C++ in Quantitative Finance

Labs #2

Inheritance and virtual functions

Paweł Sakowski

Spring 2026

University of Warsaw
Faculty of Economic Sciences

Virtual functions

- The main method in our MC project, `operator()` is an example of virtual function.
- A virtual function is a function whose address is bound at *runtime* instead of at *compile time*.
- In the code, where a `PayOff` object has been specified, the code when running will encounter an object of a class that has been inherited from `PayOff`.
- It will then decide what function to call on the basis of what type that object is.
- If the object is of type `PayOffCall`, it calls the `operator()` method of `PayOffCall`, and if it is of type `PayOffPut`, it uses the method from the `PayOffPut` class and so on.

Pure virtual functions

- As well as being a virtual function, the `operator()` method has an =0 after it.
- This means that it is a pure virtual function.
- A pure virtual function has the property that it need not be defined in the base class and must be defined in an inherited class.
- Thus by putting =0 we are saying that the class is incomplete, and that certain aspects of its interface must be programmed in an inherited class.

Passing the inherited objects by reference

- In our Monte Carlo routine, we have a parameter of type `const PayOff&` called `thePayOff`.
- This means that the parameter is being passed by reference rather than by value.
- The routine therefore works off the original object passed in.
- If we had not used the '&' it would copy the object: it would be passed by value not by reference.
- When the object is passed by reference, the function is passed the address of the object in memory, no copying occurs and the object's state is precisely as it was outside.
- However, if the object's state does change inside the routine it will also change outside which may not be what we want.
- We therefore include the `const` to indicate that the routine cannot do anything which may change the state of the object.
- The function can 'look but not touch'.

Key concepts for today

- Inheritance can be used to implement a PayOff class that is closed for modification but open for extension.
- Inheritance expresses 'is a' relationships.
- A virtual function is bound at *runtime* instead of at *compile time*.
- We cannot have objects from classes with pure virtual functions.
- We have to pass inherited class objects by reference if we do not wish to change the virtual functions.
- Virtual functions are implemented via a table of function pointers.
- If a class has a pure virtual functions then it should have a virtual destructor.

Exercises I

1. Write additional subclass called PayOffDigital inheriting after PayOff class so that it can handle digital options.
2. Write additional subclass called PayOffDoubleDigital inheriting after PayOff class so that it can handle double digital options.
3. Write additional subclass called PayOffPower1 inheriting after PayOff class so that it can handle squared power options with the payout $\max^2[S_T - K, 0]$ for the call and $\max^2[K - S_T, 0]$ for the put.
4. Write additional subclass called PayOffPower2 inheriting after PayOff class so that it handle squared power options with the payout $\max[S_T^2 - K^2, 0]$ for the call and $\max[K^2 - S_T^2, 0]$ for the put.

Exercises II

Additional exercises based on `structures`

5. a) Create a structure to hold information about an option:
   - strike
   - spot
   - risk-free rate
   - volatility
   - expiry
   - type (call or put)

   b) Create and instance of the proposed structure.

   c) Assign specific values to the members of the structure.

   d) Define and implement a function that will accept the structure as argument and print out all characteristics of the option.

   e) Create a pointer to the structure. What is its address?

   f) Rewrite function from d) so that it accepts the structure (ie. the object) passed: 1) by reference using the pointer, and 2) by reference using the structure (ie. the object).

Exercises III

6. Define and implement a function that will take as an argument the structure above and return theoretical price of European call/put option using the Black-Scholes formula. The structure with option description should be passed:
   1) by value,
   2) by reference using the pointer,
   3) by reference using the object.

   You can use following code for the Black-Scholes function:
   `https://gist.github.com/pawelsakowski/9502742`
   For this you will need:
   `https://gist.github.com/pawelsakowski/9502821`

Exercises IV

7. As in previous exercise but this time function returns aproximation of the theoretical price for European call and put obtained by the Monte Carlo experiment. Does the approximation converge to the BS price? You may use approach we took in previous semester (takes more time, as we generate whole paths for underlying prices) or in cpp01.

Thank you!

Paweł Sakowski
sakowski@wne.uw.edu.pl