MSc in Financial Mathematics, MSc in Financial Computing,
MSci in Financial Mathematics (2022/23)

MTH773P / MTH773U

Advanced Computing in Finance

Class Test 2

Wednesday 12th April 2023

Name:		
Student number:		
Start time:		(Invigilator use only)
Finish time:		(Invigilator use only)

IMPORTANT

Please read the Test Rules and Instructions carefully (pages 2 and 3).

Any suspected violation of these rules or instructions will automatically be referred to the Chair of the Examination Board, and possibly to the University authorities.

The penalties for assessment offences can be severe.

DO NOT REMOVE THE PRINTED COPY OF THIS QUESTION PAPER FROM THE EXAM ROOM.

Test rules

Official Queen Mary assessment regulations will apply.

You must work independently, and must not communicate with anyone else whilst undertaking the test.

Your solutions must be entirely your own work. It is an offence to seek or obtain help from anyone else whilst undertaking the test.

Any similarities between solutions submitted by different students will be thoroughly investigated by the examiners.

The University may also use plagiarism-detection software.

Instructions

- The duration of the test is **2 hours**. However, you will be given a maximum of **4 hours** in which to complete the test and upload your solutions to QMplus (see below).
- After uploading your solutions, you must ask an invigilator to validate your submission and record your finish time before you may leave the room.
- This question paper is available both electronically (on QMplus) and in printed form.
- The printed copy of the question paper will be retained by the School (including any rough working that you do on the blank pages at the end) although nothing on it will be marked.

The test instructions are continued on the next page...

When undertaking the test...

- You are permitted to use the following software only:
  1. **Notepad** or another text editor (for creating `.txt` files).
  2. **Visual Studio** (for entering, building and running C++ programs).
  3. An **internet browser** (but only for viewing the MTH773P QMplus page, and for accessing the static resources on that page, i.e. lecture notes, sample code, coursework question sheets and model solutions).
- You may not access any other files (e.g. your own solutions to coursework questions).
- You may not access the internet.
- You may not use any other resources (e.g. textbooks, your own notes).
- Rough working should be done on the blank pages at the end of the printed copy of this question paper. (This will not be marked.)
- Generally speaking, we expect you to use C++ language features that have been covered in this module (or in your previous module on C++). If you use any other C++ features then you must provide detailed explanatory comments in your code.
- Note that excessively long, highly elaborate and/or over-engineered solutions may be penalised.

When you have finished answering the questions...

- Please upload the following files to QMplus:

Question 1	`main01.cpp`
Question 2	`main02.cpp`
Question 3	`main03.cpp`

  Do not upload any other files such as Visual Studio configuration or build files, object (`.obj`) files or executables (`.exe`).
- Ask an invigilator to validate your submission. As part of the validation process, the invigilator will watch as you email the same set of files to `maths@qmul.ac.uk`. The invigilator will also record your finish time on the question paper.
- You may then leave the room.

Question 1 [20 marks]

(a) Create a new Visual Studio C++ project with a single file, `main01.cpp`. You should implement your entire solution for this question in this single file. On the first line of this file please type a C++ comment with your name and student ID number.

(b) From the file `bisectionSolverDemo2.cpp` (in the code samples for Topic 10 on QMplus), copy-and-paste the class "`Function`" and the function "`BisectionSolver`" into your file `main01.cpp`. You should not make any changes to this code when answering this question.

(c) Add a `main()` function into `main01.cpp`, initially containing no code.

Now add some new code to `main01.cpp` to solve the following two problems numerically. You should...

- Use the solver implemented in the function `BisectionSolver`.
- Determine (and display) each result to an accuracy of 12 decimal places.

(d) Find all the values of x that satisfy the equation

$$\sin x = x^2,$$

explaining your methodology carefully.

(Type your answer as a C++ comment at the end of the file `main01.cpp`.) [8]

(e) For each of $N = 1, 2, 3, \ldots, 10$, find the value of x between 0 and 1 that satisfies

$$\sqrt{2} + 3x \sum_{k=1}^{N} (-1)^k \frac{3^{2k} - 1}{(2k+1)!} x^{2k} = 0.$$

Hence, estimate the value of x that satisfies

$$\sqrt{2} + 3x \sum_{k=1}^{\infty} (-1)^k \frac{3^{2k} - 1}{(2k+1)!} x^{2k} = 0,$$

justifying your answer.

(Type your answer as another C++ comment at the end of the file `main01.cpp`.) [12]

Overview of Questions 2 and 3

In questions 2 and 3, you will use two different numerical methods (Monte Carlo and finite-difference) to price a barrier option in the case where the volatility of the underlying stock is assumed to be time-dependent.

Please begin by reading the following overview which applies to both questions.

Barrier options

A barrier option is similar to an ordinary European call or put option with strike K, except that the payoff at expiry time T (the time from now, $t = 0$) also depends on whether the stock price has crossed (or touched) some pre-agreed barrier B at least once during the lifetime of the option.

In practice, the stock price is not monitored continuously to determine whether the barrier has been crossed. Rather, the stock price will be sampled only at some particular pre-agreed times $t_1, t_2, \ldots, t_m$. For questions 2 and 3, you will be pricing an option where the times t_i are equally-spaced, with $t_i = iT/m$ $(i = 1, 2, \ldots, m)$.

We will consider one particular category of barrier options known as knock-out options. Specifically, for a down-and-out option, the payoff of the option at time T will be zero if, at any of the times t_i, the stock price had been less than (or equal to) the barrier B. Otherwise the payoff will be the same as for an ordinary European option.

Time-dependent volatility model

We will assume that the stock price $S(t)$ evolves (in the risk-neutral measure) according to the stochastic differential equation

$$dS(t) = r\,S(t)\,dt + \sigma(t)\,S(t)\,dW(t)$$

where r is the risk-free interest rate, $W(t)$ is the Wiener process, and $\sigma(t)$ is the time-dependent instantaneous volatility, given by

$$\sigma(t) = \sigma_0 \exp\left[-\beta t\right]$$

The quantities σ_0 and β are parameters of the model, and the time t is given in years (with $t = 0$ being today). Note that the volatility here is deterministic (and so this is not a stochastic volatility model).

You may assume that the underlying stock pays no dividends during the lifetime of the option.

Question 2 [40 marks]

In this question, you will use the Monte Carlo method to price a down-and-out call option, using the time-dependent volatility model specified on the previous page.

(a) Create a new Visual Studio project, with a single file `main02.cpp`. You should implement your entire solution for this question in this single file. On the first line of this file please type a C++ comment with your name and student ID number.

(b) Copy the entire contents of the file `MonteCarloDemo2.cpp` (in the code samples for Topic 12 on QMplus) into your file `main02.cpp`. You will use this code as the starting point for your solution to this question. (This program currently prices an Asian call option.)

(c) The program requires the classes `Stats1` and `NormalRandomGenerator`. So, download the following five files (from the model solutions to Coursework 1 on QMplus), and add them to your Visual Studio project.

`stats.h`	`normalRandomGenerator.h`	`error.h`
`stats.cpp`	`normalRandomGenerator.cpp`	

You should not edit these files, and you do not need to submit them with your solution.

(d) In `main02.cpp`, change the function name from `MonteCarloDemo2()` to `main()`, and check that you can build and run the program.

(e) Now, modify this program so that it can price (at time $t = 0$) a down-and-out call option with $K = 155$, $B = 140$, $m = 12$ and $T = 1$, with the market data $S(0) = 150$ and $r = 0.06$, and using the time-dependent volatility model with model parameters $\sigma_0 = 0.3$ and $\beta = 0.9$.

Implementation instructions

- You may hard-code the various parameter values rather than asking for any user input.
- You should remove (or comment out) any code specific to the Asian call option, and replace it with some corresponding code for the barrier option.
- You will also need to make various other modifications to the program, but do not change any existing code that does not need to be changed.
- Your solution should use object-oriented features where appropriate, but, for simplicity, all new code should be in the file `main02.cpp`.
- **You should add comments to any new code that you write, explaining clearly what it does and how it works.**

Estimate the price $V(0)$ of the option (and its standard error) using $N_{\text{paths}} = 10\,000$ (the number of simulation paths).

(Type your answer as a C++ comment at the end of the file `main02.cpp`.) [25]

(f) Using your previous result, perform a short calculation (not using the computer) to estimate how many paths will be needed to obtain a standard error of 0.01 in the price, explaining your reasoning clearly. Rerun your program with this new value of N_{paths}. What is your new estimate of the price of the option?

(Type your answer as another C++ comment at the end of the file `main02.cpp`.) [5]

(g) Finally, add some code to your program to estimate the gamma Γ and the rho ρ for the option at time $t = 0$. (When determining ρ, you should shift r by 0.001.)

What values does your program give for each of these?

(Type your answer as another C++ comment at the end of the file `main02.cpp`.) [10]

Question 3 [40 marks]

In this question, you will use the finite-difference method (explicit scheme) to price the same down-and-out call option as in question 2, using the same model.

Please read the overview on page 5 if you have not already done so.

(a) Create a new Visual Studio project, with a single file `main03.cpp`. You should implement your entire solution for this question in this single file. On the first line of this file please type a C++ comment with your name and student ID number.

(b) Copy the entire contents of the file `explicitFiniteDifferenceMethodDemo.cpp` (in the code samples for Topic 11 on QMplus) into your file `main03.cpp`. You will use this code as the starting point for your solution to this question. (This program currently prices a European put option.)

(c) Change the function name from `explicitFiniteDifferenceMethodDemo()` to `main()`, and check that you can build and run the program.

(d) Now, modify this program so that it can price (at time $t = 0$) a down-and-out call option with $K = 155$, $B = 140$, $m = 12$ and $T = 1$, with the market data $S(0) = 150$ and $r = 0.06$, and using the time-dependent volatility model with model parameters $\sigma_0 = 0.3$ and $\beta = 0.9$.

Implementation instructions

- You may hard-code the various parameter values rather than asking for any user input.
- You should remove (or comment out) any code specific to the European put option, and replace it with some corresponding code for the barrier option.
- You will also need to make various other modifications to the program, but do not change any existing code that does not need to be changed.
- You are not expected to implement a fully object-oriented solution for this question, and the entire program should be in the file `main03.cpp`.
- **You should add comments to any new code that you write, explaining clearly what it does and how it works.**

What value for the price $V(0)$ does your program give when using gridpoint spacings $\Delta t = 10^{-4}$ and $\Delta S = 1$?

(Type your answer as a C++ comment at the end of the file `main03.cpp`.) [25]

(e) The explicit method becomes unstable if the value of Δt is too high for a given value of ΔS.

For the case where $\Delta S = 1$, determine the highest value of Δt (or, equivalently, the lowest value of the number of time steps $N_t = T/\Delta t$) for which your program gives a reasonably sensible result. You should explain in detail the steps that you took to do this.

(Type your answer as another C++ comment at the end of the file `main03.cpp`.) [5]

(f) Now, rerun your program multiple times, with $\Delta S = 1/N$ and $\Delta t = 10^{-4}/N^2$, for $N = 1, 2, 4, 8, 16$ and 32, to obtain a sequence of (approximate) values for the option price.

We expect that, as N increases, this sequence of values will converge to the exact price. By inspecting the sequence of values, give a conjecture for the exact price, justifying your answer fully.

(Type your answer as another C++ comment at the end of the file `main03.cpp`.) [5]

(g) Finally, discuss carefully whether your results from the finite-difference method in parts (d) and (f) are consistent with that from the Monte Carlo method in question 2(f). If you do not believe that the results are consistent with each other, explain why this might be.

(Type your answer as another C++ comment at the end of the file `main03.cpp`.) [5]

END OF PAPER

Space for rough working (this will not be marked)

DO NOT DETACH THIS PAGE

Space for rough working (this will not be marked)

DO NOT DETACH THIS PAGE