FE545 Design, Patterns and Derivatives Pricing

Encapsulation, Inheritance, and Virtual Function

Steve Yang

Stevens Institute of Technology

steve.yang@stevens.edu

01/30/2024

Overview

A Simple Monte Carlo Call Option Pricier in C++

- **European options** define the timeframe when holders of an options contract may exercise their contract rights.
- The rights for the option holder include buying the underlying asset or selling the underlying asset at the specified contract price - the strike price.
- With European options, the holder may only exercise their rights on the day of expiration. Suppose the underlying asset price S_t at time t, and strike price K and expiration T.
- And then the call and put payoffs (at expiration time $t = T$) will be:

$$C_T(S_T, K) = max(S_T - K, 0), \text{ and } P_T(S_T, K) = max(K - S_T, 0)$$

A Simple Monte Carlo Call Option Pricier in C++

- **A double digital option** is a particular variety of option. At maturity, the payoff is 1 if the spot price of the underlying asset is between two numbers, the lower and upper strikes of the option; otherwise, it is 0.
- A double digital with lower strike K_1 and upper strike K_2 can be replicated by going long a digital option with strike K_1 and short another digital option with strike K_2.

$$D_T = \begin{cases} 0 & \text{for} \quad S_T \geq K_2 \\ 1 & \text{for} \quad K_1 < S_T < K_2 \\ 0 & \text{for} \quad S_T \leq K_1 \end{cases} \tag{1}$$

A Simple Monte Carlo Call Option Pricier in C++

- A Simple Implementation of a Monte Carlo Call Option Pricier
- Critiquing the Simple Monte Carlo Routine
  - **Reusability** is the use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation.
  - **Agility** is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.

Demand from an 'evil' boss?

- What might the evil boss demand?
  - "Do puts as well as calls"
  - "I can't see how accurate the price is, put in the standard error."
  - "The convergence is too slow, put in antithetic sampling."
  - "I want the most accurate price possible by 9am tomorrow so set it running for 14 hours."
  - "It's crucial that the standard error is less than 0.00001, so run it until that is achieved."
  - "I read about low-discrepancy numbers on the weekend. Just plug them in and see how good they are."
  - "Apparently, standard error is a poor measure of error for low-discrepancy simulations. Put in a convergence table instead."
  - "We need a digital call pricer now!"
  - "What about geometric average Asian calls?"
  - "How about arithmetic average Asian puts?"
  - "Put in a double digit geometric Asian option."

- Identifying Objects and Classes
    - When you approach a programming problem in an object-oriented language, you no longer ask how the problem will be divided into functions, but how it will be divided into objects. The match between programming objects and real-world objects is the happy result of combining data and functions: The resulting objects offer a revolution in program design.
    - A **class** serves as a plan, or blueprint. It specifies what data and what functions will be included in objects of that class. Defining the class doesn't create any objects, just as the mere existence of data type int doesn't create any variables.
- What Will Classes Buy Us?
    - Classes encapsulate natural financial concepts.
    - Our code becomes clearer - easy to read.
    - Separate interface from implementation.

Implementing the Pay-off Class

- Implementing the Pay-off Class
  - PayOff1.h
  - PayOff.cpp
- Privacy
  - **private** means that the data cannot be accessed by code outside the class. The reason is that as soon as we let the user access the data directly, it is much harder for us to change how the class works.
  - By making the data private, we can enforce the contract between coder and user in such a way that the contract does not say anything about the interior workings of the class.
- Using the Pay-off Class
  - SimpleMonteCarlo2.h
  - SimpleMonteCarlo2.cpp

The Open-Closed Principle

- The ‘open’ refers to the idea that code should always be open for extension.
- The ‘closed’ means that the file is ‘closed for modification’. This refers to the idea that we should be able to do the extension without modifying any existing code, and we should be able to do so without even changing anything in any of the existing file.
- How can one possibly make new forms of pay-offs without changing the pay-off class? Let’s consider how we may accomplish this using C style techniques.
  - We may use a function pointer. Thus we replace

    `OptionType`
  - What do we do when the boss comes by and demands a double-digit pay-off? A double digital option is a particular variety of option (a financial derivative). At maturity, the payoff is 1 if the spot price of the underlying asset is between two numbers, the lower and upper strikes of the option; otherwise, it is 0.

Inheritance and Virtual Functions

- The mechanism for expressing the 'is a' relationship in C++ is inheritance. We shall refer to the class we inherit from as the base class and the class which does the inheriting will be called the inherited class.
- The base class will always express a more general concept than the inherited class. The key point is that each inherited class refines the class above it in the hierarchy to something more specific.
- Coding Inheritance
  - class PayOffCall : public PayOff
    - PayOffCall inherits all the data members and methods of PayOff. And most importantly, the compiler will accept a PayOffCall object wherever it expects a PayOff object.
    - The rules of public inheritance say that we can access protected data and methods of the base class inside the methods of the inherited class but we cannot access the private data.

Inheritance and Virtual Functions

- What is a virtual function? In technical terms, it is a function whose address is bound at runtime instead of compile time.
- Virtual functions are really a fancy way of using function pointers. If you run a program involving virtual function pointers through a debugger, and examine through the watch window an object from a class with virtual functions, you will find an extra data member, the virtual function table.
- If virtual functions are just function tables, why bother with them?
  - They are syntactically a lot simpler. The structure of our program is much cleaner. If we can say this is a pay-off call object and this is a pay-off put object rather than to say that we have function table.
  - We gain extra functionality. With the inherited class, we can add more data members.
- A pure virtual function is a virtual function that needs not to be defined in the base class and must be defined in an inherited class.

Inheritance and Virtual Functions

- Virtual Function
  - PayOff2.h
  - PayOff2.cpp
  - SimpleMCMain3.cpp -

PayOff2.h

```
#ifndef PAYOFF2_H
#define PAYOFF2_h

Class PayOff
{
public:
      PayOff(){};
      virtual double operator() (double Spot) const=0;
      virtual ~PayOff(){};
privite:
}
```

Inheritance and Virtual Functions

SimpleMCMain2.cpp

```
#include "SimpleMC2.h"
#include "PayOff2.h"
#include "Random1.h"
#include <cmath>

//the basic math functions should be in
#if !defined(_MSC_VER)
using namespace std;
#endif

double SimpleMonteCarlo2(const PayOff& thePayOff, // use PayOff class, actually
                         double Expiry,
                         double Spot,
                         double Vol,
                         double r,
                         unsigned long NumberOfPath)
{
    double variance = Vol*Vol*Expiry;
    double rootVariacne = sqrt(variance);
    double itoCorrection = -0.5*variance;
    ...
```

Breakout Exercise: Use Inheritance

- Implement a *PayOffParameters* class which inherit from a base class *BasePayOffParameters*. The goal is to redesign the *PayOff2.h* to make the PayOff class "open" so that it can be extended to other option types.
- Please download a set of files from Canvas named *lect02_breakout.zip*.
  - Use the $c++$ project named *lect02_breakout* as the starting point for your exercise, and complete the implementation.
  - You should get the call price of 7.7866 and put price of 5.20969 for a European option.

Inheritance and Virtual Destruction

- Why we must pass the inherited object by reference?
  - The caller works off the original object passed in.
  - If we had not used & it would copy the object: it would be passed by value not by reference.
  - We include the *const* to indicate that the routine cannot do anything which may change the state of the object.
- Not knowing the type and virtual destruction
  - SimpleMCMain4.cpp
  - Two ways to forget the type of the object:
    ① Via a reference to the base type.
    ② Via a pointer.

  -
    ```
    if (optionType==0)
             PayOffCall thePayOff(Strike)
    else
             PayOffPut thePayOff(Strike)
    ```
  - Virtual destructor: if it is not virtual for the derived class, the base destructor will be called. The object may not be destroyed properly. As pure virtual class, there is no base object created.

Inheritance and Virtual Functions

SimpleMCMain4.cpp

```
#include "SimpleMC2.h"
#include "PayOff2.h"
#include "Random1.h"
#include <cmath>

//the basic math functions should be in
#if !defined(_MSC_VER)
using namespace std;
#endif

double SimpleMonteCarlo2(const PayOff& thePayOff, // use PayOff class, actually
                         double Expiry,
                         double Spot,
                         double Vol,
                         double r,
                         unsigned long NumberOfPath)
{
    double variance = Vol*Vol*Expiry;
    double rootVariacne = sqrt(variance);
    double itoCorrection = -0.5*variance;
    ...
```

Adding Extra Pay-offs without Changing Files

- Adding extra pay-offs without changing files
- Suppose the pay-off we wish to add is the double digit pay-off. This pay-off pays 1 if spot is between two values and 0 otherwise.
  - DoubleDigital.h
  - DoubleDigital.cpp
  - SimpleMCMain5.cpp

Inheritance and Virtual Functions

SimpleMCMain5.cpp

```
#include"SimpleMC2.h"
#include "DoubleDigital.h"
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double Expiry;
    double Strike;
    double Spot;
    double Vol;
    double r;
    double Low, Up;
    unsigned long NumberOfPath;

    ...

    PayOffDoubleDigital doubleDigitalPayoff(Low, Up);

    double resultDouble = SimpleMonteCarlo2(doubleDigitalPayoff,
                                            Expiry,
```

Key Points

- Inheritance expresses ‘is a’ relationship.
- A virtual function is bound at run time 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 function then it should have a virtual destructor.