Project 1 (30% of the final mark)

Asian options pricing using procedural programming

C++ Programming with Applications to Finance Autumn 2021

The aim of this project is to create a program in C++ that can be used to study the prices of Asian call and put options on an underlying asset driven by a **binomial model** in discrete time. Your program should be accompanied by end-user and developer documentation. The project is aimed at testing your knowledge from the Term 1 of the course and must be implemented in the procedural programming paradigm. You **must not** use object-oriented approach when creating this project, **including vectors** (please, stick to the arrays and pointers instead).

An Asian option (or average value option) is a special type of option contract. For Asian options the payoff is determined by the average underlying price over some pre-set period of time. This is different from the case of the usual European option and American option, where the payoff of the option contract depends on the price of the underlying instrument at exercise; Asian options are thus one of the basic forms of exotic options.

(source: Wikipedia)

We assume that the price of the underlying is strictly positive at the moment 0 ($S_0 = S(0) > 0$) and at each moment *t* it can either move up by a factor *(1+U)* or down by a factor *(1+D)*. As always in the binomial model there is a risk-free security growing by a factor *(1+R)* during each time step. Here we assume that *-1<D<R<U*. In what follows, for "moving up by a factor *(1+U)*" we can also use expressions "moving up" or "growing" (although we understand that it is possible that *(1+U)<1* and the price drops, the terms "moving up" or growing reflect rather the fact that *(1+U)* is higher than *(1+D)*). Similarly, we will sometimes say "moving down" or "falling" instead of "moving down by a factor *(1+D)*". Detailed description of the binomial model can be found in Capinski & Zastawniak (2012) pp. 1-2.

Further, we use *S(t)* for the price of the underlying asset at moment *t*, *K* for the strike, *T* for maturity and *A(0,T)* for the average price for the period *[0, T]*.

Fixed strike Asian call payoff is given by $C_{fixed\ strike}(T) = \max\left(A(0,T) - K, 0\right)$.

Fixed strike Asian put payoff is given by $P_{fixed\ strike}(T) = \max\left(K - A(0,T), 0\right)$.

Two different types of averaging can be used for Asian options:

- arithmetic mean $A(0,T)=\frac{1}{T}\sum_{t=1}^{T}S(t)$;
- geometric mean $A(0,T)=\sqrt[T]{\prod_{t=1}^{T}S(t)}=\left(\prod_{t=1}^{T}S(t)\right)^{1/T}$.

The price of the Asian option equals to its discounted expected payoff. The expectation is taken with respect to the risk-neutral probability and the discounting is done with respect to the risk-free return:

$$price_{Asian\ call}(T,K)=\frac{1}{(1+R)^{T}}E\left[\max\left(A(0,T)-K,0\right)\right],$$

$$price_{Asian\ put}(T,K)=\frac{1}{(1+R)^{T}}E\left[\max\left(K-A(0,T),0\right)\right].$$

General hints and tips

- Read the submission instructions at the end of this document before starting work on the project.
- Read through all the tasks before starting work on the project. Tasks do not have to be completed in the order that they are listed here.
- You are free to recycle any code produced by yourself during the module. You are also free to use any code provided in Moodle during the module, provided that such code is acknowledged.

This project contributes 30% to the mark for this module. The deadline for submission is **23:59 on** Monday, 28th Feb. 2022 The marks for each of the tasks below will be split into 60% for coding style, clarity, accuracy and efficiency of computation, and 40% for documentation (including comments within the code) and ease of use.

Task.

Your program should ask the user to enter the value of the parameters S_0, U, D, R, K and N (the number of steps to expiry – the same as T in the description of the model).

Your program should then calculate the prices of Asian put and call options with different types of averaging: arithmetic and geometric.

There are three header files attached to the task. They have prototypes of the functions you need to implement. For the input functions you can refer to Project 7 from Capinski & Zastawniak (2012) (do not forget to include the reference!).

For pricing the option one needs to find the discounted value of the average payoff. The main difficulty you may face is that the payoff is defined by all values of the underlying asset along the path, so the payoffs must be computed along each path separately.

Clearly, for the N-period model there are 2^N different paths, and one can easily construct a bijection between these paths and the set of arrays of length N consisting of zeros and ones. For example, if $N = 3$ then then an array {0,0,1} may correspond to the underlying asset growing in the first two periods and then falling in the period 3, i.e. getting $S_1=S_0(1+U)$, $S_2=S_0(1+U)^2$ and $S_3=S_0(1+U)^2(1+D)$, if one codes an upward moving of the asset with 0 and a downward moving with 1 (or falling in the first two periods and then growing in the period 3 if one codes a downward moving of the asset with 0 and an upward moving with 1, or perhaps, even the elements of the array may describe the changes in the underlying asset in the opposite order with the last elements corresponding to the moves in the first periods – you can use any approach that suits you, but you need to describe it carefully and be consistent). For this purpose you should use the function GenPathByNumber mentioned in AvgPrices.h.

After successfully coding the path with number x as an array of zeros and ones you can generate a vector of prices along this path with GenPricesByPath, these prices will then be used for finding arithmetic (geometric) average using ArAverage (GeAverage) and computing the payoffs. Next, you need to calculate payoffs for each path separately by calling AsianCallPayoff and AsianPutPayoff functions.

The probability of moving along the path should be calculated with GenProbabilityByPath. The probability of getting a path along which the price moves up i times and down $(N-i)$ times equals $q^i(1-q)^{N-i}$, where q is the risk-neutral probability $q = \dfrac{U-R}{U-D}$.

Finally, you need to combine the results of the previous steps and write the function Price that returns the price of the Asian option of the selected type with selected type of averaging. A price of an Asian option equals to its discounted expected payoff. If by p_j we denote the probability of getting path j and by $payoff_j$ the payoff on this path then $E\left(payoff\right) = \sum_{j \in Paths} p_j \, payoff_j$.

Discounting is done with respect to the risk-free return: $price = \dfrac{E\left(payoff\right)}{\left(1+R\right)^N}$

The main function should ask for inputs and price two Asian call options (with arithmetic and geometric averaging) and two Asian put options (with arithmetic and geometric averaging).

Submission instructions

Submit your work by uploading it in Moodle by **23:59 on** Monday 28th Feb. 2022

Format

Submit the code as a single compressed .zip file, including all Code::Blocks project (.cbp) files (if you used Code::Blocks), source code and header (.cpp and .h) files, all residing in a single parent directory. The .zip file should preserve the subdirectory structure of the parent directory (that is, the subdirectory structure must be automatically recreated when unzipping the file).

The code should be accompanied by detailed documentation, split into two parts:

1. **Code developer documentation** containing information to help understand the code. It should contain the following:
    - Description of the file structure.
    - Description of the available functions.
    - Instructions on how to extend the code by adding new options or derivatives.
    - Test runs, including tables. Graphs are optional but encouraged.

    The developer documentation should not include extensive extracts from the code (brief snippets are perfectly fine, if typeset correctly—no screenshots!), and there is no need to describe the mathematical methods in any detail. It is expected that the developer documentation will be less than 5 pages in length.

2. The **end user instructions** should contain instructions on how to use the compiled .exe program, how to input data and how the results are presented, and a brief description of the methods implemented. The contents of this document should be appropriate for a reader who is not familiar with C++. It is expected that the end user instructions will be less than 2 pages in length.

The documentation files must be submitted in .pdf format (two separate .pdf files containing developer documentation and user instructions) and uploaded in Moodle separately from the code .zip file.

It is advisable to allow enough time (at least one hour) to upload your files to avoid possible congestion in Moodle before the deadline. In the unlikely event of technical problems in Moodle please email your .zip files to eric.dykeman@york.ac.uk before the deadline.

Code usage permissions and academic integrity

You may use and adapt any code submitted by yourself as part of this module, including your own solutions to the exercises. You may use and adapt all C++ code provided in Moodle as part of this module, including code from Capinski & Zastawniak (2012) and the solutions to exercises. **Any code not written by yourself must be acknowledged and referenced both in your source files and developer documentation.**

This is an individual project. You may **not** collaborate with anybody else or use code that has not been provided in Moodle. You may not use code written by other students. Collusion and plagiarism detection software will be used to detect academic misconduct, and suspected offences will be investiaged.

If things go wrong

Late submissions will incur penalties according to the standard University rules for assessed work.

It may prove impossible to examine projects that cannot be unzipped and opened due to file corruption, or projects that cannot be compiled and run due to coding errors. **In such cases a mark of 0 will be recorded.** It is therefore essential that all project files and the directory structure are tested thoroughly on your machine before being submitted in Moodle. It is advisable to run all such tests starting from the .zip files about to be submitted, and using a different computer to that on which the files have been created.

All files must be submitted inside the zipped project directory, and all the files in the project directory should be connected to the project. A common error is to place some files on a network drive rather than in the submitted directory. Please bear in mind that testing on a lab computer may not catch this error if the machine has access to the network drive. However, the markers would have no access to the file (since they have no access to your part of the network drive) and would be unable to compile the project.

References

Capinski, M. J. & Zastawniak, T. (2012), *Numerical Methods in Finance with C++*, Mastering Mathematical Finance, Cambridge University Press.