# C++ Programming with Applications to Finance Programming Project

Deadline: 27 April 2017

This project contributes 30% towards the final mark for the module.

## Background

The two-dimensional Black-Scholes model is

$$S_0(t) = S_0(0)e^{(r-\frac{1}{2}\sigma_0^2)t+\sigma_0 W_0(t)},$$
$$S_1(t) = S_1(0)e^{(r-\frac{1}{2}\sigma_1^2)t+\sigma_1\rho W_0(t)+\sigma_1\sqrt{1-\rho^2}W_1(t)},$$

where $W_0$ and $W_1$ are two independent Brownian motions, $r \geq 0$ is the interest rate, $\sigma_0 > 0$ and $\sigma_1 > 0$ are the volatilities of $S_0$ and $S_1$ respectively, and $\rho \in [-1, 1]$ is the correlation between the logarithmic returns of $S_0$ and $S_1$. The current stock prices $S_0(0)$ and $S_1(0)$ are required to be positive.

### Calibration

Before the model can be used to price options, it needs to be calibrated, that is, the values of the parameters should be calculated to match market data. Quotes for $S_0(0)$, $S_1(0)$ and $r$ are readily available, but not so for $\sigma_0$, $\sigma_1$ or $\rho$. A popular way to calibrate $\sigma_0$, $\sigma_1$ and $\rho$ is to estimate them from historical data. Suppose that stock prices $S_{0,i}, S_{1,i}$, $i = 0, 1, \ldots, m$ have been recorded at some fixed time intervals of length $\tau > 0$ measured in years (for example, $\tau = 1/12$ for monthly recorded stock prices). The log returns on these prices over each time interval of length $\tau$ are

$$r_{k,i} = \ln\left(S_{k,i}/S_{k,i-1}\right)$$

for $k = 0, 1$ and $i = 1, \ldots, m$. The sample variances of these returns are $s_0^2$ and $s_1^2$, where

$$s_k^2 = \frac{1}{m-1}\sum_{i=1}^{m} r_{k,i}^2 - \frac{m}{m-1}\overline{r}_k^2$$

and

$$\overline{r}_k = \frac{1}{m}\sum_{i=1}^{m} r_{k,i}.$$

The sample covariance of these returns is

$$s_{01} = \frac{1}{m-1}\sum_{i=1}^{m} r_{0,i}r_{1,i} - \frac{m}{m-1}\overline{r}_0\overline{r}_1.$$

The sample variance $s_k^2$ estimates the variance $\sigma_k^2\tau$ of the log return on the stock price $S_k$ over a time interval of length $\tau$. We can therefore make the estimation

$$\sigma_k = \frac{s_k}{\sqrt{\tau}}.$$

This is called the *historical volatility.* The historical correlation is then

$$\rho = \frac{s_{01}}{\tau\sigma_0\sigma_1}.$$

## Option pricing

Once the model has been calibrated, it can be used to price a variety of options. Many of the most commonly traded options depend on the spread $S_0 - S_1$, the minimum $\min\{S_0, S_1\}$ or the maximum $\max\{S_0, S_1\}$ of the prices of the two stocks. Here are three examples of payoff functions:

| | |
|---|---|
| Spread call | $f(S_0, S_1) = \max\{S_0 - S_1 - K, 0\}$, |
| Min call | $f(S_0, S_1) = \max\{\min\{S_0, S_1\} - K, 0\}$, |
| Max call | $f(S_0, S_1) = \max\{\max\{S_0, S_1\} - K, 0\}$. |

While explicit option pricing formulae can be found for some options, the only viable pricing method for most options is approximation.

The continuous-time model can be approximated by a two-dimensional tree with step size $h > 0$; note that $h$ is usually different from the parameter $\tau$ used in calibration. In such a tree, the stock prices $S_0^h(n)$ and $S_1^h(n)$ at every step $n = 0, 1, \dots$ of the tree approximate the prices $S_0(nh)$ and $S_1(nh)$ at time $nh$ in the continuous-time model. There are $(n+1)^2$ different possibilities for the stock price at time step $n$ in the tree model; each possibility is called a *node.* Each node at time $n$ can be represented uniquely in the form $(j_0, j_1)$ where $j_0 \in \{0, \dots, n\}$, $j_1 \in \{0, \dots, n\}$. Every node $(j_0, j_1)$ at time step $n$ has four successors at time step $n+1$, namely $(j_0, j_1)$, $(j_0 + 1, j_1)$, $(j_0, j_1 + 1)$ and $(j_0 + 1, j_1 + 1)$. Stock prices in the tree are given by the formulae

$$\begin{aligned} S_0^h(n; j_0, j_1) &= S_0(0)e^{\alpha_0^h n + \beta_0^h(2j_0 - n)}, \\ S_1^h(n; j_0, j_1) &= S_1(0)e^{\alpha_1^h n + \beta_1^h(2j_0 - n) + \beta_2^h(2j_1 - n)} \end{aligned}$$

for all $n$ and $(j_0, j_1)$, where

$$\begin{aligned} \alpha_0^h &= (r - \tfrac{1}{2}\sigma_0^2)h, & \beta_0^h &= \sigma_0\sqrt{h}, \\ \alpha_1^h &= (r - \tfrac{1}{2}\sigma_1^2)h, & \beta_1^h &= \sigma_1\rho\sqrt{h}, \\ & & \beta_2^h &= \sigma_1\sqrt{(1-\rho^2)h}. \end{aligned}$$

The risk-neutral probability of any node $(j_0, j_1)$ at time $n$ is

$$Q^h(n; j_0, j_1) = \binom{n}{j_0}(q_0^h)^{j_0}(1 - q_0^h)^{n-j_0} \times \binom{n}{j_1}(q_1^h)^{j_1}(1 - q_1^h)^{n-j_1},$$

where $\binom{n}{j_0} = \frac{n!}{(n-k)!k!}$ and $q_0^h$ and $q_1^h$ satisfy the system

$$\begin{aligned} e^{rh}S_0(0) = (1 - q_0^h)(1 - q_1^h)S_0^h(1; 0, 0) + q_0^h(1 - q_1^h)S_0^h(1; 1, 0) \\ + (1 - q_0^h)q_1^h S_0^h(1; 0, 1) + q_0^h q_1^h S_0^h(1; 1, 1), \quad (1) \\ e^{rh}S_1(0) = (1 - q_0^h)(1 - q_1^h)S_1^h(1; 0, 0) + q_0^h(1 - q_1^h)S_1^h(1; 1, 0) \\ + (1 - q_0^h)q_1^h S_1^h(1; 0, 1) + q_0^h q_1^h S_1^h(1; 1, 1). \quad (2) \end{aligned}$$

The tree is said to be free of arbitrage if $0 < q_0^h, q_1^h < 1$.

Suppose that a European option with payoff function $f$ in the continuous-time model offers the payoff $f(S_0(T), S_1(T))$ at time $T$. Its fair price at time 0 can be approximated by

$$\pi^h(f) = e^{-rhN} \sum_{j_0=0}^{N} \sum_{j_1=0}^{N} Q^h(N; j_0, j_1) f(S_0^h(N; j_0, j_1), S_1^h(N; j_0, j_1)) \tag{3}$$

in the two-dimensional tree with time step $h = \frac{T}{N}$.

## Project tasks

Based on the above information, complete the following tasks. You should use the header file `Project.h`, which should be included in your project without change.

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. Tasks do not have to be completed in the order that they are listed here. Credit will be given for part completion of each task.

### 1. Calibration (20%)

Calibrate the Black-Scholes model by computing the historical volatilities $\sigma_0$ and $\sigma_1$ and the historical correlation $\rho$ from a given sequence of historical stock prices $S_{0,i}$, $S_{1,i}$, $i = 0, 1, \ldots, m$ recorded at equal time intervals of length $\tau$. Your program should ask the user to enter a value for $\tau$, and display the values of $\sigma_0$, $\sigma_1$ and $\rho$ once they have been calculated.

Your program should read the time series of stock prices $S_{0,i}$, $S_{1,i}$ $i = 0, 1, \ldots, m$ from a file called `data.csv` in which each pair of numbers is listed on a separate line and separated by a comma, as in the sample `data.csv` file provided. (The actual `data.csv` file used for testing the program by the marker will contain a different and much longer time series.) Include a `data.csv` file in your submission, placed in the directory where your program will be looking for it, but do not hard wire the absolute path to the file into your program, as this path is likely to be different on the marker's computer. Explain in the end-user instructions how to prepare and provide the `data.csv` file so that your program can read it.
*Hint.* Refer to a good C++ manual to learn about reading input files. You may wish to investigate the standard libraries `iostream` and `fstream`.

### 2. Two-dimensional Black-Scholes model (10%)

Your program should ask the user to enter values for $S_0(0)$, $S_1(0)$ and $r$. Implement all member functions of the class `BSModel2` that aren't defined in `Project.h`.

### 3. Tree approximation (25%)

Build the option pricing tree by implementing all member functions of the class `CorrBinModel` that aren't defined in `Project.h`.

Choose an appropriate method for computing $q_0^h$ and $q_1^h$ accurately to within `EPSILON` (defined in `Project.h`). You should provide details of the method used to solve (1)–(2) in the developer documentation.
*Hint.* Note that $S_0^h(1; 0, 0) = S_0^h(1; 0, 1)$ and $S_0^h(1; 1, 0) = S_0^h(1; 1, 1)$, and so (1) can be rearranged to give

$$q_0^h = \frac{e^{rh} S_0(0) - S_0^h(1; 0, 0)}{S_0^h(1; 1, 0) - S_0^h(1; 0, 0)}.$$

### 4. Option payoff (10%)

Define and implement subclasses of the `Payoff` class for each of the example payoffs given. These subclasses will be used for both European and American option pricing.

### 5. Option pricing (15%)

Implement the pricing formula (3) for European options.

The function `PriceAmerican` in `Project.h` can be used to price American options. You should analyse this function, and the documentation should include a full description of the method used.

### 6. Numerical study of option prices (20%)

Your program should ask the user to enter a value for the expiry date $T$, choose an option payoff from the list of example payoffs, and enter a minimum strike $K_{\text{min}}$ and maximum strike $K_{\text{max}}$.

Your program should then produce two files, `Eprices.csv` (with European option prices) and `Aprices.csv` (with American option prices) using the correct `.csv` format. Each file should contain a table of option prices, together with appropriate row and column headings. Every entry in the $i^{\text{th}}$ row ($i = 0, \dots, 10$) should correspond to an option with strike price

$$K_i = K_{\text{min}} + \frac{i}{10}(K_{\text{max}} - K_{\text{min}}),$$

and every entry in the $j^{\text{th}}$ column ($j = 1, \dots, 5$) should correspond to an option with expiry date $100j$ in the tree model with step size $h_j = \frac{T}{100j}$. A sample file `Eprices.csv` has been provided to illustrate the structure of the output; your output should contain numerical results in place of each of the entries marked “`price`”.

Include two sample files `Eprices.csv` and `Aprices.csv` in your submission, placed in the directory where your program will be producing these files. Do not hard wire any absolute paths into your program, as these paths may not exist on the marker’s computer. Include information in the end-user instructions on how these files can be used.

You should include reports on test runs of your code in a separate section of the developer documentation. This section should include numerical results, comments on the convergence of option prices, as well as graphs produced from your output.

## 1 Submission instructions

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

**Code developer documentation** Presenting the structure of the code, available classes and functions, main variables and any other information to help understand the code. This should include information on how to extend the code by adding new payoffs. Test runs of your code, as well as more detailed explanations of the methods used, should be reported in separate sections of the developer documentation.

**End user instructions** 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.

You may use and adapt all C++ code provided in Moodle as part of this module, including the code developed in support of lectures and the solutions to exercises. You may use and adapt any code submitted by yourself (but not by other students) as part of this module, including your own solutions to the exercises. **Any code not written by yourself must be clearly acknowledged.**

Submit your work by uploading it in Moodle by 23:55 on Thursday 27 April 2017. Submit the code as a single compressed `.zip` file, including all Code::Blocks project (`.cbp`) files, data files (`.csv`), source code and header (`.cpp` and `.h`) files and the executable (`.exe`) file produced by compiling and linking the project, all residing in a single parent directory, whose name should contain your name and student code. 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). It must be possible to open all project files and to compile, link and run the project using the version of Code::Blocks running on computer lab machines.

The documentation files must be submitted in `.pdf` format (two separate `.pdf` files containing developer documentation and user instructions) and uploaded in Moodle as a single `.zip` file separate from the code file. The `.zip` file containing documentation should also bear your name and student number.

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 `alet.roux@york.ac.uk` before the deadline. 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, compiled and run on computer lab machines directly from the directories created by unzipping the submitted `.zip` files. 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 computer lab machines 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 lab computer to that on which the files have been created.

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. All files must be submitted inside the zipped project directory, and all the files in the project directory should be connected to the project.

As part of the marking process, the file `Project.h` will be replaced. If the copy of `Project.h` used in a project has been modified in such a way that the code does not compile with the original file, then a mark of 0 will be recorded. It is therefore advisable to check for inadvertent modification by replacing `Project.h` with a freshly downloaded copy before submission.