Numerical Methods for Financial Mathematics. **Exercise Handout 10**
Lecture: Prof. Dr. Christian Fries, Exercises: Dr. Andrea Mazzon, Tutorium: Roland Bachl
Sommersemester 2021

Exercise 1

This exercise might be done during the Tutorium

Generalize the implementation of the class `BrownianMotion` you wrote for Exercise 2 Handout 9 in order to:

- generate a possibly multi-dimensional Brownian motion (the components have to be independent);
- allow for a time discretization
$$t_0 < t_1 < \cdots < t_n$$
with possibly non-constant time step lengths $t_i - t_{i-1}$, $i = 1, \dots, n$;
- be able to get the value of the Brownian motion (both a single path and all the paths) for a specific `double` time, i.e., giving the time t_i instead of the index i. (But still keep the methods where you give the index i).

Test then your implementation: you can do the same checks of Handout 9, giving now the time instead of the time index, and also checking that two components W^i and W^j are independent for $i \neq j$.
Hint: you can use the Finmath library classes and interfaces: for example, you can exploit the methods of `TimeDiscretization`.

Exercise 2

This exercise might be done during the Tutorium

The abstract class `AbstractProcessSimulation` that you can find in

`com.andreamazzon.exercise10.approximationschemes`

is devoted to the discretization and simulation of the solution of a continuous, one-dimensional SDE driven by a one-dimensional Brownian motion.
It has some `public` methods (some of them already implemented, some others to be implemented by yourself) and a `private` method `generate` that must fill the field `paths`, consisting of a one-dimensional array of `RandomVariable` objects. The idea here is that at every time iteration the drift of the process and its diffusion are got by calling the `protected abstract` methods

`RandomVariable getDrift(RandomVariable lastRealization, int timeIndex)`

and

`RandomVariable getDiffusion(RandomVariable lastRealization, int timeIndex)`

implemented in the derived classes. Drift and diffusion are then summed to the last realization of the process. Implement this method according to the comment you find in the class.
Extend the class above in three classes, all simulating a geometric Brownian motion

$$dX_t = \mu X_t dt + \sigma X_t dW_t, \quad t \geq 0. \tag{1}$$

In particular, in one class you have to use the Euler scheme, in one class the log Euler scheme (i.e., you simulate the logarithm of the process by Euler) and in the third one the Milstein scheme. Note that, for all the derived classes, you have to give the values for μ and σ in (1).
Hint: in order to solve this exercise, you can make extensive use of the methods of the `RandomVariable` interface, for example to perform operations between objects of type `RandomVariable`.

Exercise 3

Write a class `CallOption` whose goal is to compute the Monte-Carlo discounted price of an european call option, for given strike, maturity and risk-free rate, whose underlying is given by an instance of `AbstractProcessSimulation`. Check then the errors with respect to the analytical price when the underlying is the process in (1), simulated using the three classes above.

Hint: in `CallOption`, you are free to decide which of the data you need to calculate the value of the option you want to be fields of the class and which you want to be given as an argument to the method computing the value of the option. Also note that, when you do the test, the value of μ in (1) must be equal to the risk-free rate.