C++ for Quantitative Finance

Term Two 2020

WARWICK BUSINESS SCHOOL

80% Project Assignment

1500 words:
This is a strict limit **not** a guideline: **any piece submitted with more words than the limit will result in the excess not being marked**

Summary and Requirements

Choose **ONE** out of **FOUR** sections: A, B, C or D.

Your project assignment will be a combination of a well-structured and readable student report (in pdf form) and C++ code. For the latter, use the same style as in the mid-term test. Please pay attention to:

- Create a solution called WbsProject and containing **ONE** out of **FOUR** projects A,B,C or D.
- Deliver the project in **Release** mode (and **Subsystem** property set to CONSOLE)
- Implement each question a), b), in modules A and B by dedicated free functions rather than one monolithic and unreadable block of code in `main()`.

15% of the grade is devoted to how well the project and code is structured. In particular, code should be readable and appropriately documented.

All work should have test code. All code should be compilable.

SECTION A: Fundamental Numerical Processing in C++11

Question 1. (C++11 Error Function; Standard Gaussian Probability and Cumulative Distributions)

Use the error function in C++ to write two free/global functions to compute the standard normal probability density function and cumulative distribution function. Call these functions, $n(x)$, $N(x)$, respectively and use them as and when you need them in later exercises.

Create a test program and compute the values of these functions for a range of the input arguments. Check your answers against a reliable source and mention both that source and the test values that you are using. In order words, the functions must give accurate results for all values of the input arguments. Motivate your reasoning on how you set up your *test environment.*

Stress test the program with large positive and negative values of the input argument, for example:

```
#include <climits>
double max = std::numeric_limits<double>::max();
double inf = std::numeric_limits<double>::infinity();
```

Question 2. (One-Factor Option Pricing and Sensitivities)

Use the Black-Scholes formula in combination with the code in Question 1 to compute the price of one-factor European call and put options.

Answer the following questions:

a) Create two free functions that encapsulate the formulae for call and put options. The input arguments are the well-known parameters that are used in the Black-Scholes formula.
b) Create two sets of parameters that will be used as input to the functions in part a). This means that you produce four output values, in other words two call values and two pit values.
c) Check your answers with a known source and *quote that source*. Both sets of answers should be the same. If the answers differ, then the code has errors and you should debug it until it gives the correct answers.
d) Create two free functions that encapsulate the formulae for the delta of call and put options. The input arguments are the well-known parameters that are used in the Black-Scholes formula. Instead of delta, you should implement your code for one of the other Greeks of your choice, such as gamma, rho or theta.
e) Calculate the values of the sensitivities based in part d) and using the data sets in part b).
f) Test your code and make sure that the values are correct.

Question 3. (The C++ <random> Library)

The goal of this exercise is to gain fundamental experience with the *predefined number generators* and *random number generators* in the C++ library `<random>`. It is expected that you will use it in all applications instead of third-party and home-grown code.

Prerequisite: you should have a good understanding of pseudo-random numbers and their applications in computational finance.

Answer the following questions:

a) Give a short introduction to *Mersenne Twister* pseudorandom number generator (PRNG). What are its advantages and disadvantages in computational finance in your opinion and based on your research? Discuss the relative merits of the generators `std::mt19937` and `std::mt19937_64`.

b) Explain what the following piece of code is doing. Use comments directly in the code for the benefit of someone who is seeing `<random>` for the first time:

```
#include <random>
#include <iostream>

int main()
{
      std::random_device rd;
      std::mt19937 gen(rd());
      std::uniform_int_distribution<> dis(1, 6);
      for (int n = 0; n < 10; ++n)
      {
            std::cout << dis(gen) << ' ';
      }
      std::cout << '\n';
}
```

c) We now extend the code in part b) to count the number of occurrences of the value 4 in the above list of numbers. The upper limit of the loop iteration is N (take N = 100, 200, 300, 400) and rerun the code each time. Does the number of occurrences of the value 4 converge to a number? What is that number?

d) We now create a histogram for this problem. To this end, use a `std::map<Key, Value>` (`std::map<int, int>`) where there are 6 integral keys {1,2,3,4,5,6} and the value is the total number of occurrences for each. Write code for this problem. What output do you get? Is it expected?

e) Perform the same procedure as in steps b) to d) for the standard (mean = 0, standard deviation = 1) normal distribution `std::normal_distribution<double>`. What are your conclusions?

f) (*Stress Testing*) Compare the accuracy and run-time performance when comparing `std::mt19937` and `std::mt19937_64` as engines to generate 100 million standard normal variates. Use the `StopWatch<>` class to measure run-time performance.

Question 4. (Random Number Generation, small Research Topic)

The C++ `<random>` library has functionality for major random number distributions, for example:

1. Integral and floating point uniform distribution
2. Integral and floating point normal distribution
3. Chi-squared distribution
4. Student t-distribution
5. Poisson distribution
6. Bernoulli and binomial distributions

Answer the following questions:

a) Give a summary (approximately 150 words) on the applications of these distributions to finance. How much effort do you think is needed to use them in applications?

b) Generate 100,000 Poisson variates and display them as a histogram as we did in exercise Q3 b) for the uniform distribution.

c) Consider the Stochastic Differential Equation for Geometric Brownian Motion:

$$dS = \alpha S dt + bSdW, \text{ where } a, b \text{ constants and } W \text{ Wiener process}$$

The analytical solution is:

$$S(t) = S(0)exp\left(\left(a + \frac{1}{2}b^2\right)t + bW(t)\right), 0 < t \leq T$$

We now divide the interval $(0, T)$ into NT equal subintervals and we approximate the solution at each mesh point by:

$$S(t_{n+1}) = S(t_n)exp\left(\left(a + \frac{1}{2}b^2\right)\Delta t_n + b\Delta W_n\right)$$
$$\Delta W_n = \sqrt{\Delta t}y,\ y \sim N(0,1)$$
$$n = 0, \dots, NT - 1$$
$$S(0) = S_0, \text{ given } \Delta t = T/NT.$$

Implement this solution in C++. Ideally, you should have some way to display the results on the console and as a graph if possible.

Question 5. (The `<tuple>` Library)

A *tuple* (or *n-tuple*) is a fixed-size collection of *heterogeneous* (various data types) elements. They can be used to hold *configuration data* that is needed in applications. It is even possible to *nest* a tuple within another tuple.

The goal of this exercise is to learn the following features:

- Defining a tuple
- Tuple constructors
- `std::make_tuple`
- Accessing and modifying the elements in a tuple using `std::get()`

An example of use to get you up to speed is:

```
using TupleType = std::tuple<std::string, int>;

// Creating tuples
TupleType myTuple(std::string("A"), 1);
TupleType myTuple2 = std::make_tuple("B", 0);


// Accessing the elements of a tuple
std::cout << std::get<0>(myTuple) << std::get<1>(myTuple) << '\n'; // A,1

std::get<0>(myTuple) = std::string("C");
std::get<1>(myTuple) = 3;
std::cout << std::get<0>(myTuple) << std::get<1>(myTuple) << '\n'; // C,3
```

Answer the following questions (in all cases assume that the underlying data type is `double`):

a) Create a tuple that contains two values, for example the up and down jump parameters when constructing the lattice data structure in the binomial method.
b) Explain the performance problems when creating, modifying and accessing tuples in a for loop. For example, a tuple with two elements. Can you think of a better alternative?
c) Create a tuple that models the Black Scholes parameter $\{K, r, div, T, vol\}$. Show how to initialise it as well as accessing its elements.

END OF SECTION A

SECTION B: STL Containers and Algorithms

Question 1. (The <vector> Container)

The objective of this exercise is to become acquainted with vectors in C++ so that you can use them in STL algorithms and in your applications. In particular, study what is said about it on the site www.cppreference.com

Use the instantiated class as follows:

```
using value_type = double;
using Vector = std::vector<value_type>;
```

Answer the following questions:

a) Study the vector class, in particular its constructors, operator overloading [ ] and `const` and `non-const` iterators.
b) Create two free functions to print a vector; the first one uses operator overloading and the second one uses `const` iterators (input argument is `const` call by reference).
c) Write a home-grown free function to multiply the elements of a vector by a double value (this is called *scalar multiplication*). The return type of the function is `void`.
d) Determine how to resize a vector. Examine the values in the vector before and after the resize operator. Describe what you see.

Question 2. (Home-grown Matrices in C++, Part 1)

Standard C++ has no support for matrices but they are needed in many applications. In this exercise we create a simple matrix class as a nested vector (that is, a vector whose elements are also vectors). The definition is:

```
template <typename T>
      using Matrix = std::vector<std::vector<T>>;
```

This can be used as class. It models both *rectangular matrices* (the focus of this exercises) and *jagged matrices* (that we need for the *Binomial method*).

Answer the following questions:

a) Describe in your own words how this class can be used to model both *rectangular matrices* and *jagged matrices*. Think in particular about constructors and accessing the elements of the matrix using operator overloading with []. We create a matrix having n rows and m columns and this should be used in the constructors. You will need to write a free function to initialise the matrix (think hard about this).
b) Create two free functions to *neatly* print a matrix (*row by row*) using the first one uses operator overloading and the second one uses `const` iterators. (input argument is `const` call by reference).
c) Write a free function to add two matrices having the same number of rows and columns (these are input). The output is a newly created matrix having the same shape as the input matrices. Determine what you should do if the input matrices have mutually incompatible structure.
d) Write a function to multiply the elements of a matrix by a `double` value (this is called *scalar multiplication*). The return type of the function is `void`.
e) What are the advantages and limitations of the alias `Matrix` with regards to memory management, robustness and the ability to add new functionality? Would a full-blown class resolve these issues?
f) Explain the different uses of `colon` in the above code.

Question 3. (Home-grown Matrices in C++, Part 2)

Write a C++ class called that models matrices and their operations. The interface is (TBD means "to be done by the student"!):

```
#ifndef NestedMatrix_HPP
#define NestedMatrix_HPP

template <typename T> class Matrix
{ // Simple matrix class
private:
      std::vector<std::vector<T>> mat;

      std::size_t nr;
      std::size_t nc;
public:
      Matrix(std::size_t rows, std::size_t cols) : nr(rows), nc(cols)
      { // TBD }

      NestedMatrix(const NestedMatrix<T>& m2) : nr(m2.nr), nc(m2.nc)
      { // TBD }

      T& operator ()(std::size_t row, std::size_t col)
      { // TBD }

      const T& operator ()(std::size_t row, std::size_t col) const
      { // TBD }

      NestedMatrix operator + (const NestedMatrix<T>& m2) const
      { // TBD }

      NestedMatrix operator - (const NestedMatrix<T>& m2) const
      { // TBD }

      std::size_t size1() const
      { // TBD }

      std::size_t size2() const
      { // TBD }
      };

#endif
```

Answer the following questions:

a) Use C++ syntax and in particular the already discussed checklist questions C1 to C12 to implement this class (use *inline* code).
b) Test the class using the same data as in Question 2.
c) Add any other member functions as you see as being fit for use in applications.
d) Write a function to multiply two *compatible* matrices.

Question 4. (First Encounters with STL Algorithms)

STL support hundreds of algorithms that are applied to containers. We focus on that subset of the functionality that is directly applicable to the current applications in computational finance. A prototypical motivational example is:

```
int size = 5;
std::vector<int> vec(size);
vec[0] = 1; vec[1] = 2; vec[2] = 3; vec[3] = 4; vec[4] = 5;

print(vec, std::string("original vector vec: "));

// Accumulate: Computing the result of one sequence.
int initVal = 1; // Must be initialised to a value; Add initVal to all
elements
int acc1 = std::accumulate(vec.begin(), vec.end(), initVal);
std::cout << "Sum 1: " << acc1 << std::endl;
```

Answer the following questions:

a) Write two functions to compute the *geometric and arithmetic average* of a vector (hint: use `std::accumulate(), std::pow(), static_cast()).`
b) Investigate the algorithms `std::min_max`, `std::inner_product` and `std::count`. Give some examples of each one.

Question 5. (Lambda functions, Function Objects and Type-safe Function Pointers, Research)

Carry out some research into these language features. Useful links are:

```
// 4. Type-safe function pointers
//
// https://en.cppreference.com/w/cpp/utility/functional/function
//
// 5. Function object ==> overloads () operator
//
// https://www.quantstart.com/articles/Function-Objects-Functors-in-C-Part-1/
//
// 6. Lambda expressions
//
// https://en.cppreference.com/w/cpp/language/lambda
```

A simple example of lambda functions that we store in function variables `f1` and `f2` is:

```
#include <functional>

// Define two lambda functions and store them.
std::function<double (double)> f1=[](double v){ return v*v; };
auto f2=[](double v){ return sqrt(v); };
```

Answer the following questions:

a) Explain what function objects and type-safe function pointers are. Give a simple working example of each one.

b) Let's say you wanted to model a one-factor call payoff. How would you implement it using each of these three options?

END OF SECTION B

SECTION C: The Binomial Method

Question 1. (Binomial Method, Preparation)

This is a well-known method and it is assumed that the student is familiar with its financial and mathematical foundations as discussed in standard textbooks. Some exposure to its implementation in a programming language would be advantageous.

In the interest of efficacy, we provide the student with small software framework that you will customise and extend. The files making up the framework contain dedicated *loosely-coupled* functions:

- `OptionDate.hpp` (Black Scholes parameters)
- `Structures.hpp` (creating, accessing and modifying lattice data structures; see also useful links in comments section!)
- `BinomialPricing.hpp` (CRR strategy and function that implements Binomial method)
- `TestBinomial101.cpp` (a test program for call and put option prices)

Answer the following questions:

a) Study the code and relate it to your financial knowledge of the binomial method. Write a small paragraph (approximately 10 sentences) to explain how the different functions work together to compute an option price based on the Black Scholes parameters as input. In particular, describe the flow of data from input to output.

b) The framework uses *lambda functions* and *type-safe function objects*. You should be familiar with these language features (see Question 5 in Section B). Describe in your own words where they are used and what the advantages are. Are there advantages when compared to using free functions and class hierarchies with *pure virtual functions*? In particular, explain the rationale for the following:

```
template <typename T>
      using Lattice = std::vector<std::vector<T>>;

// A function: T -> tuple<T,T> (read from left to right)
template <typename T> // FORWARD
      using LatticeFunctionType = std::function<std::tuple<T, T>(T)>;

// A function: (upper, lower) -> T (read from left to right)
template <typename T>
      using LatticeFunctionTypBack = std::function<T(T upper, T lower)>;

// A function: T -> T (read from left to right)
template <typename T>
      using PayoffFunctionType = std::function<T(T)>;
```

c) Implement the *Jarrow-Rudd* (JR) method and incorporate it into the software framework. See

http://simulations.lpsm.paris/binomial_trees/jr

Test this method and compare the accuracy against the CRR method.

d) We created an in-memory lattice data structure in the software framework. An alternative approach is to compute the lattice values as follows:

$S_j(T) = S_0 u^j d^{n-j}, 1 \leq j \leq n$

u= up parameter
d= down parameter
S_0= stock price where we wish to price the option
n= number of two steps in the binomial method.

or in computational form as:

$$S_j(T) = S_{j-1} u/d, 1 \leq j \leq n$$

Incorporate this algorithm into the framework. Build, run and test your program again using the same data as before.

Question 2. (Binomial Method, Extending the Software Framework)

The goal of this exercise is to extend the framework to support new features. Extra points will be given if the modified code remains readable, maintainable and is properly documented. For each new feature in parts a) to c) below, you need to address the following issues:

a) What effect the new feature on the finance of the problem?
b) In which part of the framework should the new feature be placed?
c) Implement the feature in C++ and test the program again.

We now describe the features:

a) American put options: satisfy the constraint and typical code is:

```
double K = opt.K;
auto PutPayoff = [&K] (double S)-> double
                      {return std::max<double>(K - S, 0.0);};

// American early exercise constraint
auto AmericanPutAdjuster = [&PutPayoff](double& V, double S)->void
{ // e.g. early exercise

     V = std::max<double>(V, PutPayoff(S));
};
```

b) Computing hedge sensitivities; focus on delta:

$$\frac{\partial V}{\partial S} \sim \frac{V_1^1 - V_0^1}{S_1^1 - S_0^1}$$

where in general

V_j^n= Option value at row n and column j.
S_j^n=Stock value of row n and column j.

Compute the computed value and the analytical price for delta.

c) Implement the Binomial method with a continuous dividend yield. Compare against the analytical price.

END OF SECTION C

SECTION D: The Monte Carlo Method

Question1. Monte Carlo Method for One-Factor Option Pricing

We have discussed the Monte Carlo method during the course and we have given minimalist code to price one-factor European options. You should use and extend the algorithmic structure as basis that was provided.

Answer the following questions:

a) Include an estimate of the standard error in all calculations and code.

b) Use *exact simulation* in addition to the Euler method to create paths of the underlying stock. To this end, we follow the steps:

$$dS = (r-d)Sdt + \sigma SdW$$
$$x = \log S$$
$$dx = vdt + \sigma dW,\ v = r - d - \frac{1}{2}\sigma^2$$
$$x_{t+\Delta t} = x_t + v\Delta t + \sigma\Delta W_t\ (\Delta W_t = W_{t+\Delta t} - W_t)$$

then:

$$S_{t+\Delta t} = S_t exp(vt + \sigma\Delta W_t)$$
$$(\Delta W_t = \sqrt{\Delta t}N(0,1))$$

or

$$S_j = exp(x_j),\ x_j = x_{j-1} + v\Delta t + \sqrt{\Delta t}N(0,1),\ j = 1, \dots, N$$

Implement this algorithm and compare it to the Euler method (accuracy and run-time performance).

c) We now wish to price Arithmetic Asian options using the Monte Carlo method. First, write a class (or function) to model fixed strike options:

Put payoff = $\max\left(0, K - \sum_{j=1}^{n} S_j\right)$

Call payoff = $\max\left(0, \sum_{j=1}^{n} S_j - K\right)$

d) Now modify the MC algorithm to price this class of Asian option.

END OF SECTION D

SUBMISSION DEADLINE: 12:00 (noon, UK time) 27 April 2020

Word Count Policy and Formatting (found in your Masters Student Handbook)

https://my.wbs.ac.uk/-/academic/37360/resources/in/381545,786874/item/786880/

The submission deadline is precise and uploading of the document must be completed before 12.00 (UK time) on the submission date. Any document submitted even seconds later than 12.00 precisely will be penalised for late submission in line with WBS policy. Please consult your student handbook on my.wbs for more detailed information.

The online assignment submission system will only accept documents in portable documents format (PDF) files. Please note that we will not accept PDF files of scanned documents. You should create your assignment in your chosen package (for example, Word), then convert it straight to PDF before uploading. Please place your student ID number, NOT YOUR NAME, on the front of your submission as all submissions are marked anonymously.

All the scripts should also have the following paragraph included on the front page:

This is to certify that the work I am submitting is my own. All external references and sources are clearly acknowledged and identified within the contents. I am aware of the University of Warwick regulation concerning plagiarism and collusion.

No substantial part(s) of the work submitted here has also been submitted by me in other assessments for accredited courses of study, and I acknowledge that if this has been done an appropriate reduction in the mark I might otherwise have received will be made.

PLEASE ENSURE YOU KEEP A SECURITY COPY OF YOUR ASSESSMENT

Please ensure that any work submitted by you for assessment has been correctly referenced as WBS expects all students to demonstrate the highest standards of academic integrity at all times and treats all cases of poor academic practice and suspected plagiarism very seriously. You can find information on these matters on my.wbs, in your student handbook and on the University's library web pages here.

The University's Regulation 11 clarifies that '…'cheating' means an attempt to benefit oneself or another by deceit or fraud. This includes reproducing one's own work…' It is important to note that it is not permissible to re-use work which has already been submitted by you for credit either at WBS or at another institution (unless you have been explicitly told that you can do so). This is considered **self-plagiarism** and could result in significant mark reductions.

Upon submission of assignments, students will be asked to agree to one of the following declarations:

Individual work submissions:

I declare that this work is entirely my own in accordance with the University's Regulation 11 and the WBS guidelines on plagiarism and collusion. All external references and sources are clearly acknowledged and identified within the contents. No substantial part(s) of the work submitted here has also been submitted by me in other assessments for accredited courses of study, and I acknowledge that if this has been done it may result in me being reported for self-plagiarism and an appropriate reduction in marks may be made when marking this piece of work.

By agreeing to these declarations (when the message pops up on submission) you are acknowledging that you have understood the rules about plagiarism and self-plagiarism and have taken all possible steps to ensure that your work complies with the requirements of WBS and the University.

You should only indicate your agreement with the relevant statement, once you have satisfied yourself that you have fully understood its implications. If you are in any doubt, you must consult with the Module Organiser or Named Internal Examiner of the relevant module, because, once you have indicated your agreement, it will not be possible to later claim that you were unaware of these requirements in the event that your work is subsequently found to be problematic in respect to suspected plagiarism or self-plagiarism.