Computer practical*

Svetlana Borovkova
FEWEB / VU

Artem Tsvetkov
FEWEB / VU

September 2, 2016

1 Matlab refreshment

Please refresh your Matalb knowledge before the practice. We will move quickly through this in the class. Please look through the accompanying m-file. There is more on Matlab programming in it.
HELP is very useful in Matlab. Access it through Search Documentation in the upper right corner and learn how to use it.

1.1 Comments

Matlab ignores everything on the line after the percentage symbol %. We will use it for commenting the code.

1.2 Run the code

You can run Matlab code in different ways.

- Press F5, if want to run the code from the beginning till the end,
- Or in Editor tab click Run.
- To run a section from double comment, %%, to the next double comment, click anywhere inside and press Ctrl-Enter,
- Or click Run Section in the Editor menu.

1.3 Vectors, matrices, and operations with them

Matlab is a natural tool to operate on vectors and matrices. Vectors are actually treated as one-dimensional matrices in Matlab. It is possible to use arrays of three and more dimensions, but we do not need them.
You can define an arbitrary row vector by typing

*Course on Stochastic Processes for Finances 2016

```
x = [1, 3, 4]
```

Replace commas by semicolon and you get a column vector

```
y = [1; 3; 4]
```

You can also do it by transposing vector x

```
y1 = x'
```

Regularly spaced row vector t with values between $[a, b]$ can be created with predetermined steps dt as

```
t = (a:dt:b)
```

or with predetermined number of steps N

```
t = linspace(a,b,N)
```

Note that the operation ended with semicolon ';' is not printed in Command Window, but those ended with comma or without punctuation will be printed. This is useful to see your results.
Element-wise multiplication between two matrices of the *same* size is done with operator ".∗".

```
C = A.*B
```

Matrix multiplication of two matrices A, $m \times n$, and B, $n \times k$, is simply

```
C = A*B
```

and results in matrix C, $m \times k$.

Exercise 1 (Cholesky decomposition)**.** To practice a bit with matrices let us look at Cholesky decomposition, which will be used later to correlate random variables. Given a symmetric positive definite matrix C, Cholesky decomposition factorizes it into a product of upper triangular matrix U and its transpose lower triangular matrix U^T

$$C = U^T U \tag{1}$$

Define correlation matrix C

$$C = \begin{bmatrix} 1 & 0.4 & 0.3 \\ 0.4 & 1 & 0.5 \\ 0.3 & 0.5 & 1 \end{bmatrix},$$

compute matrix U using Cholesky decomposition

```
U = chol(C);
```

and show that Eq. 1 is fulfilled.

1.4 Graphs

Matlab has very powerful graphical capabilities. We will use them to visualize the results. To plot vector x as a function of its index, for example, one can type

```
plot(x)
```

A plot of vector x as a function of another vector t can be done as

```
plot(t,x)
```

A plot of all rows in $m \times n$ matrix Y as a function of $1 \times n$ vector t is simply

```
plot(t,Y)
```

It is often convenient to plot different graphs in different windows. Put $figure(i)$ in front of the plot command to direct the plot in figure i

```
figure(1)
plot(t,x)
figure(2)
plot(t,Y)
```

The first plot will be window 1, while the second will be in window 2.
Add legend if there are more than one line on the plot.

```
plot(x,sin(x),x,cos(x))
legend('Sin','Cos')
```

In some exercises we need to plot multiple figures. It may be convenient to combine them in tabs of one figure. The following construction can be used. It is a bit cumbersome but let us to avoid lots of standalone figures. For example in Exercise 9, you can write

```
hfig9 = figure(9); %Declare figure 9 and get handler 'hfig9'
set(hfig9,'Name','Geometrical Brownian Motion'); %give name
set(hfig9,'WindowStyle','normal'); %make standalone window
htabgroup9 = uitabgroup(hfig9); %create a group of tabs

htab9_1 = uitab(htabgroup9, 'Title', 'Abs returns');
hax9_1 = axes('Parent', htab9_1);
plot(t,x) %plot x(t) in tab 'Abs returns'

htab9_2 = uitab(htabgroup9, 'Title', 'Log returns');
hax9_2 = axes('Parent', htab9_2);
plot(t,y) %plot y(x) in tab 'Log returns

% etc.
```

These are just suggestions. Use your favorite method to visualize your results.

2 Random number generation

2.1 Random numbers in one dimension

We will use normally distributed random numbers a lot. The function to generate a matrix $n \times m$ of independent random numbers is

```
Z = randn(n,m);
```

This function generate pseudo-random numbers of standard normal distribution, i.e. with mean zero and variance 1.

Exercise 2 (Random numbers)**.** Generate vector, say Z, of normal random numbers of length $N = 100,000$. Plot this series by using

```
figure(1)
plot(Z)
```

Remember declaration '$figure(1)$' above is not required, but is convenient in case you want to plot the next graph in your code in a different window. Put $figure(2)$ in front of the second figure and so on. You better plot every 100-th value to make the plot 'nicer'. You can do it by putting $Z(1:100:end)$, which means that only indexes from 1 till the last one with step 100 are used.
Compute mean, standard deviation, excess kurtosis, and skewness of the series, see that these are indeed standard normal parameters:

```
mZ = mean(Z)
sZ = std(Z)
ekZ = kurtosis(Z) - 3
skZ = skewness(Z)
```

Observe that $mean(Z)$ is close but not quite equal to zero due to the finite sample. The deviation will be even higher if we take a half of vector Z

```
ZH = Z(1:N/2)
mean(ZH)
```

Combine ZH with itself with negative sign and compute the mean.

```
ZA = [ZH;-ZH];
```

This so called 'antithetic sampling' is a legitimate technique to reduce variance in random number simulations with symmetric pdf. Why does this trick 'magically' make the mean exactly zero? Would this 'magic' work for any anti-symmetric component of the averaging function, like x^3 for example?
Plot a histogram of the empirical density function. Use the following code

```
figure(2)
[y,x] = hist(Z,100);
plot(x,y)
```

Function *hist* sorts elements in Z into 100 equally spaced bins. y is the number of counts in bins, while x are the center values in each bin. Function *plot* draws the graph of y as a function of x.
Modify your code to compare the obtained distribution with the standard normal distribution, computed as

```
yn = normpdf(x);
plot(x,y,x,yn)
```

Why do the plots not coincide? How do you need to rescale y to match normal pdf *yn*?
In practice you need to generate vectors with non-zero mean and non-unit variance. Use the very same vector Z, computed in the previous step, to obtain another vector $Z1$ with mean 3 and standard deviation 2. Compare to the corresponding normal pdf (do not forget to rescale y).

```
yn = normpdf(x,3,2);
plot(x,y,x,yn)
```

2.2 Random numbers in many dimensions

Usually, we need to generate random numbers for two or more correlated random variables. We use the fact that the sum of two *normal* random variables is again a normal random variable.
Suppose we have realizations of three *independent* random variables X, Y, and Z, all with zero mean. We can generate three correlated random variables, $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$, by making linear combinations of vectors X, Y, and Z. Or in matrix form, we multiply matrix $[XYZ]$, which consists of three columns X, Y, and Z,

$$[XYZ] = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}.$$

with some loading 3×3 matrix U:

$$\left[\tilde{X}\tilde{Y}\tilde{Z}\right] = [XYZ] \cdot U.$$

It turns out that in order to have a correlation matrix C between $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$, the corresponding matrix U is obtained by the Cholesky decomposition.
It can be shown analytically that the random variables produced in this way have indeed correlation matrix C. We will do it empirically.
For that you can use the standard formulas. Suppose that $X = \{x_1, x_2, ..., x_N\}$ and $Y = \{y_1, y_2, ..., y_N\}$ are series of observations for two normal random variables with mean zero. We can estimate the covariance and correlation between

them in the following way

$$\mathrm{cov}[X,Y] = \mathbb{E}[X \cdot Y] \approx \frac{1}{N-1}\sum_{i=1}^{N} x_i y_i$$

$$\rho_{XY} = \frac{\mathbb{E}[X \cdot Y]}{\sqrt{\mathbb{E}[X^2]\mathbb{E}[Y^2]}} \approx \frac{\sum_{i=1}^{N} x_i y_i}{\sqrt{\sum_{i=1}^{N} x_i^2 \cdot \sum_{i=1}^{N} y_i^2}}$$

or by using corresponding Matlab functions.

Exercise 3 (Independent normal random variables)**.** Use $randn(N, M)$ to generate a two column matrix Z with say $N = 10,000$. Check that the columns are linearly independent. For that, compute covariances and correlations either by

```
cov_Z = cov(Z(:,1),Z(:,2));
rho_Z = corr(Z);
```

or

```
cov_z = Z(:,1)'*Z(:,2)/(N-1);
rho_z = Z(:,1)'*Z(:,2)/sqrt((Z(:,1)'*Z(:,1)) * (Z(:,2)'*Z(:,2)));
```

Think what exactly is done in the second method and why it gives slightly different values than the first method?
Plot $Z(:,1)$ vs. $Z(:,2)$ and observe a round 'cloud' of points.

Exercise 4 (Correlated normal random variables)**.** Define in Matlab correlation matrix

$$C = \begin{bmatrix} 1 & -0.7 \\ -0.7 & 1 \end{bmatrix},$$

Compute matrix $Z1$ with two columns of correlated random variables using matrix Z and the Cholesky decomposition.

```
Z1 = Z * U;
```

Compute covariance and correlation matrices, check that the correlation matrix is C. Plot $Z(:,1)$ vs. $Z(:,2)$ and observe that the cloud becomes elongated. Feel free to play with positive and negative correlations.

Exercise 5 (If Cholesky fails)**.** Define correlation matrix

$$C = \begin{bmatrix} 1 & 0.9 & -0.9 \\ 0.9 & 1 & 0.95 \\ -0.9 & 0.95 & 1 \end{bmatrix},$$

try to apply the Cholesky decomposition. The algorithm to compute the decomposition is very robust. If it fails it means that the matrix is not positive definite. Think in terms of correlations why the decomposition fails? Are all correlation matrices possible?

3 Monte Carlo simulation of stochastic processes

3.1 Brownian motion generation

We start with generation of paths for Brownian motion $W(t)$. Remember that

$$
\begin{aligned}
&W(0) = 0 \\
&\mathbb{E}\left[W(t) - W(s)\right] = 0 \\
&\mathrm{var}\left[W(t) - W(s)\right] = t - s \\
&\mathbb{E}\left[\left(W(t) - W(s)\right)\left(W(v) - W(u)\right)\right] = 0, \forall s \leq t \leq u \leq v
\end{aligned}
$$

We simulate Brownian motion on the interval $[0, T]$. For that we introduce a time grid $\{t_0 = 0, t_1 = \Delta T, t_2 = 2\Delta T, \ldots, t_M = M\Delta T = T\}$. At any time t_i the value of the process for a single realization can be presented as

$$W(t_i) = \sum_{j=1}^{i} \left(W(t_j) - W(t_{j-1})\right) = \sum_{j=1}^{i} \Delta W(t_j)$$

Here $\Delta W(t_j)$ is a normal random variable

$$Z_j \sim \mathcal{N}(0, \sqrt{\Delta T})$$

with mean zero and standard deviation $\sqrt{\Delta T}$. Thus if $\{Z_j\}$ is a series of standard normal random numbers, we can generate a single realization for $W(t_i)$ as

$$W(t_i) = \sum_{j=1}^{i} \Delta W(t_j) = \sum_{j=1}^{i} Z_j \sqrt{\Delta T}.$$

Exercise 6 (Single path generation)**.** Generate and plot a single path of Brownian motion on $[0, 1]$ interval. Try it a few times to see how it behaves from a realization to realization.

Exercise 7 (Multiple paths generation)**.** Suppose we want to compute an expectation of some function of Brownian motion at time T. We need to generate multiple paths. For that we need to loop over time steps and over different paths. Or we can use the Matlab matrix functionality to simplify the code and to speed up the calculations. There are different ways but the most efficient and transparent way is to generate a vector of increments for N paths on every time step and loop over the time steps like shown below.

```
W = zeros(npath,nstep+1); % Initial initialization of
% the simulated variable (to zero in this case).
for i = 1:nstep
    Z = randn(npath,1);
    % Do what you need to do here
end
WT = W(:,end); % Values at time T
```

Plot the paths to visualize them for yourself. You can do it in one go. If *tgrid* is a $1 \times (nstep + 1)$ time grid vector, then plot

```
plot(tgrid, W)
```

Observe a square root divergence of the paths.
Generate $N = 10,000$ paths and show that $W(t = 1)$ has indeed zero mean, variance one, and zero excess kurtosis and skewness. Plot the empirical distribution of $W(T)$ at time T and compare it to the normal pdf.

Exercise 8 ($dW^2 = dt$ and $dW_1 dW_2 = 0$)**.** Show numerically that the following relations are satisfied

$$I_1(t) = \int_0^t dW^2 = t$$
$$I_2(t) = \int_0^t dW_1 dW_2 = 0$$

For this, approximate integrals I_1 and I_2 by sums, respectively,

$$S_1(t_i) = \sum_{j=1}^{i} \Delta W(t_j)^2$$
$$S_2(t_i) = \sum_{j=1}^{i} \Delta W_1(t_j) \Delta W_2(t_j)$$

Here $\Delta W_{1,2}(t_j)$ are independent normally distributed increments. Simulate 10 paths on the time interval [0,1] with different number of time steps N: $10, 100, \ldots, 10^5$. Plot the paths for each number of steps. See if the paths for S_i converge to $S_1(t_i) = t_i$ and $S_2(t_i) = 0$, respectively.

3.2 Itô lemma

GBM is described by the following stochastic differential equation (SDE):

$$dS(t) = rS(t)dt + \sigma S(t)dW_t$$
$$S(0) = S_0$$

or by the discrete-time approximation

$$S(t_{i+1}) = S(t_i) + rS(t_i)(t_{i+1} - t_i) + \sigma S(t_i)(W_{i+1} - W_i)$$

Using Itô lemma the equation can be written as

$$d\ln(S_t) = (r - \frac{\sigma^2}{2})dt + \sigma dW_t$$

or in the discrete-time form

$$s(t_{i+1}) = \ln S(t_{i+1}) = \ln S(t_i) + (r - \frac{\sigma^2}{2})(t_{i+1} - t_i) + \sigma(W_{i+1} - W_i)$$

Observe the additional term $\sigma^2/2$ in the drift.

Exercise 9 (Prove numerically Itô lemma)**.** Simulate $S(t_i)$ and $s(T_i) = \ln S(t_i)$.

1. Show that the distributions of $S(t_N)$ and $\hat{S} = \exp(s(t_N))$ are the same.
2. Compute log-returns $\ln X(t_N)/X(t_0)$ for both processes, $S(t_i)$ and $\hat{S}(t_i)$ and show that they both have the same *normal* distribution.

You can use the following parameters for the model:

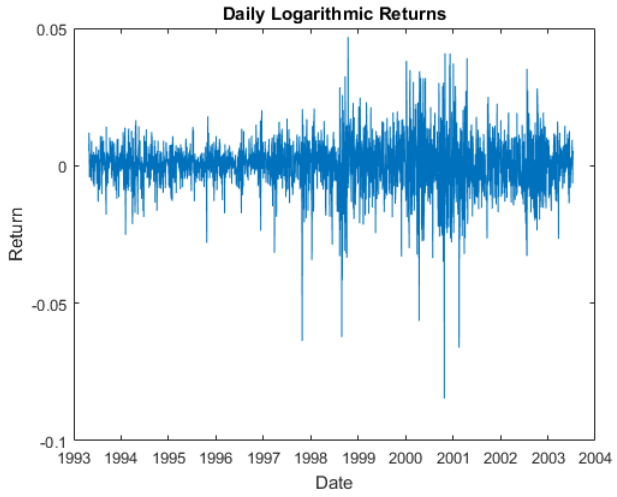
$$S_0 = 1$$
$$r = 0$$
$$\sigma = 0.2$$

You can also plot paths for $S(t_i)$ and $exp(s(t_i))$ and observe that they are close but there is a difference. Simulation of $\ln S$ is exact in this case, while simulation of $S(t_i)$ converges to $exp(s(t_i))$ only for small time steps.

3. Plot log-returns as a function of time for a single path. Take 1000 time steps to obtain a dense path. The volatility is pretty constant visually as one can expect.

3.3 Other stochastic processes

GBM model predicts log-normal returns as we could see in the previous exercise. This is typically not the case in practice. Look at the log-returns of DAX index and observe the volatility clustering, i.e. splitting in regions where volatility is small and regions where it is high. This indicates that volatility does not stay



constant in time. To account for that more sophisticated models are used. One class of them is called stochastic volatility models. There are different way to introduce stochastic volatility. Two very popular models are Heston and SABR. The first is popular in equity derivatives, the second in interest rate derivatives.

3.3.1 Heston model

Heston model has a mean reverting variance. The process for stock value S and variance ν are given by the following SDE.

$$\begin{aligned} dS(t) &= rS(t)dt + \sqrt{\nu(t)}S(t)dW_1(t) \\ d\nu(t) &= \kappa(\theta - \nu(t))dt + \xi\sqrt{\nu(t)}dW_2(t) \\ \langle dW_1, dW_2\rangle &= \rho dt \\ S(0) &= S_0 \\ \nu(0) &= \nu_0 \end{aligned}$$

Here r is a risk-free rate, κ is a speed of mean reversion, θ is a long term variance, and ξ is a volatility of volatility. Brownian motions W_1 and W_2 are correlated with correlation ρ.

3.3.2 SABR model

The stochastic α, β, ρ (SABR) model is given by the following SDE

$$\begin{aligned} dF(t) &= \sigma(t)F(t)^\beta dW_1(t) \\ d\sigma(t) &= \alpha\sigma(t)dW_2(t) \\ \langle dW_1, dW_2\rangle &= \rho dt \\ F(0) &= F_0 \\ \sigma(0) &= \sigma_0 \end{aligned}$$

Here F is a forward rate, usually defined in case of stock as $F = F(t,T) = Se^{r(T-t)}$. Forward rate is a rate at which two parties agree to buy/sell stock at a future time T. α is a volatility of volatility. β is some power of Forward rate that controls the skewness of return. Brownian motions are again correlated.

Exercise 10 (Stochastic volatility)**.** Choose one of the models, Heston or SABR. Simulate the paths to maturity $T = 1$ year.

1. Plot a single path and look if you can observe a volatility clustering.
2. Plot histograms of log-returns and compare them to the normal distribution with the same mean and variance. Observe the skewness and fat tails.

As initial parameters for the model choose the following:

Heston	SABR
$S_0 = 1$	$F_0 = 1$
$r = 0$	$r = 0$
$\nu_0 = 0.2^2$	$\sigma_0 = 0.2$
$\theta = \nu_0$	$\beta = 1$
$\rho = -0.7$	$\rho = -0.7$
$\xi = 0.5$	$\eta = 0.5$
$\kappa = 1$	

4 Option pricing

In the current computer practical, we will use Monte Carlo simulation to price options on stock. We start with the celebrated Black-Scholes model and continue with other models. In all models, we will follow almost the same route to emphasize similarities and differences between models.
A small remark on the simulation approach. In the previous computer practical we simulated the stock price directly. We will continue to do this also in the current practical for clarity of exposition. However in many real life situations, it is better to use some transformations to make the computation quicker and more robust. See Appendix for performing computation in log-space, for example.
The price of European type options, like call and put, can be computed using expectation under the risk-neutral measure as

$$V(0) = e^{-rT}\mathbb{E}_0^Q\left[\Phi(S_T)\right].$$

Here S_t is a random process of stock price, S_T is a stock price at maturity T, and $\Phi(S_T)$ is a payoff function. r is a risk-free rate. Let us express this expectation in terms of integral.

$$V(0) = e^{-rT}\mathbb{E}_0^Q\left[\Phi(S_T)\right] = e^{-rT}\int_0^\infty \Phi(S_T)\phi(S_T)dS_T,$$

where $\phi(S_T)$ is a probability density function (PDF). In most cases, we do not know the analytical expression for PDF. We can, however, approximate it by an empirical PDF obtained by Monte Carlo simulations. MC simulation gives discrete points at maturity T, S_i, and the corresponding empirical PDF can be expressed as a sum of Dirac delta functions $\delta(x - S_i)$

$$\phi_e(S_T) = \frac{1}{N}\sum_{i=1}^{N}\delta(S_T - S_i).$$

The empirical PDF converges to the actual PDF as N goes to infinity in the sense that the following integral converges to the desired expectation.

$$\int_0^\infty \Phi(S_T)\phi_e(S_T)dS_T \underset{N\to\infty}{\to} \int_0^\infty \Phi(S_T)\phi(S_T)dS_T.$$

Thus, we can approximate the expectation as

$$\begin{aligned} V(0) = e^{-rT}\mathbb{E}_0^Q\left[\Phi(S_T)\right] &\approx e^{-rT}\int_0^\infty \Phi(S_T)\phi_e(S_T)dS_T \\ &= e^{-rT}\int_0^\infty \Phi(S_T)\frac{1}{N}\sum_{i=1}^N \delta(S_T - S_i)dS_T \\ &= e^{-rT}\frac{1}{N}\sum_{i=1}^N \Phi(S_i), \end{aligned}$$

i.e. by computing a discounted mean of the payoff function with the generated values of stock price S_i.

4.1 Black-Scholes model

4.1.1 Call and put prices, comparison to Black-Scholes formula

Exercise 11 (Black-Scholes)**.** Compute European call and put prices using Monte Carlo simulation code from the previous practical for Geometrical Brownian motion under the risk-neutral measure

$$dS_t = (r-c)S_t dt + \sigma S_t dW \qquad\qquad S(0) = S_0,$$

where r is a risk-free rate, c is a continues dividend rate, σ is a volatility. The payoff functions are

$$\Phi(S) = \begin{cases} (S-K)^+ & \text{call,} \\ (K-S)^+ & \text{put.} \end{cases}$$

The expectation is approximates as shown above

$$V(0) = e^{-rT}\frac{1}{N}\sum_{i=1}^N \Phi(S_i).$$

The first thing one usually checks, if the code produces the right results for simple cases. Compute the forward price of the stock $F(T)$ at maturity $T = 1$ and compare it to the analytical expression

$$F(T) = e^{(r-c)T}S(0).$$

4.1.2 Put-Call parity

Graph 1: Check the put-call parity. Plot the graph for the difference between call and put prices $C(K) - P(K)$ for different strikes K and compare it to $e^{-cT}S(0) - e^{-rT}K$.

4.1.3 Comparison to Black-Scholes formula

In case of GBM, we can use analytical solution to compare to our Monte Carlo simulations. The celebrated Black-Scholes formula

$$\begin{aligned} V(0) &= \psi \left(e^{-cT} S \mathcal{N}(\psi d_1) - e^{-rT} K \mathcal{N}(\psi d_2) \right) \\ d_1 &= \frac{\ln S/K + (r-c)T + \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}} \\ d_2 &= d_1 - \sigma\sqrt{T} \\ \psi &= \begin{cases} +1 & \text{for call} \\ -1 & \text{for put.} \end{cases} \end{aligned}$$

is given in the function

`bsprice(S,K,r,c,v,T,pc)`

The function takes as input the current stock price S, a vector of strikes K, risk-free rate r, dividend rate c, volatility v, time to maturity T, and put or call indicator (-1 for put and +1 for call).
Graph 2: Plot on the graph the Monte Carlo prices of call and put for different strikes together with the prices of Black-Scholes formula.

4.1.4 Implied volatility

It is market standard to quote the price of options in terms of the volatility. For this, traders use the Black-Scholes formula. All other parameters are usually readily available on the market. Moreover, the stock price is most volatile, and by quoting volatility to the counterparty traders avoid the dependency on the stock price. When the deal is finalized, the actual stock price at that moment is put into the formula.
Graph 3: Implied volatility. We will use the Black-Scholes formula to compute the volatility implied from your MC call and put prices and plot it as a function of strike. Function

`bsimpvol(S,K,r,c,T,pc,price)`

does this. The function takes the same arguments as 'bsprice', except for the volatility, plus the prices of the options and returns the corresponding implied volatility. Compare on the graph the implied volatility and the Black-Scholes input volatility that we used in our simulations for different strikes.

4.2 SABR model

Use the Monte Carlo framework for SABR developed in the first practical to compute option prices.

Exercise 12 (SABR)**.**

4.2.1 Put-Call parity

Check that the simulated forward price is in line with the analytical formula.
Graph 1: Do as in the Black-Scholes case a check that the put-call parity is satisfied.

4.2.2 Call and put prices, comparison to Black-Scholes formula

Graph 2: Compare the simulated prices to the Black-Scholes formula. Use the volatility of log-returns in Black-Scholes formula. You will see the price difference and slightly different shape. It is not convenient to compare prices as they depend strongly on strike. We will use implied volatility for more proper comparison.

4.2.3 Implied volatility

Graph 3: Compute the implied volatility for SABR model. Plot it together with the Black-Scholes volatility as a function of strike. Observe the smile and the skew of the volatility $\sigma_{imp}(K)$. Play with different parameters of the SABR model to observe which parameters are responsible for the level, slope (skew), and curvature (smile) of the volatility smile.

4.2.4 Volatility approximation

The reason why the SABR model is so popular is the existence of an analytical approximation for implied volatility in SABR model. This makes the calibration of the model to the market quotes very easy. The derivation of this approximation is very technical [1], and is beyond this course.
Graph 4: Use function 'sabrVolatility(S,K,r,c,alpha,beta,rho,eta,T)' to compute the implied volatility using this approximation and compare to the implied volatilities obtained from your simulations.

4.3 Heston model

Use the Monte Carlo framework for SABR developed in the first practical to compute option prices.
In case of the Heston it is not that easy to guaranty the positivity of the volatility. There are different computational schemes to do it, which are beyond this

course. We will use a simple truncation to achieve this goal, i.e. we will use $v^+ = \max(v, 0)$ in our simulations.

$$\begin{aligned} dS_t &= (r-c)S_t dt + \sqrt{\nu_t^+} S_t dW_1 && S(0) = S_0 \\ d\nu_t &= \kappa(\theta - \nu_t^+)dt + \xi\sqrt{\nu_t^+} dW_2 && \nu(0) = \nu_0 \\ \langle dW_1, dW_2 \rangle &= \rho dt \end{aligned}$$

where we truncated variance ν to positive values ν^+.

Exercise 13 (Heston)**.**

4.3.1 Put-Call parity

Check that the simulated forward price is in line with the analytical formula. Graph 1: Do as in the Black-Scholes case a check that the put-call parity is satisfied.

4.3.2 Call and put prices, comparison to Black-Scholes formula

Graph 2: Compare the simulated prices to the Black-Scholes formula. Use the volatility of log-returns in Black-Scholes formula.

4.3.3 Implied volatility

Graph 3: Compute the implied volatility for the Heston model. Plot it together with the Black-Scholes volatility as a function of strike. Observe the smile and the skew of the volatility $\sigma_{imp}(K)$. Play with different parameters of the Heston model to observe which parameters are responsible for the level, slope (skew), and the curvature (smile) of the volatility smile.

4.4 Local volatility model

SABR and Heston are parametric models and in many cases do not allow to fit all market quotes for all strikes and maturities exactly. There is another type of model, called Local Volatility model, which allows to fit all the available quotes on the market . In this model the volatility is a deterministic function of stock price and time, $\sigma_{loc}(S_t, t)$:

$$dS_t = (r-c)S_t dt + \sigma(S_t, t)S_t dW \qquad S(0) = S_0.$$

Generally, it is a daunting task to compute the local volatility function for real quotes. We however will follow an easy way and assume that the market quotes are completely in line with the SABR model. Then we can use the SABR approximation for the Black-Scholes implied volatility to compute the local volatility function. The corresponding function 'sabr2LocalVol(S0,K,r,c,alpha,beta,rho,nu,t)' returns the local volatility $\sigma(S_t = K, t)$ for a given vector of strikes K and maturity t.

Again we perform the Monte Carlo simulation is as in case of GBM. Only instead of the constant volatility, we put in the calculations the local volatility, corresponding to the current time step and the current stock price at this time and in this path.

Exercise 14 (Local volatility)**.**

4.4.1 Call and put prices, comparison to Black-Scholes formula

Graph 1: Calculate call and put prices and compare them to the Black-Scholes. Plot also the call and put prices computed with the volatility given by the SABR approximation.

4.4.2 Implied volatility

Graph 2: Compute the implied volatility and compare it to Black-Scholes and to the SABR approximation.

4.5 ⋄ Pricing Barrier option

To show how different models affect pricing of exotic derivatives we look at barrier option pricing in SABR and local volatility models. These models give the very same European call and put prices for all strikes and maturitites. So, we may expect that the barrier prices are the same as well, but they are not. This shows that the choice of the model is very important for pricing exotic options.
A barrier option has the same payoff as vanilla option, however an Out option can be knocked out if the stock price crosses a predefined barrier during the life time of the option. If the stock price crosses the barrier at any time during the life time, the option ceases to exist, and the holder does not get anything even if the vanilla option is in the money. The payoff of the up-and-out option for example is

$$\Psi(S_T) = (S_T - K)^+ \mathbb{I}(S_t < B, \forall t \leq T),$$

where $\mathbb{I}(x)$ is an indicator function.
Our Monte Carlo simulation is very suitable to compute the price of such exotic. In our simulations we need only to add an additional variable for each path i, say I_i, with initial value $I_i(0) = 1$, and check at every step if the barrier B is crossed. In the latter case, I_i becomes zero for the rest of the life time. The expectation is then taken over the product of call payoff and this function I_i.
Compute and compare the prices of barrier options in the two models. Compare the difference to the difference in vanilla call prices for the same strikes.

4.6 ⋄ Convergence of Monte Carlo simulations

We did not pay much attention to numerical aspects of Monte Carlo simulation. One interesting question in this respect can be how many paths do we need to

achieve a particular accuracy in price. Compute the price of a call option in the Black-Scholes model with different number of paths N and show that the error decreases as $1/\sqrt{N}$.

References

[1] P. S. Hagan, D. Kumar, A. S. Lesniewski, and D. E. Woodward. Managing smile risk. *Wilmott Magazine*, September:84–108, 2002.

A Computations in the log-space

A.1 SABR model

In the first computer practical we compute dynamics for stock price S. Due to the final steps in our simulations, the stock and volatility can become negative. This results in some case in run-time error. In case of the SABR model it is easy to overcome by switching the simulation into log-price and log-volatility space in the following way.
The original model is defined as

$$\begin{aligned} dF &= \alpha F^{\beta} dW_F & F(0) &= F_0 \\ d\alpha &= \nu \alpha dW_\alpha & \alpha(0) &= \alpha_0 \\ \langle dW_F, dW_\alpha \rangle &= \rho dt \end{aligned}$$

We introduce the new variables, f_t and a_t

$$\begin{aligned} f_t &= \ln F_t & F_t &= e^{f_t} \\ a_t &= \ln \alpha_t & \alpha_t &= e^{a_t}. \end{aligned}$$

Then using the multidimensional Itô lemma, we get SDE for f_t and a_t

$$\begin{aligned} df &= -\frac{1}{2} e^{2a+2(\beta-1)f} dt + e^{a+(\beta-1)f} dW_F & f(0) &= \ln F_0 \\ da &= -\frac{\nu^2}{2} dt + \nu dW_\alpha & a(0) &= \ln \alpha_0 \\ \langle dW_F, dW_\alpha \rangle &= \rho dt \end{aligned}$$

This approach guaranties that $F_t = e^{f_t}$ and $\alpha_t = e^{a_t}$ stay always positive.

A.2 Heston model

In case of the Heston it is not that easy to guaranty the positivity of the volatility. There are different computational schemes to do it, which are beyond this course. We will use a simple truncation to achieve this goal, i.e. we will use $v^+ = \max(v, 0)$ in our simulations. it is still useful to work in log-price space, so our simulations will be as follows.

The original model is defined as

$$\begin{aligned} dS_t &= (r-c)S_t dt + \sqrt{\nu_t} S_t dW_1 & S(0) &= S_0 \\ d\nu_t &= \kappa(\theta - \nu_t)dt + \xi\sqrt{\nu_t} dW_2 & \nu(0) &= \nu_0 \\ \langle dW_1, dW_2 \rangle &= \rho dt \end{aligned}$$

We define the log price as

$$s_t = \ln S_t \qquad\qquad S_t = e^{s_t}$$

And after using the Itô lemma, the simulation SDEs become

$$\begin{aligned} ds_t &= \left(r - c - \frac{1}{2}\nu_T^+\right) dt + \sqrt{\nu_t^+} dW_1 & s(0) &= \ln S_0 \\ d\nu_t &= \kappa\left(\theta - \nu_t^+\right) dt + \xi\sqrt{\nu_t^+} dW_2 & \nu(0) &= \nu_0 \\ \langle dW_1, dW_2 \rangle &= \rho dt \end{aligned}$$

where we truncated variance ν to positive values ν^+.