Computational Finance and its implementation in Python with applications to option pricing.
Exercise Handout 2

Lecture: Dr. Andrea Mazzon

Exercise 1

Consider a continuous, adapted stochastic process $X = (X_t)_{t\geq 0}$ with dynamics

$$dX_t = rX_t dt + \sigma X_t dW_t, \quad t \geq 0,$$

where $r, \sigma > 0$ and $W = (W_t)_{t\geq 0}$ is a Brownian motion. Suppose you want to price a down-and-out call option with underlying X, maturity $T > 0$, strike K and lower barrier B_u.
We have seen that the dynamics of $X = (X_t)_{0\leq t\leq T}$ can be approximated by N time steps of a Binomial model with parameters

$$u = e^{\sigma\sqrt{T/N}}, \qquad d = 1/u, \qquad \rho = e^{r\sqrt{T/N}},$$

for N large enough.
In a similar way as in

`binomialmodel.optionvaluation.americanOptionPriceConvergence`,

check if the valuation of a down-and-out option performed with the code you wrote to solve Exercise 1 of Handout 1, or the one you can find in

`binomialmodel.optionvaluation.knockOutOption`,

approximates well the analytic one for large N.
Also compare this approximation with the one obtained by using the Monte-Carlo discretization and simulation of X, that we test in

`binomialmodel.optionValuation.knockoutOptionTest`,

both in terms of time and accuracy.

Exercise 2

Consider the market where you have zero interest rate $\rho = 0$, and where the risky asset $\bar{S} = (\bar{S}_t)_{t=0,1,\dots,T}$ is defined by

$$\bar{S}_t = \log(S_t),$$

where $S = (S_t)_{t=0,1,\dots,T}$ is the binomial model described in the script, i.e.,

$$S_t = S_0 \cdot Y_1 \cdot \dots \cdot Y_t, \quad t = 1, \dots, T,$$

where Y_t can take the two values d, u with $0 < d < 1 < u$, for any $t = 1, \dots, T$, and $(Y_t)_{t=1,\dots,T}$ are i.i.d. and such that Y_{t+1} is independent of $\mathcal{F}_t$.
In Exercise 2 of Handout 1 you had to find the risk neutral probability Q of such a market, i.e., the probability

$$q = Q(Y_t = u)$$

such that $\bar{S}$ is a martingale under Q.
Implement now such a model in Python, similarly to what has been done in the classes of

`binomialmodel.creationandcalibration`.

If you like, you can just focus on the `binomialModelSmart` implementation. In particular, do the following:

- Check that $\bar{S}$ is indeed a martingale under the measure Q you found.
- Print the evolution of the probability $Q(\bar{S}_j > \bar{S}_0) = Q(S_j > S_0)$. Compare it with the one we had found under the martingale measure for S. What do you note?