Quantitative Finance: An Object–Oriented Approach in C++

Erik Schlögl

For Jesu

Contents

Preface

In the forty years since the seminal article by Black and Scholes (1973), quantitative methods have become indispensable in the assessment, pricing and hedging of financial risk. This is most evident in the techniques used to price derivative financial instruments, but permeates all areas of finance. In fact, the option pricing paradigm itself is being increasingly applied in situations that go beyond the traditional calls and puts. In addition to more complex derivatives and structured financial products, which incorporate several sources of risk, option pricing techniques are employed in situations ranging from credit risk assessment to the valuation of real (e.g. plant) investment alternatives.

As quantitative finance has become more sophisticated, it has also become more computationally intensive. For most of the techniques to be practically useful, efficient computer implementation is required. The models, especially those incorporating several sources of risk, have also become more complex. Nevertheless, they often exhibit a surprising amount of modularity and commonality in the underlying method and approach. Ideally, one would want to capitalise on this when implementing the models.

C++ is the de facto industry standard in quantitative finance, probably for both of these reasons. Especially for models implemented "in–house" at major financial institutions, computationally intensive algorithms are typically coded in C++ and linked into a spreadsheet package serving as a front–end. The object–oriented and generic programming features of C++, when used properly, permit a high degree of code reusability across different models, and the possibility to encapsulate algorithms and data under a well–defined interface makes the maintenance of implemented models far easier.

Object–oriented Programming (OOP) is a loaded concept, and much has been written about its advantages and disadvantages, accompanied by scholarly efforts to delineate boundaries between this and other programming paradigms, such as Generic Programming. This book is not an ideological manifesto. If there is any dogma at all in the approach here, it might be the pursuit of "implement once," meaning that any particular functionality in a piece of software should be implemented in one place in the code only[1] — everything else is approached pragmatically, following the motto "horses for courses." While there are certainly many traditional "objects" in the code presented in this book, for example the C++ abstract base class `TermStructure`

[1] This often means that parts of the software have to be rewritten more than once during the development process.

in Chapter 2, C++ templates (a language element more of Generic Programming than OOP) are used extensively as well, especially where a more purist object–oriented solution would entail a performance penalty. Thus this book is about a practical approach to working solutions, and quite a few of those solutions are supplied in form of the C++ code accompanying this book. If there is a manifesto which influenced the code development, it is Gamma, Helm, Johnson and Vlissides (1995) — titled "Design Patterns: Elements of Reusable Object–Oriented Software," but whose design patterns in practice more often than not are implemented as templates.

The aim of this book is to provide a foundation in the key methods and models of quantitative finance, from the perspective of their implementation in C++. As such, it may be read in either of two ways.

On the one hand, it is a textbook, which introduces computational finance in a pragmatic manner, with a focus on practical implementation. Along with the fully functional C++ code, including additional C++ source files and further examples, the companion website[2] to this book includes a suite of practical exercises for each chapter, covering a range of levels of difficulty and problem complexity. While the code is presented to communicate concepts, not as a finished software product, it nevertheless compiles, runs, and deals with full, rather than "toy," problems.

The approach based on C++ classes and templates highlights the basic principles common to various methods and models, and the actual algorithmic implementation guides the student to a more thorough understanding. By moving beyond a purely theoretical treatment to the actual implementation of the models, using the de facto industry standard programming language, the student also greatly enhances her career opportunities in the field.

On the other hand, the book can also serve as a reference for those wishing to implement models in a trading or research environment. In this function, it provides recipes and extensible code building blocks for some of the most common methods in risk management and option pricing.

The book is divided into eight chapters. **Chapter 1** introduces the requisite elements of the C++ programming language. The aim is to keep the book as self–contained as possible, though readers without prior exposure to C++ may also wish to consult one of the many excellent textbooks on this topic.

Chapter 2 provides some basic building blocks, which will be used in the implementation of financial models in the subsequent chapters. The more generic (i.e. non-financial) building blocks are drawn to a large part from excellent open source projects, specifically the Boost libraries and for numerical arrays, Blitz++. Numerical integration, optimisation and root search are also discussed (and implemented) to the extent needed in the subsequent chapters. The remainder of this chapter deals with representing and manipulating term structures of interest rates. Term structure fitting, interpolation and the pricing of basic fixed income instruments is discussed at this opportunity.

[2] This is found at `http://www.schlogl.com/QF`

Almost all fundamental results of option pricing can be illustrated in lattice models and they are also effective numerical methods in their own right. This is the topic of **Chapter 3**. Two broad classes of lattice models are considered in a unified framework: binomial models for a single underlying and models for the term structure of interest rates.

Moving on to a continuous time framework, **Chapter 4** introduces the Black/Scholes model, which for a large range of applications still remains the dominant paradigm. The key assumption is that the underlying asset(s) exhibit deterministic proportional volatility. Whenever this assumption holds, Black/Scholes–type derivative pricing formulae typically follow. These pricing formulae are specific to each option payoff, but have strong commonalities, which can be exploited to price a large class of payoffs in a single ("quintessential") formula. The abstraction from the specific functional form of the deterministic proportional volatility is an important object–oriented feature.

As demonstrated in Chapter 4, the arbitrage–free prices of derivative financial instruments can be expressed as solutions to partial differential equations. In cases where these cannot be solved analytically, finite difference schemes provide one set of alternative numerical methods. **Chapter 5** presents a selection of such methods under a common interface.

Chapter 6 pays heed to the fact that many option contracts are now being liquidly traded in the market, and as such have become bearers of information, to which models need to be calibrated. In the extreme, given (market–quoted) option prices for a continuum of strikes, one can extract an implied risk–neutral distribution for the future price of the underlying asset. In practice, the construction of such implied distributions is linked to the arbitrage–free interpolation of implied volatilities. The fact that implied volatilities vary depending on the moneyness of the option invalidates the Black/Scholes assumption of deterministic proportional volatility, so stochastic volatility is considered as an alternative.

Monte Carlo simulation methods are presented in **Chapter 7**. In many cases, MC simulation is the easiest numerical method to implement, and for high–dimensional problems it is often the most efficient. This efficiency can be substantially improved by various variance reduction techniques. We will also consider cases where valuation by simulation is not straightforward, i.e. for options with the possibility of early exercise. The chapter closes with a discussion of quasi–random methods as an alternative to pseudo–random Monte Carlo.

As a capstone, **Chapter 8** covers the implementation of a tractable multifactor continuous–time model of the term structure of interest rates, the Gauss/Markov case of Heath, Jarrow and Morton (1992). It draws extensively on the building blocks introduced in previous chapters, e.g. the term structure of interest rates in Chapter 2, the deterministic proportional volatility analytical solutions from Chapter 4, and the Monte Carlo simulation engine of Chapter 7.

In the **appendix**, the reader will find useful supplementary information. Appendix A shows how to employ Microsoft Excel as a front end for the models implemented in C++, relieving the implementer of the tedious task of programming a graphical user interface. Generating HTML documentation from appropriately commented C++ source code using a freely available software tool is illustrated in Appendix B.

Quantitative Finance has become a very broad field, and the models and methods covered in this book are only an introductory subset. Beyond this subset there are key models and methods which are undoubtedly of very high practical importance as well. To mention just a few, this includes more sophisticated finite difference methods, local volatility models, upper bound Monte Carlo estimators for early exercise premia, and the LIBOR Market Model. Nevertheless, as in particular Chapter 8 demonstrates, the approach based on C++ classes and templates taken in this book allows reuse of what was introduced in a simpler context, for the solution of similar problems in a more complex context.