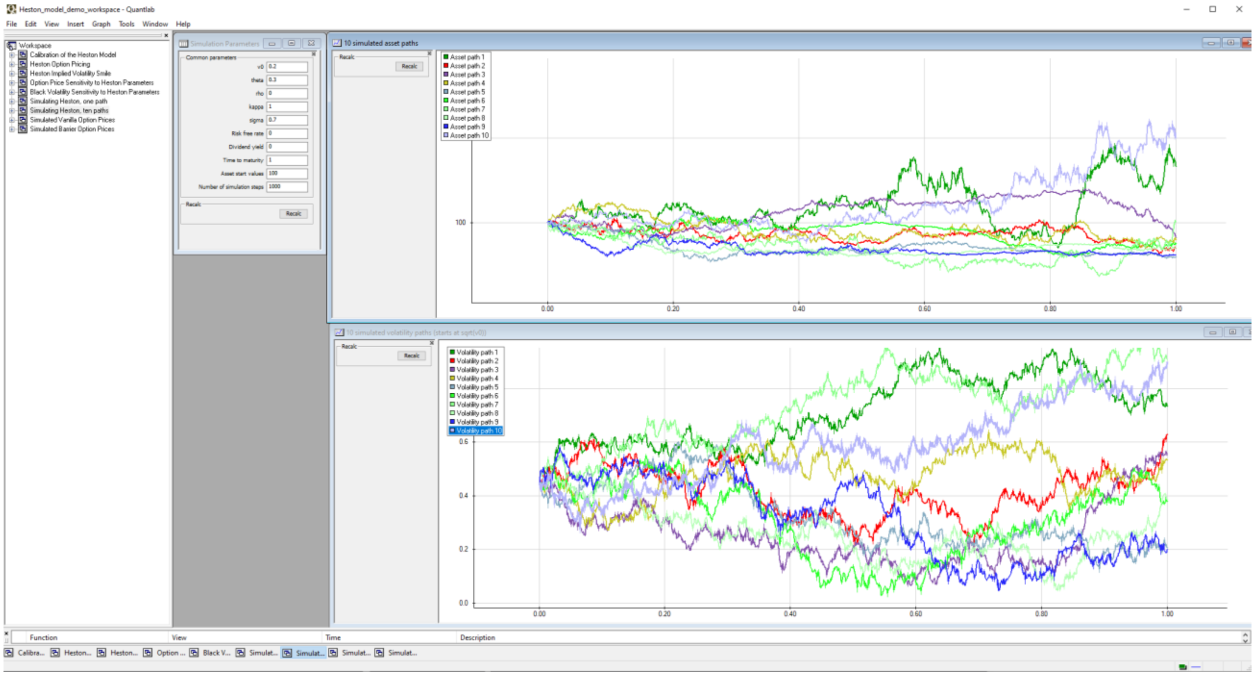
Simulated Barrier Up-and-Out Call Option prices

simulate_uoc_barrier_option - 1

time_to_maturity	1
barrier	130
rebate	0
discount_rate	0.0
pay_rebate_at_expiry	☑
n_steps	1000
n_paths	100000
n_bins	80

strikes	opt_prices_implied_from_heston_simulation
50.000000000	24.560454088
55.000000000	21.747907916
60.000000000	18.976898217
65.000000000	16.232954183
70.000000000	13.537166807
75.000000000	11.048500802
80.000000000	8.942320302
85.000000000	7.179514324
90.000000000	5.523481086
95.000000000	3.926692206
100.000000000	2.422201914
105.000000000	1.379664353
110.000000000	0.733499808
115.000000000	0.308740610
120.000000000	0.080061882
125.000000000	0.008822672
130.000000000	0.000000000
135.000000000	0.000000000
140.000000000	0.000000000
145.000000000	0.000000000
150.000000000	0.000000000

Workspace Tab “Simulating Heston, ten paths”



Heston_model_demo_workspace - Quantlab

Simulation Parameters

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

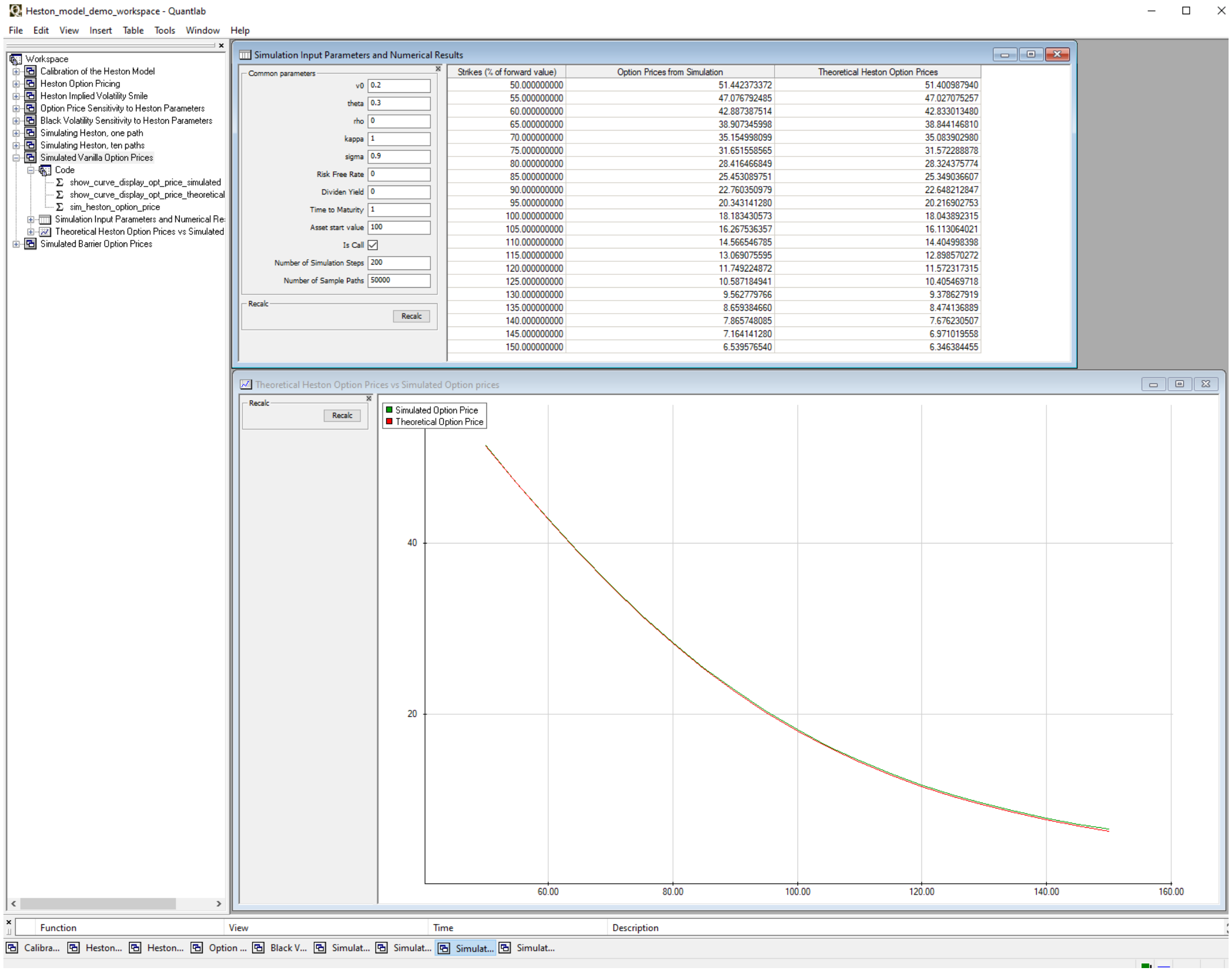
10 simulated asset paths

10 simulated volatility paths (starts at sqrt(v0))

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
Simulation Input Parameters and Numerical Results
Common parameters
Recalc
Recalc
Theoretical Heston Option Prices vs Simulated Option prices
Recalc
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
Calibra...
Heston...
Heston...
Option ...
Black V...
Simulat...
Simulat...
Simulat...
Simulat...

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

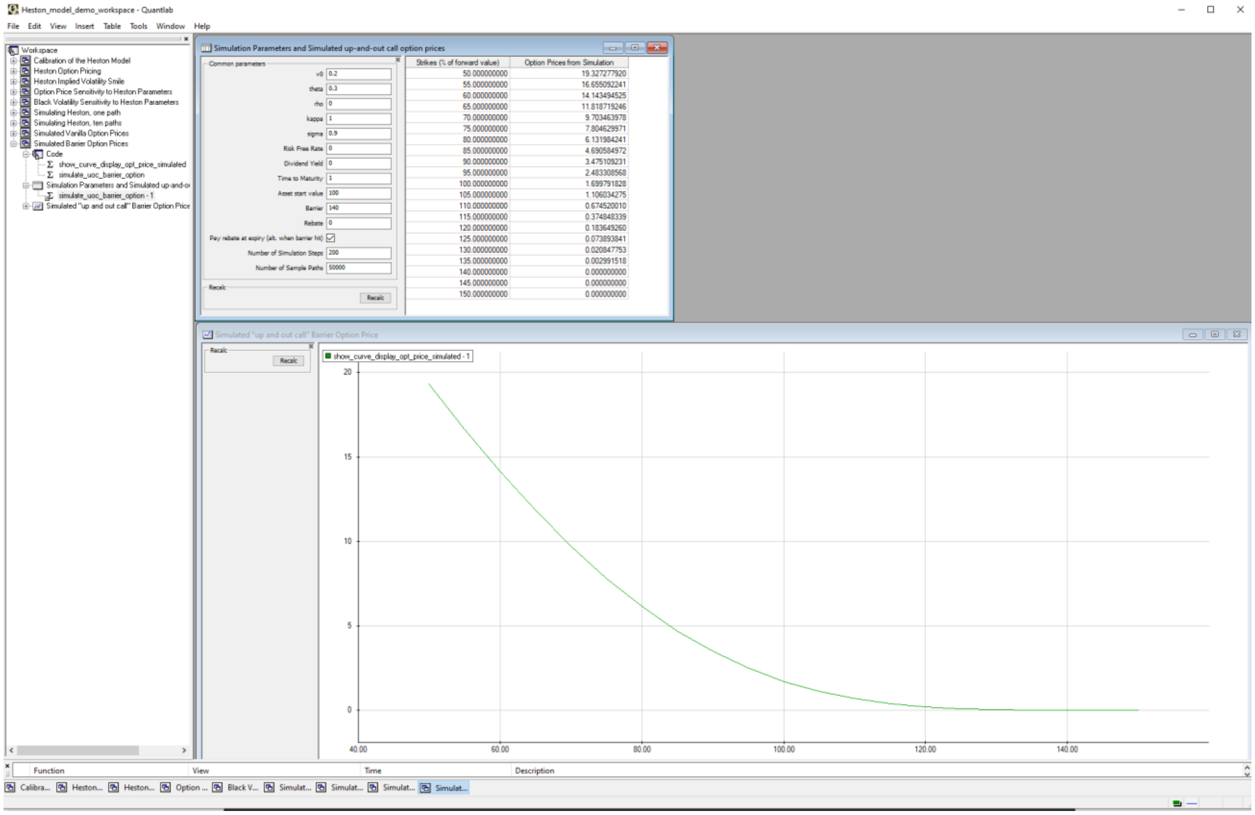
Strikes (% of forward value)	Option Prices from Simulation	Theoretical Heston Option Prices
50.000000000	51.442373372	51.400987940
55.000000000	47.076792485	47.027075257
60.000000000	42.887387514	42.833013480
65.000000000	38.907345998	38.844146810
70.000000000	35.154998099	35.083902980
75.000000000	31.651558565	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
Recalc
Simulated "up and out call" Barrier Option Price
Recalc
Recalc
show_curve_display_opt_price_simulated - 1
20
15
10
5
0
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...

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 substeps 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.