**Computational Finance and its implementation in Python with applications to option pricing.**
**Final project 3**

**Lecture: Dr. Andrea Mazzon**

## General guidelines

The final project for the lecture *Computational Finance and its implementation in Python with applications to option pricing* consists of solving at least two of the following problems, and discussing and presenting the solution in a small oral exam/seminar.

The exposition is supposed to be individual. You are of course very welcome to ask if you need any help. Please note that your solution has not be one hundred percent exact, the most important thing is that you are able to discuss and motivate your results.

## Problem 1

This problem follows the lines of what you have already done in Handout 1, Handout 2 and Handout 3. 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 $\mathcal{F}_t$.

(a) 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 $Q$. Check that $0 < q < 1$.

(b) Implement such a model in Python, similarly to what has been done in the classes of

`binomialmodel.creationandcalibration`.

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?
- Check the implementation using Monte-Carlo, in particular for a large number of times. Compare it to the one we have seen in the lecture when we considered the market with $S$.

(c) Valuate a call option for some general parameters under this new model.

(d) Consider the model above with $N$ time steps, for $N$ large enough, and parameters

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

Find the dynamics

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \quad t \geq 0, \tag{2}$$

of a process $X$ which is approximated by $\bar{S}$ under this setting.

Is it possible to compute the analytic value of a call option under these dynamics for $X$? If yes, check that the price obtained under $\bar{S}$ with parameters (1) for large $N$ approximates this price.

(e) Simulate the process in (2) and compute the approximate price of the call with the Monte-Carlo method.

(f) Find the PDE associated to the SDE (2). Solve it numerically, with a scheme at your pleasure, and use this method in order to approximate the price of a call option.

(g) Compare all the approaches above with respect to accuracy and time needed to compute the price of the call option.

## Problem 2

Write a class extending the one in

`finitedifferencemethods.pricingWithPDEs`,

in order to solve the PDE (14) at page 131 in the script with implicit Euler. The main issues here are two:

- Give some appropriate artificially boundary conditions.
- Solve a nonlinear system at every time step.

Simulate the $G$-Brownian motion using this class to solve the PDE instead of the one in `gPDESolution` (you don't have to change anything else in `gBrownianMotion`). Compare the two approaches in terms of time needed to simulate a certain number of trajectories of the $G$-Brownian motion and also in terms of accuracy (you can for example consider the average of the realizations of the Brownian motion for the specific case when $\underline{\sigma} = \overline{\sigma}$).
Also investigate how changing parameters like the number of points on which we compute the distribution first, or time and space discretization for the PDE, affects your result.

## Problem 3

You might remember the issue we have noted doing our experiments in the module

`binomialmodel.optionvaluation.KnockoutOptionConvergence`:

if we want to valuate a down-and-out option via a Binomial model, approximating a Black-Scholes model with interest rate $r \geq 0$ and volatility $\sigma > 0$ for a large number of time steps, the price oscillates a lot if the barrier is close to the initial value. However, we have seen that actually the price of the option is approximated very well for some specific numbers of time steps $N$.
In the repository you can find the paper *A discrete-time algorithm for pricing double barrier options*, from M. Costabile, that investigates exactly this problem. In particular, Section 3 is devoted to the explanation of a method found by P.P. Boyle and S.H. Lau in 1994 (unfortunately their paper is not available for free) which permits to find the values of $N$ for which the binomial model better approximate the price of the option. The main intuition is that for these values, *the barrier is close to but just above a layer of horizontal nodes in the tree.*
Your goal here is to:

- understand the formula by which these numbers are computed, and the motivation for this formula;
- write a function or method that computes these numbers, up to a given $N_{\max}$, for given initial value $S$ of the underlying, (lower) barrier $S$, maturity $T$, log-volatility $\sigma$ and interest rate $r$;
- check that the numbers you compute with the method/function above are indeed the ones that minimize the error. For example, you can do that by running again

  `binomialmodel.optionvaluation.KnockoutOptionConvergence`

  and looking at the values of $N$ for which the price is close to the analytic one: these should correspond to the ones given by your method.
- Optional: do the same for the extension to double barrier options in Section 5.