LM Advanced Mathematical Finance Module

Part B C++ Spring Term 2021

PROBLEM SHEET 3	SPRING TERM, PROGRAMMING IN C++ LECTURE 3
Lecturer	Dr. Simon Hartley (s.hartley@bham.ac.uk) and Dr. James Allsopp (j.allsopp@bham.ac.uk)
Of	School of Mathematics, College of Engineering and Physical Sciences, University of Birmingham
Due	Friday 29th April 2022, 23:59
Weighting	20% of Part B

INSTRUCTIONS FOR THE SUBMISSION

Submit your code on Canvas as zip:

Submit as a single zip file containing all the code for the problem.

The source files should compile without errors or warnings on Visual Studio 2019 as installed on the clusters, and the resulting executables should run without problems, **Using debug and x64 build options**.

You do not have to submit the executable. You should comment your code and describe what each function does and you should use intelligible naming, the only exception being i, j, and k for any loop counters.

If you use another compiler other than Visual Studio, you should take your final code and compile it on Visual Studio on the remote clusters (https://remoteclusters.bham.ac.uk/maps).

Unfortunately, different compilers follow slightly different rules, and even a small difference may cause the code to fail on Visual Studio.

The source code should also be well formatted according to Google’s C++ guidelines and this will be checked against cpplint as described in the TimeSeriesTransformations.h file. This is an automated part of most jobs in software now and is best practice, so it’s good experience. I’ve not adopted the standard in its entirety and left some parts out, which is what the filter options in the command line control.

For this assignment, I will run your test.cpp file, but I will also overwrite that with my test.cpp file and recompile. I will then run those tests with my own data file. The success or failure of these tests will account for the majority of the marks (2 marks per test). There is no need to complete the TimeSeriesTransformationsApplication.cpp executable to run your library, but this may help you and go towards the discretionary 10% marks. You could, for instance, put a text-based menu system or handle options from command-line. I have attempted to clear up any issues in the TimeSeriesTransformations.h file, but if there’s something unclear, email me (James). The name of each test should give you a guide to what I’m testing.

Other marks come from:

- Compiling without warnings (10%, reduced by 1 for each warning)
- Passing the style check (5%, reduced by 1 for each warning)
- 10% discretionary marks for things the markers believe to be worthy of credit; such as but not limited to, good use of the STL, algorithms library, writing good tests, naming.

The total mark will be capped at 100%.

PLAGIARISM

You are not allowed to plagiarise. This means that you are not allowed to directly copy the code from any online sources or any of your fellow students. Direct group work is also not allowed in this assignment. While you are allowed (and even encouraged) to discuss the contents of this module (including problem sheets) with your fellow students, every one of you needs to write your own code.

PROBLEM 1 Create Time Series Share monitoring class in C++

Problem

In the analysis of time series data, filtering, the task of identifying and removing values, often needs to be carried out to identify both short and long-term trends. For instance, the rolling average is commonly used with time-series data to smooth out short-term fluctuations and highlight longer-term trends or cycles. The first step toward this is to implement this filtering in C++.

In C++ transformations can straightforwardly be applied to the data when it is stored in container data structures. This is a very common task, and tools to do so are provided by the STL, which includes templated methods that execute common tasks, such as sorting and selection. The STL algorithms for doing this can be applied to data stored in the STL containers such as std::vector, std::array, and std::list. We do not expect you in this task to implement any algorithms yourself.

We have provided a zip file that contains three projects and one test file; these are;

- A static library TimeSeriesTransformations, containing TimeSeriesTransformations.cpp and TimeSeriesTransformations.h
- A Google test project called TimeSeriesTransformations-Test, containing test.cpp
- An executable project called TimeSeriesTransformationsApplication, containing the file TimeSeriesTransformationsApplication.cpp
- A file of test data called Problem3_DATA.csv

Passing this assignment requires you to build all the functions referred to in TimeSeriesTransformations.h in TimeSeriesTransformations.cpp and create tests in test.cpp to check they work. Using the TimeSeriesTransformationsApplication.cpp file to create a user interface is entirely optional, but may help you. As long as you don't alter any of the code I have provided in TimeSeriesTransformations.h, any functions or internal class members you want to create are up to you. I would study the comments in the TimeSeriesTransformations.h and the test names in test.cpp to get a better idea of how to solve this. Feel free to delete all the tests in test.cpp and replace them with your own.

The test data file contains time series data, which contains a column with a timestamp stored in UNIX time (see Lecture 3), and a column of data (floating point doubles) representing a share price labeled X like the following:

```
TIMESTAMP,ShareX
1619120010,61.43814038
1619125010,58.74814841
1619130010,90.10017163
1619135010,100.5352112
.....
```

I've left the #includes I used in as a hint, feel free to use or not according to your preference.

The functions described in TimeSeriesTransformations.h provide the following functionality, although there are some additional functions in the header file added to make testing easier;

- a constructor which can load a file given the filename (e.g. "testdata.csv") and store the time data and the share prices internally.
- a constructor which can load the data into the class, where the input is given by two std::vectors representing the time series and the share prices and a name.
- functions mean and standardDeviation which compute the mean and standard deviations of the share prices for the column of data.
- functions computeIncrementMean and computeIncrementStandardDeviation which compute the mean and standard deviation of the increments (the difference between the share price on two consecutive timestamps).
- a function addASharePrice that takes a date and a value, and adds that at the end of the data.
- a function removeEntryAtTime that allows the user to remove an entry by providing the timestamp. Your solution should utilise the remove_if, or partition templates.
- a function removePricesGreaterThan that allows the user to remove all entries above a given value (exclusive).
- a function removePricesLowerThan that allows a user to remove all entries below a given value (exclusive).
- a function printSharePricesOnDate that allow us to print all share prices on a given day, and a function printIncrementsOnDate which allows us to print all increments that occur on a given day.
- a saveData function that will write the data (including any new data and a header) to a filename given by the user.
- a function findGreatestIncrements, which returns the time and amount of the greatest increment. user.
- a function getPriceAtDate, which returns the price at a specific datetime.