John Armstrong

C++ for Financial Mathematics - Solutions

Solutions to Selected Exercises

The C++ code referenced in the solutions can be found at `https://nms.kcl.ac.uk/john.armstrong/cppbook/cpp-website.html`.

Section 1.4

1.4.1. The error message depends on the development environment you are using. Visual Studio gives the helpful message:

```
error C2144: syntax error : 'double' should be preceded by ';'
```

It identifies the line containing the error as the line starting `double principal`. A human would think the error occurred by missing the semi-colon on the line above.

1.4.2. On Visual Studio the error message was:

```
error C2143: syntax error : missing ')' before ';'
```

This is a helpful message although it is not quite correct. The bracket should be before the `*` and not the `;`. Of course, the computer doesn't know what actual formula we are trying to type so it has to guess.

1.4.3. On Visual Studio, this produces a screen full of error messages. The first of these messages is helpful. The other messages are misleading. If the compiler becomes confused it may produce many error messages. This is why the first message is always the most important to fix.

1.4.4. This exercise is designed to show that the computer can sometimes identify entirely the wrong file as containing an error. Inserting the letter `x` at the beginning means the compiler becomes confused before any of the `#include` statements. These statements cause library code written by other people to be compiled as part of the program. Since the compiler is confused, it thinks that these files contain errors. They do not.

1.4.5. Once you type an unexpected character, in this case a dollar sign our code stops working. This is because it assumes that the user will only enter numbers. Writing code that can cope with unexpected user inputs is a skill of its own, but not one we will cover in this course. Usually software development is divided in teams with user interface experts and mathematicians writing different code. We will assume in this book that your job will be writing mathematical code.

1.4.6. In Visual Studio the error is as follows.

```
error C2065: 'principal' : undeclared identifier
```

This error message isn't as helpful as it could be to a novice. It is saying that we haven't yet declared the type of the variable principal. `principal` is the name of a variable, in other words it is an identifier for a variable. We have not declared its type. So it is an `undeclared identifier`.

1.7.1. See main.cpp in ProfitCalculator, line 13.

Code with long names is generally considered better by computer programmers because it doesn't need to be documented as thoroughly. Code with short variable names will need lots of comments.

Section 2.5

2.5.1. See main.cpp in Exercises2, line 88.

2.5.2. See main.cpp in Exercises2, line 99.

2.5.3. See main.cpp in Exercises2, line 113.

2.5.4. See main.cpp in Exercises2, line 126.

2.5.5. See main.cpp in Exercises2, line 140.

2.5.8. It prints out (on a 32 bit compiler) that a is bigger because we are interpreting a as equal to 4294967295. Written in binary this number is 11111111111111111111111111111111. When interpreted as an unsigned integer this is the binary representation of 4294967295 . When interpreted as a signed integer this is the binary representation of -1.

Section 3.9

3.9.1. See main.cpp in Exercises3, line 82.

3.9.3. See chapter3.cpp in FMLibInstructor, line 88.

This is an inefficient method since if the n-th fibonacci number. If c_n is the numnber of times this function is called to compute the n-th fibonacci number we see that $c_n = c_{n-1} + c_{n-2} + 1$ and $c_0 = 1$, $c_1 = 1$. Since the Fibonacci number themselves grow according to the difference equation $f_n = f_{n-1} + f_{n-2}$ and $f_0 = 1$, $f_1 = 1$, we see that $c_n > f_n$. So the number calls required is even greater than the Fibonacci number we're calculating!

3.9.4. See main.cpp in Exercises3, line 91.

3.9.5. See main.cpp in Exercises3, line 90.

3.9.6. See main.cpp in Exercises3, line 143.

3.9.7. See main.cpp in Exercises3, line 176.

3.9.8. See main.cpp in Exercises3, line 201.

3.9.9. You do not need to know anything about these algorithms to use these functions. By dividing code into functions we can divide responsibility across a development team. This allows us to have larger teams and hence write more complex software. No individual has to understand everything.

3.9.10. In our solution we have printed out the values calculated by first computing `norminv` and then applying `normcdf`. This combination of functions should give the identity. See main.cpp in Exercises3, line 242. On the other hand, our test for the Black Scholes call price is pretty weak at the moment. We have used someone elses online calculator to find what the answer should be. The problem is what would we do to test our code if we couldn't cheat and use someone else's answer? We'll consider this question again in the chapter on testing.

Section 4.8

4.8.2. See main.cpp in Exercises4, line 84.

4.8.3. See main.cpp in Exercises4, line 96. This question is of financial interest because the integrand is equal to the cumulative density of the normal distribution up to a factor of $\frac{1}{2\pi}$. This means that we can use this to check whether the `normcdf` function written in earlier chapters is correct.

4.8.5. The solution is included in the answer to the next question.

4.8.6. See main.cpp in Exercises4, line 169.

4.8.7. You're not allowed to look anything up!

Section 5.5

5.5.2. The header file contains less detail and so provides a better overview. This is why comments for users of your library should be put in the header file because that is what they will be reading.

5.5.4. The question really is where do you think that a user of your library will expect to find the definition of the number π. Since most people associate π with basic geometry results, the file `geometry.h` is probably the best choice.

Section 6.6

6.6.4. The interesting part of this question is deciding how to test the pricing formula for the put option. This requires some creativity. Possible ideas include:

- Checking that the put option is nearly worthless if the stock price is far above the strike price
- Checking that the price of the put option is always positive
- Checking that the put option is equal to the integral given in the appendix during the derivation of the Black–Scholes formula.
- Checking the put-call parity formula.

If you are new to financial mathematics, these ideas are probably not very obvious. It is important to develop your understanding of financial mathematics to be able to develop financial software, but that is beyond the scope of this book.

Section 7.7

7.7.2. See matlib.cpp in FMLib9, line 395.

7.7.3. See matlib.cpp in FMLib9, line 399.

7.7.4. See matlib.cpp in FMLib9, line 404. See matlib.cpp in FMLib9, line 408.

7.7.5. See matlib.cpp in FMLib9, line 412.

7.7.6. See matlib.cpp in FMLib9, line 421.

7.7.9. See matlib.cpp in FMLib9, line 438.

7.7.11. The key ingredient is a function called `escapeJavascriptString`. This replaces characters such as quotation marks with 'ánd so forth. Since this is a general purpose routine that is likely to be useful for any web application we have put it in separate file called `textfunctions.cpp`. See textfunctions.cpp in FMLib9, line 89. We use this function whenever we write a string into a web page representing See LineChart.cpp in FMLib9, line 99.

Section 8.6

8.6.1. See PutOption.cpp in FMLib9, line 127.

8.6.2. See LineChart.cpp in FMLib9, line 190.

8.6.3. See geometry.cpp in FMLib9, line 99.

8.6.4. See geometry.h in FMLib9, line 133.

The code will not compile if you omit the `const` keywords in `distanceTo` but leave them in perimeter. Thus if you want to use `const` in `perimeter` you must use `const` in every function that `perimeter` uses.

Section 8.8

8.8.1. See PutOption.cpp in FMLib9, line 80.

Section 9.3

9.3.1. Answering this question simply requires copying and pasting the code for a call option and changing the word call to put throughout. This violates the once only principle. We see how to address this problem in the next chapter.

9.3.2. It should be a log normal distribution with mean $e^{\mu T} S_0$ and standard deviation $\sigma\sqrt{T}$.

9.3.3. See UpAndOutOption.cpp in FMLib12, line 104.

9.3.4. This example is calculated in detail in

It is important to notice that the main complexity that is added is not in the `MonteCarloPricer`, but in the new `UpAndOutOption` class. We can test the modified `MonteCarloPricer` by checking that it still prices ordinary call options correctly. If we know that the `payoff` method of `UpAndOutOption` is correct we can be reasonably confident of our code.

See UpAndOutOption.cpp in FMLib12, line 104.

Some sensible additional checks on the final answer would be to see what happens as the barrier is increased or decreased. For very large barriers, the price should be approximately the same as an ordinary call option. For low barriers the price should be approximately the same as an ordinary put option.

Section 10.6

10.6.1. See DigitalCallOption.cpp in FMLib13, line 97. See DigitalPutOption.cpp in FMLib13, line 97. Note that this implementation actually uses the extension technique discussed in a later chapter.

10.6.2. See matlib.cpp in FMLib13, line 554.

10.6.3. See matlib.cpp in FMLib13, line 390.

10.6.4. See matlib.cpp in FMLib13, line 560.

10.6.8. See RectangleRulePricer.cpp in FMLib13, line 134.

10.6.9. See FMLib13 for a complete solution.

Section 11.8

11.8.2. See pointersolutions.cpp in FMLib12, line 94.

11.8.4. See pointersolutions.cpp in FMLib12, line 109.

11.8.7. See pointersolutions.cpp in FMLib12, line 149.

Section 12.9

12.9.1. See DigitalCallOption.cpp in FMLib13, line 97. See DigitalPutOption.cpp in FMLib13, line 97. Note that this implementation actually uses the extension technique discussed in a later chapter.

12.9.2. The Asian option should extend `ContinuousTimeOptionBase`. See AsianOption.cpp in FMLib14, line 120.

Section 13.5

13.5.1. See Portfolio.cpp in FMLib14, line 195.

13.5.2. See Portfolio.cpp in FMLib14, line 217.

Our Monte Carlo method doesn't evaluate the portfolio consisting of one Up and Out option, one Up And In option and minus a Call Option as being exactly worthless even though we know the payoff must be zero. This is because we're using different random numbers to price each component of the portfolio. This is an area where our code could be improved.

Section 14.4

14.4.1. See HedgingSimulator.cpp in Exercises14, line 212.

14.4.3. See HedgingSimulator.cpp in Exercises14, line 230.

14.4.4. See HedgingSimulator.cpp in Exercises14, line 190.

`ContinuousTimeOption` has been given a `delta` method. By default this uses the new delta function on `MonteCarloPricer`. `PutOption` has its own custom delta function. The `toHedge` parameter of `HedgingSimulator` is a pointer to a `ContinuousTimeOption`.

14.4.5. See HedgingSimulator.cpp in Exercises14, line 248.

A `StockPriceModel` class has been introduced. The simulation model of Hedging Simulator has been changed to use this instead. In addition `HedgingSimulator` now has a `riskFreeRate` member variable as this should be configured separately from the stock price model. In practice one would configure an interest rate model that would allow e.g. stochastic interest rates

14.4.6. See HedgingSimulator.cpp in Exercises14, line 284.

A `Strategy` class has been introduced together with three subclasses. The interpretation of the charts is that if you think the drift is very high and the volatility is very low then, unless you are very risk averse, you will probably think that investing in stock is a more effective investment strategy than delta hedging. The philosophical difference is that investing in stock is a risky strategy, whereas delta hedging is about providing a service to customers in exchange for a commission.

Section 15.4

Section 16.7

16.7.4. See Matrix.cpp in FMLib20, line 660.

16.7.5. See matlib.cpp in FMLib20, line 600.

Section 17.3

17.3.1. The compiler I used says that the error is in `montecarlopricer.h`. We know the error is in `CallOption.h`. It is very confusing for novice users to be given inaccurate information as to which file contains the error. It is not particularly helpful for experienced users either.

Section 18.12

18.12.2. See MargrabeOption.cpp in FMLib20, line 103.

18.12.3. See Portfolio.cpp in FMLib20, line 104.

Section 19.4

19.4.1. See matlib.cpp in FMLib20, line 768.

19.4.2. See RectangleRulePricer.cpp in FMLib20, line 134.

Section 20.4

20.4.1. See Executor.cpp in Exercises20, line 279.

20.4.2. See chapter20.cpp in Exercises20, line 119.

20.4.3. See Pipeline.h in Exercises20, line 13.
Note that the definitions are in the header as well as the declaration.

20.4.4. See chapter20.cpp in Exercises20, line 185.

20.4.5. See chapter20.cpp in Exercises20, line 185.