1 Executive Overview of this Book

1.1 WHAT IS THIS BOOK?

The goal of this book is to model financial instruments, such as options, bonds and interest-rate products by partial differential equations, finite differences and C++. It is intended for IT and quantitative finance professionals who know this material and wish to deepen their knowledge and for those readers who use techniques such as Monte Carlo, Fourier transform methods, stochastic differential equations and lattice methods (for example, the binomial method) for instrument pricing.

We integrate a number of well-known areas to create a traceable and maintainable path from when a financial engineer proposes a new model to when he or she codes the resulting equations in C++. When viewed as a black box, the core process in this book is to produce C++ classes and code for financial engineering applications. Furthermore, we give lots of examples of code that developers can use without much hassle. The accompanying CD contains all the source code in this book. We provide guidelines, algorithms and reusable code to help the reader to achieve these ends. The main activities that realise the core process are:

- Activity 1: Map the financial model to a partial differential equation (PDE)
- Activity 2: Approximate the PDE by the finite difference method (FDM)
- Activity 3: Implement the FDM using C++ and design patterns.

In this book we shall concentrate on Activities 2 and 3. Since this is a book on the application of C++ to financial engineering we concentrate on mapping the numerical algorithms from Activity 2 to robust and flexible C++ code and classes. However, we shall provide sufficient motivation and background information to help the reader to understand the complete 'instrument life cycle'. This life cycle describes the processes, activities, decisions and alternatives that describe how to program models for financial instruments in C++.

The topics in this book relate to finance, partial differential equations, numerical schemes and C++ code, and for this reason we use the term *Computational Finance* to sum up these related activities (see Seydel, 2003, where the same phrase is used). The foundations for partial differential equations and finite difference schemes for financial engineering applications are discussed in Duffy (2004b).

1.2 WHAT'S SPECIAL ABOUT THIS BOOK?

This book is part of a larger, ongoing project. It is the outcome of one part of this project and concentrates on showing how to program finite difference schemes in C++. Our approach is novel in a number of respects.

1. We use modern *object-oriented* and *generic* design patterns in C++ to solve a range of partial, stochastic and ordinary differential equations in financial engineering. Traditionally, engineers have used packages such as Matlab, Maple, the C language or

other specialised libraries. Each alternative solution has its own benefits of course, but using C++ means that your code is portable, flexible and future-proof (C++ will still be used 20 years from now). Using C++ means that you are not tied into one vendor or operating system.

2. We give a thorough introduction to finite difference methods, how to apply them to Black–Scholes type equations and how to map them to C++ code. We avoid glib recipe-type schemes that work well for toy problems but do not always scale to real-life problems. In particular, we show how to program the famous Crank–Nicolson scheme and discuss when it breaks down, especially in applications with small volatility, discontinuous payoff functions or non-linearities. We propose new schemes that overcome these problems and produce uniformly good approximations to the delta of an option. The book discusses finite difference schemes for both one-factor and two-factor problems.
3. Successful software always needs to be adapted and extended, and to this end we design our classes and applications so that they can easily be modified. Our book is novel in the sense that we apply the powerful and proven design patterns (see Gamma *et al*., 1995; Buschmann *et al*., 1996) to help us to produce applications that can be extended or adapted to different organisational, hardware and software contexts.
4. Last, but not least, it is vital that our software artefacts are well documented. We document these artefacts at the design level and, in particular, we use the *de-facto* Unified Modeling Language (UML) to visually display the structural, functional and behavioural relationships between the classes and objects in our applications.

In short, this book describes in a step-by-step manner how to create 'good' software for financial engineering applications; it also integrates established techniques from fluid mechanics, numerical analysis and software design to produce a coherent and seamless approach to the design and implementation of financial models in C++.

1.3 WHO IS THIS BOOK FOR?

This book is meant for IT and quantitative finance professionals (risk managers, product development and derivatives research groups) who work in financial institutions and software companies and are involved in designing and implementing pricing models in C++. This book deals with fundamental issues such as C++ design and implementation, design patterns, finite difference methods and advanced software environments. Thus, it is of value to financial engineers ('Quants'), software developers and financial modellers.

We feel that the book is useful for universities and other educational institutes that deliver financial courses. This is not a book on instrument theory as such, and we assume that the reader has knowledge of option theory as presented in books by Hull (2000) and Wilmott (1998), for example. We also assume that the reader has had some exposure to differential equations, differential and integral calculus and matrix algebra. Finally, the reader should have a working knowledge of C++.

As we have already mentioned in this chapter, the book is suited not only to those readers from a partial differential equation (PDE) background but also to those who use techniques such as Monte Carlo, Fourier transform methods, stochastic differential equations (SDEs) and the binomial method for instrument pricing. We do our best to show that finite differences compare well with, and even outperform, these former methods, especially for complex and non-linear one-factor and two-factor Black–Scholes models.

Finally, the real option theory is emerging and many of the techniques in this book can be used in decision support systems in the oil, gas and energy industries. Thus, the book is also of interest to engineers, scientists and financial engineers in these fields.

1.4 SOFTWARE REQUIREMENTS

We have written this book from a number of viewpoints that have to do with what we call *software quality*. In general, we adopt the ISO 9126 quality characteristics (see Kitchenham and Pfleeger, 1996) as our working model. ISO 9126 describes how good a software product is. It consists of six top-level characteristics:

- *Functionality*: The ability of the software to satisfy stated or implied customer needs.
- *Reliability*: Does the software maintain its level of performance for a stated period of time?
- *Usability*: Is the software easy to understand, learn or integrate with other applications?
- *Efficiency*: Describes the response times in the application and the corresponding resources that are needed.
- *Maintainability*: How easy is it to modify, adapt and test the application? How stable is the application under change?
- *Portability*: The ease with which the application can be adapted to work in some new software or hardware environment.

Any one (or all) of the above requirements may be important for *your* new or existing software project. In general, the more requirements your applications must satisfy the more time it will take to satisfy them. In this book we classify applications into three broad categories, depending on the level of flexibility that they must have:

- *Low flexibility*: These are either throwaway prototypes or simple programs in order to test a piece of code or check the validity of some new model
- *Medium flexibility*: The code and classes in this category can be customised (by changing its source code if necessary) and used in your own applications
- *High flexibility*: The code in this category can be used in your applications without any changes.

It is important to know at the outset how flexible our solutions must be; on the one hand, we do not want to 'over-engineer' our application, but nor do we want to produce code that is difficult to maintain, understand or falls apart when we modify it. This book will provide you with guidelines to help you to produce good designs for financial engineering applications.

We *layer* the software in this book by examining it at four different levels:

- *Foundation classes and building blocks*: Reusable components for vectors, lists, matrices and other containers. We make ample use of the Standard Template Library (STL).
- *Mechanisms*: Tightly coupled groups of generic functions that are related to a specific piece of functionality. An example is a set of functions for Date manipulations (cash flows, interest rate curves).
- *Half-products*: Ready-to-use libraries that you can use as part of your own applications. We can place these half-products in assemblies and DLLs.

- *Applications*: Dedicated applications for the user (not the developer). These applications are usually executables.

There are many advantages associated with taking this layered approach to software development, as we shall see in this book.

1.5 THE STRUCTURE OF THIS BOOK

This book is partitioned into six major parts, each of which deals with a major topic and consists of a number of chapters. These chapters deal with techniques that help to achieve the goals of each part.

Part I This part is an introduction to C++ template classes. We define what templates are, how to create them and how to use them in financial engineering applications. We give an overview of the Standard Template Library (STL). This is a C++ library consisting of template classes for a wide range of data containers, algorithms and functionality for navigating in these containers. We develop a number of template classes based on STL that we can use in financial engineering applications.

Part II In this part we create classes and code that will be used when approximating partial differential equations by finite difference methods. First, we create template classes for arrays, vectors and matrices as well as the corresponding mathematical operations on them. Furthermore, we introduce several classes that solve linear systems of equations. These classes implement direct and iterative matrix solvers in numerical linear algebra. Second, we create a number of other foundation classes that we need in numerical differentiation and integration. Finally, some useful classes for statistics are introduced.

Part III This part represents the start of the book as far as the mathematical core is concerned. We motivate the finite difference method by applying it to a simple first-order ordinary differential equation in Chapter 11. This chapter discusses the most important ideas and schemes that will serve us well in later chapters. Continuing from Chapter 11, we introduce stochastic differential equations and the finite difference schemes needed in order to approximate them. We also propose several schemes to approximate two-point boundary value problems. Special attention is paid to the Crank–Nicolson scheme and why it fails to approximate the solution of the convection-diffusion equation in certain circumstances. It is in this part of the book that we introduce the class of exponentially fitted schemes and explain how they resolve the spurious oscillation problems associated with Crank–Nicolson.

Part IV In this part we introduce the one-factor and two-factor Black–Scholes equations and devise appropriate finite difference schemes for them. Before we reach this level of Nirvana, we begin with the one-dimensional heat equation and discuss explicit and implicit finite difference schemes to approximate its solution. The schemes are extensions of the time-independent schemes that we introduced in Part III. Slightly increasing the level of difficulty, we discuss the Crank–Nicolson and fully implicit schemes for the one-dimensional convection-diffusion equation (and its specialisation, the Black–Scholes equation). We analyse the schemes in some detail, discussing why they work, when they

do not work and how to produce fitted schemes that approximate the solution and the delta of the Black–Scholes equation.

Proceeding to two-factor problems, we propose Alternating Direction Implicit (ADI) and splitting methods and compare their relative merits.

Part V In this part we give an introduction to design patterns. Design is about alternatives and we have many choices when designing a system as the choices are determined by the software requirements. We begin with an introduction to some general design principles. In particular, we focus on templates and inheritance and why they are competitors. We also introduce the important notion of *delegation* whose understanding is fundamental to design patterns.

The main objective in Part V is to show how the famous design patterns of GOF (see Gamma *et al*., 1995) are applied to financial engineering applications. We pay special attention to choosing appropriate examples and to a discussion of the advantages of design patterns in this particular context. Three chapters are devoted to the Creational, Structural and Behavioural patterns.

Part VI This part contains a number of chapters that are of particular interest to financial engineers and IT personnel who write financial engineering applications. First, we give an introduction to the Extensible Markup Language (XML), a W3C standard for interoperable data representation. We also describe how it is used in option pricing applications in this book. XML will become more important in the financial world in the coming years as evidenced by the work seen with FpML and FIX. We also discuss classes and code that allow C++ code to communicate with Excel. Finally, we introduce a number of design patterns that are very useful for the current work.

1.6 PEDAGOGICAL APPROACH

In general, our approach is incremental in the sense that we begin with simple examples to illustrate the theory and progress to larger problems and examples. This approach applies to the theory of finite differences for partial differential equations as well as the C++ code and design patterns. For example, our main objective is to model one-factor and two-factor Black–Scholes equations using finite differences. The main 'flow' in this case is:

- Finite differences for scalar, linear first-order ordinary differential equations (ODEs).
- Finite differences for stochastic differential equations (SDEs).
- Two-point boundary value problems (special case: stationary convection-diffusion).
- Explicit and implicit finite difference schemes for the heat equation.
- Crank–Nicolson and fitting schemes for the one-factor Black–Scholes equation.
- Approximating the Greeks.
- Alternating Direction Implicit (ADI) and splitting schemes for two-factor Black–Scholes equations.

In a similar vein, we build up C++ expertise as follows:

- The C++ template class.
- The Standard Template Library (STL) and its applications to financial engineering.

- The Property pattern and the modelling of financial instruments.
- C++ classes for ODEs and SDEs.
- C++ foundation classes: vectors, matrices and statistics.
- C++ classes for the heat equation.
- Modelling Black–Scholes with C++.
- C++ for ADI and splitting methods.

In short, we adopt the following rule-of-thumb:

1. Get it working.
2. Then get it right.
3. Then get it optimised.

One step at a time!

1.7 WHAT THIS BOOK IS NOT

First, this is not an introductory book on C++. We assume that the reader has a working knowledge of this language. Second, this book assumes that the reader knows what an option is, what Black–Scholes is, and so on. Finally, this is not a book on the theory of partial differential equations and their approximation using finite differences, but is rather a book that motivates finite difference schemes and applies them to financial engineering applications using C++.

1.8 SOURCE CODE ON THE CD

You can use the source code on the accompanying CD free of charge in your applications provided you acknowledge the author:

Each source file has this copyright statement. Any questions or suggestions should be sent to Daniel Duffy at info@datasim.nl.