Advanced Quantitative Finance with C++

Create and implement mathematical models in C++ using Quantitative Finance

Alonso Peña, Ph.D.

Advanced Quantitative Finance with C++

Create and implement mathematical models in C++ using Quantitative Finance

Alonso Peña, Ph.D.

Advanced Quantitative Finance with C++

Credits

Author
Alonso Peña, Ph.D.

Reviewers
Marco Airoldi

Joseph Smidt

Commissioning Editor
Grant Mizen

Acquisition Editor
Harsha Bharwani

Content Development Editor
Amit Ghodake

Technical Editor
Humera Shaikh

Copy Editor
Laxmi Subramanian

Project Coordinator
Harshal Ved

Proofreader
Clyde Jenkins

Graphics
Sheetal Aute

Ronak Dhruv

Valentina Dsilva

Disha Haria

Abhinash Sahu

Indexer
Hemangini Bari

Production Coordinator
Kyle Albuquerque

Cover Work
Nilesh Bambardekar

About the Author

Alonso Peña, Ph.D. is an SDA Professor at the SDA Bocconi School of Management in Milan. He has worked as a quantitative analyst in the structured products group for Thomson Reuters Risk and for Unicredit Group in London and Milan. He holds a Ph.D. degree from the University of Cambridge on Finite Element Analysis and the Certificate in Quantitative Finance (CQF) from 7city Learning, the U.K. He has lectured and supervised graduate and post-graduate students from the universities of Oxford, Cambridge, Bocconi, Bergamo, Pavia, Castellanza, and the Politecnico di Milano. His area of expertise is the pricing of financial derivatives, in particular, structured products.

He has publications in the fields of Quantitative Finance, applied mathematics, neuroscience, and the history of science. He has been awarded the Robert J. Melosh Medal—first prize for the best student paper on Finite Element Analysis, Duke University, USA; and the Rouse Ball Travelling Studentship in Mathematics, Trinity College, Cambridge. He has been to the Santa Fe Institute, USA, to study complex systems in social sciences.

His publications include the following:

- *The One Factor Libor Market Model Using Monte Carlo Simulation: An Empirical Investigation*
- *On the Role of Behavioral Finance in the Pricing of Financial Derivatives: The Case of the S&P 500*
- *Option Pricing with Radial Basis Functions: A Tutorial*
- *Application of extrapolation processes to the finite element method*
- *On the Role of Mathematical Biology in Contemporary Historiography*

He is currently working as a tutor for CQF (Fitch Learning) and a visiting faculty for the Indian Institute for Quantitative Finance, Mumbai.

He lives in Italy with his wife Marcella, his daughters Francesca and Isabel, and his son Marco.

Acknowledgments

I would like to thank many people who have made this book a reality. First the magnificent support, enthusiasm, and patience of the entire team at Packt Publishing, particularly Harsha, Amit, Humera, and Harshal. To Dr. Pattabi Raman (Numerical Solution (U.K.) Ltd.), for his expert advice on C++. To Dr. Marco Airoldi for his knowledgeable and detailed review of the book. To the SDA Bocconi School of Management including my colleagues and students from the MBA, graduate, and undergraduate courses. To the many persons I have been privileged to work with and to teach from the Universities of Cambridge, Oxford, Bocconi, LIUC Castellanza, Bergamo, Pavia, and Politecnico di Milano. The many extraordinary quants from the Certificate in Quantitative Finance, Fitch Learning, London, as well as from Unicredit Group and Thomson Reuters. Finally, to my wife, Marcella, and my children, Francesca, Isabel, and Marco—you all always remind me that "The true voyage of discovery consists not in seeking new landscapes but in having new eyes to see" (Marcel Proust).

About the Reviewer

Marco Airoldi received his Ph.D. in Theoretical Condensed Matter Physics in 1995 from the International School for Advanced Studies (SISSA). He moved definitively to finance in 1999. Marco has been chosen as the head of financial engineering in one of the top financial institutions in Italy.

His expertise includes the Monte Carlo simulation for option pricing and pricing system architectures.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Quantitative Finance is a highly complex interdisciplinary field, which covers mathematics, finance, and information technology. Navigating it successfully requires specialist knowledge from many sources, such as financial derivatives, stochastic calculus, and Monte Carlo simulation. Crucially, it also requires a hands-on ability to transform theory into practice effectively.

In *Advanced Quantitative Finance with C++*, we provide a guided tour through this exciting field. The key mathematical models used to price financial derivatives are explained as well as the main numerical models used to solve them. In particular, equity, currency, interest rates, and credit derivatives are discussed. The book also presents how to implement these models in C++ step by step. Several fully working, complete examples are given that can be immediately tested by the reader to support and complement their learning.

What this book covers

Chapter 1, *What is Quantitative Finance?*, gives a brief introduction to Quantitative Finance, delimits the subject to option pricing with C++, and describes the structure of the book.

Chapter 2, *Mathematical Models*, offers a summary of the fundamental models used to price derivatives in modern financial markets.

Chapter 3, *Numerical Methods*, reviews the three main families of numerical methods used to solve the mathematical models described in the *Chapter 2*, *Mathematical Models*.

Chapter 4, *Equity Derivatives in C++*, demonstrates the concrete pricing of equity derivatives using C++ in a basic contract (European Call/Put), and an advanced contract (multi-asset options).

Chapter 5, *Foreign Exchange Derivatives with C++*, illustrates the pricing of foreign exchange derivatives using C++ in a basic contract (continuous barrier) and an advanced contract (terminal barrier).

Chapter 6, *Interest Rate Derivatives with C++*, shows the pricing of interest rate derivatives using C++ in a basic contract and an advanced Interest Rate Swap (IRS).

Chapter 7, *Credit Derivatives with C++*, demonstrates the concrete pricing of credit derivatives using C++ in a basic contract (Merton model) and an advanced contract (**Credit Default Swap** (**CDS**)).

Appendix A, *C++ Numerical Libraries for Option Pricing*, gives a short guide to the various numerical libraries that can be used for option pricing.

Appendix B, *References*, lists all the bibliographic references used throughout the chapters of this book.

What you need for this book

In order to implement the pricing algorithms described in this book, you will need some basic knowledge of C++ and Integrated Development Environment (IDE) of your choice. I have used Code:Blocks, which is a free C, C++, and Fortran IDE, and is highly extensible and fully configurable. You can download it from `http://www.codeblocks.org/`. You will also need a C++ compiler. I have used MinGW, which is a part of the GNU Compiler Collection (GCC), including C, C++, ADA, and Fortran compilers. This compiler can be downloaded from `http://www.mingw.org/`.

Who this book is for

This book is ideal for quantitative analysts, risk managers, actuaries, and other professionals working in the field of Quantitative Finance who want a quick reference or a hands-on introduction to pricing of financial derivatives. Postgraduate, MSc, and MBA students following university courses on derivatives in corporate finance and/or risk management will also benefit from this book. It could be used effectively by advanced undergraduate students who are interested in understanding these fascinating financial instruments. A basic familiarity with programming concepts, C++ programming language, and undergraduate-level calculus is required.

Conventions

In this book, you will find a number of styles of text that distinguish among different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "An important feature of this algorithm is the function in code snippet 2 (`random.cpp`)."

A block of code is set as follows:

```
for (int i=0; i < N; i++)
{
  double epsilon = SampleBoxMuller();  // get Gaussian draw
  S[i+1] = S[i]*(1+r*dt+sigma*sqrt(dt)*epsilon);
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "In this book, all the programs are implemented with the newest standard C++11 using **Code::Blocks** (`http://www.codeblocks.org`) and **MinGW** (`http://www.mingw.org`)".
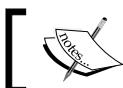
> Warnings or important notes appear in a box like this.

> Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

What is Quantitative Finance? **1**

Quantitative Finance studies the application of quantitative techniques to the solution of problems in finance. It spans diverse areas such as the management of investment funds and insurance companies, the control of financial risks for manufacturing companies and banking industry, and the behavior of the financial markets. Quantitative Finance is eminently interdisciplinary building upon key expertise from the disciplines of finance, mathematics, and informatics.

In this book, we will focus on one aspect of Quantitative Finance—the pricing of financial derivatives using the programming language C++. In the following sections, we will describe the main features of the three key disciplines that constitute Quantitative Finance:

- Finance
- Mathematics
- Informatics

Discipline 1 – finance (financial derivatives)

In general, a financial derivative is a contract between two parties who agree to exchange one or more cash flows in the future. The value of these cash flows depends on some future event, for example, that the value of some stock index or interest rate being above or below some predefined level. The activation or triggering of this future event thus depends on the behavior of a variable quantity known as **the underlying**. Financial derivatives receive their name because they derive their value from the behavior of another financial instrument.

As such, financial derivatives do not have an intrinsic value in themselves (in contrast to bonds or stocks); their price depends entirely on the underlying.

A critical feature of derivative contracts is thus that their future cash flows are probabilistic and not deterministic. The future cash flows in a derivative contract are contingent on some future event. That is why derivatives are also known as **contingent claims**. This feature makes these types of contracts difficult to price.

The following are the most common types of financial derivatives:

- Futures
- Forwards
- Options
- Swaps

Futures and forwards are financial contracts between two parties. One party agrees to buy the underlying from the other party at some predetermined date (the maturity date) for some predetermined price (the delivery price). An example could be a one-month forward contract on one ounce of silver. The underlying is the price of one ounce of silver. No exchange of cash flows occur at inception (today, *t=0*), but it occurs only at maturity (*t=T*). Here *t* represents the variable time. Forwards are contracts negotiated privately between two parties (in other words, **Over The Counter** (**OTC**)), while futures are negotiated at an exchange.

Options are financial contracts between two parties. One party (called the **holder** of the option) pays a premium to the other party (called the **writer** of the option) in order to have the right, but not the obligation, to buy some particular asset (the underlying) for some particular price (the strike price) at some particular date in the future (the maturity date). This type of contract is called a **European Call** contract.

Example 1

Consider a one-month call contract on the S&P 500 index. The underlying in this case will be the value of the S&P 500 index. There are cash flows both at inception (today, *t=0*) and at maturity (*t=T*). At inception, (*t=0*) the premium is paid, while at maturity (*t=T*), the holder of the option will choose between the following two possible scenarios, depending on the value of the underlying at maturity *S(T)*:

- Scenario A: To exercise his/her right and buy the underlying asset for *K*
- Scenario B: To do nothing if the value of the underlying at maturity is below the value of the strike, that is, *S(T)<K*

The option holder will choose Scenario A if the value of the underlying at maturity is above the value of the strike, that is, *S(T)>K*. This will guarantee him/her a profit of *S(T)-K*. The option holder will choose Scenario B if the value of the underlying at maturity is below the value of the strike, that is, *S(T)<K*. This will guarantee him/her to limit his/her losses to zero.

Example 2

An **Interest Rate Swap** (**IRS**) is a financial contract between two parties A and B who agree to exchange cash flows at regular intervals during a given period of time (the life of a contract). Typically, the cash flows from A to B are indexed to a fixed rate of interest, while the cash flows from B to A are indexed to a floating interest rate. The set of fixed cash flows is known as the **fixed leg**, while the set of floating cash flows is known as the **floating leg**. The cash flows occur at regular intervals during the life of the contract between inception (*t=0*) and maturity (*t=T*). An example could be a fixed-for-floating IRS, who pays a rate of 5 percent on the agreed notional *N* every three months and receives EURIBOR3M on the agreed notional *N* every three months.

Example 3

A futures contract on a stock index also involves a single future cash flow (the delivery price) to be paid at the maturity of the contract. However, the payoff in this case is uncertain because how much profit I will get from this operation will depend on the value of the underlying at maturity.

If the price of the underlying is above the delivery price, then the payoff I get (denoted by function *H*) is positive (indicating a profit) and corresponds to the difference between the value of the underlying at maturity *S(T)* and the delivery price *K*. If the price of the underlying is below the delivery price, then the payoff I get is negative (indicating a loss) and corresponds to the difference between the delivery price *K* and the value of the underlying at maturity *S(T)*. This characteristic can be summarized in the following payoff formula:

$$H(S(T)) = S(T) - K$$

Equation 1

Here, *H(S(T))* is the payoff at maturity, which is a function of *S(T)*. Financial derivatives are very important to the modern financial markets. According to the **Bank of International Settlements** (**BIS**) as of December 2012, the amounts outstanding for OTC derivative contracts worldwide were Foreign exchange derivatives with 67,358 billion USD, Interest Rate Derivatives with 489,703 billion USD, Equity-linked derivatives with 6,251 billion USD, Commodity derivatives with 2,587 billion USD, and Credit default swaps with 25,069 billion USD. For more information, see `http://www.bis.org/statistics/dt1920a.pdf`.

Discipline 2 – mathematics

We need mathematical models to capture both the future evolution of the underlying and the probabilistic nature of the contingent cash flows we encounter in financial derivatives.

Regarding the contingent cash flows, these can be represented in terms of the payoff function *H(S(T))* for the specific derivative we are considering. Because *S(T)* is a stochastic variable, the value of *H(S(T))* ought to be computed as an expectation *E[H(S(T))]*. And in order to compute this expectation, we need techniques that allow us to predict or simulate the behavior of the underlying *S(T)* into the future, so as to be able to compute the value of *ST* and finally be able to compute the mean value of the payoff *E[H(S(T))]*.

Regarding the behavior of the underlying, typically, this is formalized using **Stochastic Differential Equation**s (**SDEs**), such as **Geometric Brownian Motion** (**GBM**), as follows:

$$ds = \mu S dt + \sigma S dW$$

Equation 2

The previous equation fundamentally says that the change in a stock price (*dS*), can be understood as the sum of two effects—a deterministic effect (first term on the right-hand side) and a stochastic term (second term on the right-hand side). The parameter μ is called the **drift**, and the parameter σ is called the **volatility**. *S* is the stock price, *dt* is a small time interval, and *dW* is an increment in the Wiener process.

This model is the most common model to describe the behavior of stocks, commodities, and foreign exchange. Other models exist, such as jump, local volatility, and stochastic volatility models that enhance the description of the dynamics of the underlying.

Regarding the numerical methods, these correspond to ways in which the formal expression described in the mathematical model (usually in continuous time) is transformed into an approximate representation that can be used for calculation (usually in discrete time). This means that the SDE that describes the evolution of the price of some stock index into the future, such as the FTSE 100, is changed to describe the evolution at discrete intervals. An approximate representation of an SDE can be calculated using the Euler approximation as follows:

$$S_{t+1} - S_t = \mu S_t \Delta t + \sigma S_t d\Delta W$$

Equation 3

The preceding equation needs to be solved in an iterative way for each time interval between now and the maturity of the contract. If these time intervals are days and the contract has a maturity of 30 days from now, then we compute tomorrow's price in terms of todays. Then we compute the day after tomorrow as a function of tomorrow's price and so on. In order to price the derivative, we require to compute the expected payoff *E[H(ST)]* at maturity and then discount it to the present. In this way, we would be able to compute what should be the fair premium π associated with a European option contract with the help of the following equation:

$$\pi = \exp(-rT) \times E[H(\mathrm{S}_T)] = \exp(-rT) \times E[\max(\mathrm{S}_T - K, 0)]$$

Equation 4

Discipline 3 – informatics (C++ programming)

What is the role of C++ in pricing derivatives? Its role is fundamental. It allows us to implement the actual calculations that are required in order to solve the pricing problem. Using the preceding techniques to describe the dynamics of the underlying, we require to simulate many potential future scenarios describing its evolution. Say we ought to price a futures contract on the EUR/USD exchange rate with one year maturity. We have to simulate the future evolution of EUR/USD for each day for the next year (using equation 3). We can then compute the payoff at maturity (using equation 1). However, in order to compute the expected payoff (using equation 4), we need to simulate thousands of such possible evolutions via a technique known as **Monte Carlo simulation**. The set of steps required to complete this process is known as an **algorithm**. To price a derivative, we ought to construct such algorithm and then implement it in an advanced programming language such as C++. Of course C++ is not the only possible choice, other languages include Java, VBA, C#, Mathworks Matlab, and Wolfram Mathematica. However, C++ is an industry standard because it's flexible, fast, and portable. Also, through the years, several numerical libraries have been created to conduct complex numerical calculations in C++. Finally, C++ is a powerful modern object-oriented language.

It is always difficult to strike a balance between clarity and efficiency. We have aimed at making computer programs that are self-contained (not too object oriented) and self-explanatory. More advanced implementations are certainly possible, particularly in the context of larger financial pricing libraries in a corporate context. In this book, all the programs are implemented with the newest standard C++11 using **Code::Blocks** (`http://www.codeblocks.org`) and **MinGW** (`http://www.mingw.org`).

The Bento Box template

A Bento Box is a single portion take-away meal common in Japanese cuisine. Usually, it has a rectangular form that is internally divided in compartments to accommodate the various types of portions that constitute a meal. In this book, we use the metaphor of the Bento Box to describe a visual template to facilitate, organize, and structure the solution of derivative problems. The Bento Box template is simply a form that we will fill sequentially with the different elements that we require to price derivatives in a logical structured manner. The Bento Box template when used to price a particular derivative is divided into four areas or boxes, each containing information critical for the solution of the problem. The following figure illustrates a generic template applicable to all derivatives:

1 DERIVATIVE CONTRACT

In this box the description of the contract goes. Identify the underlying. Specify the counterparties involved, the cashflows, the payment dates, any other conditions and the payoff function in terms of the underlying.

2 MATH MODEL

In this box we define the mathematical model that describes the dynamics of the underlying specified in box 1.

3 NUM METHOD

In this box we specify the numerical method that will be use to solve the model specified in box 2.

4 ALGORITHM

In this final box we put together the specifications (box 1), the mathematical description of the underlying (box 2) and the numerical method used to solve the model (box 3) via a computer algorithm. The algorithm is essentially a series of steps that takes us from the input data (market variables) via a computation process to the output (typically the price, premium or mark to market of the derivative.) This algorithm will be blueprint to construct computer code in C++.

THE BENTO BOX TEMPLATE

The Bento Box template – general case

The following figure shows an example of the Bento Box template as applied to a simple European Call option:

1

European Call option on stock index FTSE100.
Counterparties are A and B. Underlying is FTSE100 index. At t=0, A pays B a **premium**. At maturity (T=3 months), A will have the right (but not the obligation) to buy the underlying from B for the strike price K. The payoff at maturity H is:

$$H(S_T) = \max(S_T - K, 0)$$

DERIVATIVE CONTRACT

2

The **underlying**, being a stock index, can be described using Geometric Brownian Motion (GBM):

$$dS = rSdt + \sigma SdW$$

MATH MODEL

3

Use **Monte Carlo Simulation** as a method for the computation of the value of the discounted expected payoff.

$$\pi = \exp(-rT) \times E^Q[H(S_T)]$$

NUM METHOD

4

INPUT: spot-price, strike, maturity, risk-free rate, volatility number simulations (M), time discretization number periods (N)
OUTPUT: estimate for the premium
PROCESS:
for i=1:M
   for i=1:N
     *Compute GBM formula and advance one timestep
   end
   *Compute underlying at maturity
   *Compute payoff at maturity
end
*Compute premium as discounted average value of payoffs

ALGORITHM

THE BENTO BOX TEMPLATE

The Bento Box template – European Call option

In the preceding figure, we have filled the various compartments, starting in the top-left box and proceeding clockwise. Each compartment contains the details about our specific problem, taking us in sequence from the conceptual (box 1: derivative contract) to the practical (box 4: algorithm), passing through the quantitative aspects required for the solution (box 2: mathematical model and box 3: numerical method).

Summary

This chapter gave an overview of the main elements of Quantitative Finance as applied to pricing financial derivatives. The Bento Box template technique will be used in the following chapters to organize our approach to solve problems in pricing financial derivatives. We will assume that we are in possession with enough information to fill box 1 (derivative contract). Further details about the mathematical models (box 2) will be described in *Chapter 2*, *Mathematical Models*.

2
Mathematical Models

In the previous chapter, we described the Bento Box template as a methodology for structuring our approach to price financial derivatives. In the context of the Bento Box template, this chapter corresponds to box 2—mathematical models. Here we review some of the key mathematical models used in the financial derivatives markets today to describe the behavior of the underlying. In particular, the future evolution of the underlying. The following are the examples of these underlyings:

- An equity or stock
- An exchange rate
- An interest rate
- A credit rating

Equity

In the equity asset class, the underlying is the price of a company stock. For instance, the current price of one share of Vodafone PLC (VOD.L) as quoted in the London Stock Exchange (`www.londonstockexchange.com`) at some particular time. The price could be £2.32 and the time could be 11:33:24 on May 13, 2013.

In mathematical terms, thus, the price of a stock can be represented as a scalar function of the current time t. We will denote this function as $S(t)$. Note that in technical terms, $S(t)$ is a time series, which even though apparently continuous (with $C[0]$ continuity), is in reality discontinuous (subject to jumps). In addition, it is not a well-behaved function, that is, its first derivative does not exist.

We are going to model *S(t)* as an stochastic variable. And all the constructions that we build around this value, such as the value of the payoff *H(S_t)* will be in consequence stochastic functions. In this situation, we are required not to use the standard tools of calculus (such as Taylor series, derivatives, Riemann integral), but are instead required to use the tools from stochastic calculus (such as Ito lemma, Radon-Nykodym derivative, Riemann-Stieltjes integral) to advance our modeling.

In this context, the behavior of the variable *S(t)* can be described by an SDE. In the case of equities, the standard SDE used to describe the behavior of equities is called **GBM.** Under the so-called real-world probability measure *P*, GBM is formally represented in continuous time as follows:

$$dS = \mu S dt + \sigma S dW^P$$

Equation 1

However, in literature, this representation is not used for the pricing of financial derivatives. It is substituted by the following representation under the risk-neutral measure *Q*:

$$dS = rS dt + \sigma S dW^Q$$

Equation 2

In the preceding equation, we have substituted the drift μ by the risk-free rate r of interest, σ is the volatility, and *dW* is the increment of a Wiener process. Equation 2 can be further represented as follows:

$$\frac{dS}{S} = r dt + \sigma dW^Q$$

In the preceding equation, we can identify the term *dS/S* on the left hand side (**LHS**) of the equation as the return of the equity. Thus, the two terms on the right hand side (**RHS**) of the equation are a "drift term" and a "volatility term". Each of these terms are "scaled" by parameters μ and σ, which are calibrated to current market prices of traded instruments, such as call and put options.

Note that equation 2 is the fundamental equation used to describe the underlyings in the word of financial derivatives.

For the purpose of pricing derivatives, we require to transform equation 2 from continuous time into discrete time to model the behavior of stocks, say every day ($\Delta t = 1$ days, but in finance we always work in annualized terms, so $\Delta t = 1/365 = 0.00274$ years). We can easily approximate equation 2 by using the Euler-Murayama discretization as follows:

$$\Delta S = rS\Delta t + \sigma S \Delta W$$

$$S_{t+1} - S_t = rS_t \Delta t + \sigma S_t \varepsilon_t \sqrt{\Delta t}$$

$$S_{t+1} = S_t + rS_t \Delta t + \sigma S_t \varepsilon_t \sqrt{\Delta t} = S_t \left(1 + r\Delta t + \sigma \varepsilon_t \sqrt{\Delta t} \right)$$

Equation 3

In the preceding equation, we approximate the differential of the Wiener process as the square root of delta *t* multiplied by a draw from a Gaussian distribution with zero mean and standard deviation 1 (*N(0,1)*). Equation 3 is a linear iterative equation, which we can compute by having a starting value of *S0* for a number of time steps *S1*, *S2*, …, *SN*. We only need the values of the parameters *r* and *σ*, and a Gaussian random number generator for the value of *ε*.

In order to compute the draw from the cumulative standard normal distribution, we will use a method called the **Box-Muller** method. The Box Muller method allows us to convert uniform random numbers into Gaussian random numbers. This is very useful because in many computer libraries we can find standard functions to generate random numbers from a uniform distribution (for example, function `rand()` in C) and through the Box Muller method, we can generate the Gaussian draws we need. We will discuss more about this in *Chapter 4*, *Equity Derivatives in C++*.

For example, imagine that we would like to simulate the behavior of the stock of company ABC for the next four days. The current value of the stock is 100 EUR. The risk-free interest rate stands at 5 percent per annum (**p.a.**), and the volatility is at 30 percent pa. How shall we proceed?

First we construct a time grid for the business days in which we need the values (note that there are 255 business days in a year). These are *t0*, *t1*, *t2*, *t3*, and *t4*. These correspond respectively to Monday, Tuesday, Wednesday, Thursday, and Friday. In finance, we always work in annualized terms and, therefore, these dates correspond to $\Delta t = 1/255$, so *t0=0*, *t1=1/255*, *t2=2/255*, *t3=3/255*, and *t4=4/255*.

We then need the grid of stock prices *S(t)*. These are *S(t0)*, *S(t1)*, *S(t2)*, *S(t3)*, and *S(t4)*, which we will assign respectively to Monday, Tuesday, Wednesday, Thursday, and Friday that are represented as *S0*, *S1*, *S2*, *S3*, and *S4*. We already know the value of *S0*, which is the initial price (as observed today), that is, *S0=100* on Monday.

Before doing that, we will need a vector of draws from the cumulative standard normal distribution as follows:

$$\varepsilon_1 = +0.4423, \varepsilon_2 = -0.1170, \varepsilon_3 = +0.0291, \varepsilon_4 = +0.6872$$

We can then apply equation 3 iteratively to go from the value of Monday *S0* to the value of Tuesday *S1* as follows:

$$S_1 = S_0 \left(1 + r\Delta t + \sigma \varepsilon_t \sqrt{\Delta t} \right)$$

Alternatively, we can go from the value of Monday *S0* to the value of Tuesday *S1* with the numerical values as follows:

$$S_1 = (100)\left(1 + (0.05)(1/255) + (0.30)(+0.4423)\sqrt{(1/255)}\right) = 102.12$$

We can then calculate the values for the rest of the days as follows:

$$S_2 = (102.12)\left(1 + (0.05)(1/255) + (0.30)(-0.1170)\sqrt{(1/255)}\right) = 99.47$$

$$S_3 = (99.47)\left(1 + (0.05)(1/255) + (0.30)(+0.1121)\sqrt{(1/255)}\right) = 101.82$$

$$S_4 = (101.82)\left(1 + (0.05)(1/255) + (0.30)(+0.7373)\sqrt{(1/255)}\right) = 104.21$$

If we put together these set of calculated stock prices and plot them against time, we obtain the following graph:

Simulated stock prices
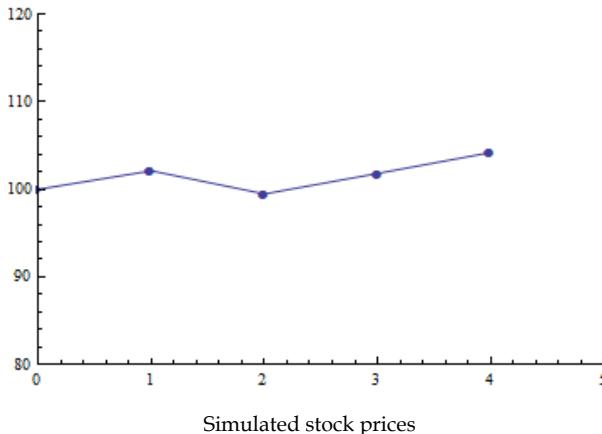
Foreign exchange

In the forex asset class, the underlying is the value of an exchange rate. For example, the current exchange rate between the euro (EUR) and the British pound sterling (GBP) at some particular time. The exchange rate could be *EUR/GBP = 1.31*, meaning that £1 will be exchanged by € 1.31, and the current time could be 11:33:24 on May 13, 2013.

Thus, in mathematical terms, the exchange rate can be represented as a function *X(t)*, which is a scalar function of time, just like in the case of equities. The exchange rate *X(t)* is thus modeled as an stochastic variable. In mathematical terms, the behavior of *X(t)* is described using an SDE just like in the case of equities. However, while for equities we used GBM, in the case of forex, we will use a variation that comes from the work of (Garman-Kohlhagen 1983). According to this model, the stochastic differential equation for exchange rates can be expressed as follows:

$$dX = (r_d - r_f)Xdt + \sigma XdW^Q$$

Equation 4

In the preceding equation, *rd* and *rf* represent the domestic and the foreign risk-free interest rates. The volatility σ is a parameter calibrated to market-quoted instruments.

As we did earlier, before proceeding for the purposes of pricing derivatives, we require to transform equation 1 from continuous time into discrete time to model the behavior of exchange rates say every day ($\Delta t = 1$ day).

We again apply the Euler-Murayama discretization to equation 1 by transforming the differential dX into a difference ΔX, keeping the constants constants *rd*, *rf*, and σ unchanged, and approximating the differential of the Wiener process as the square root of delta *t* multiplied by a draw from a cumulative standard normal distribution, as follows:

$$\Delta X = (r_d - r_f)X\Delta t + \sigma X \Delta W$$

$$X_{t+1} - X_t = (r_d - r_f)X_t\Delta t + \sigma X_t \varepsilon_t \sqrt{\Delta t}$$

$$X_{t+1} = X_t + (r_d - r_f)X_t\Delta t + \sigma X_t \varepsilon_t \sqrt{\Delta t} = X_t\left(1 + (r_d - r_f)\Delta t + \sigma \varepsilon_t \sqrt{\Delta t}\right)$$

Equation 5

As in the case of equation 2, equation 5 is also a linear iterative equation, which we can compute iteratively by having a starting value of *X0* for a number of time steps *X1*, *X2*, ..., *XN*. We only need the values of the parameters *rd*, *rf*, and *sigma* and a Gaussian random number generator for the value of *epsilon*.

For example, imagine that we would like to simulate the behavior of the EUR/USD exchange rate for the next four days at the last quoted value of the business day (known as **end of day** (**EOD**)). The current value of the exchange rate EUR/USD is 1.33. The domestic risk-free interest rate is 5 percent p.a. and the foreign risk-free interest rate is 3 percent pa, while the volatility is at 30 percent pa. How shall we proceed?

First we construct a time grid for the business days in which we need the values (note that there are 255 business days in a year). These are *t0*, *t1*, *t2*, *t3*, and *t4*. These correspond to Monday, Tuesday, Wednesday, Thursday, and Friday respectively, in turn corresponding to *t0=0*, *t1=1/255*, *t2=2/255*, *t3=3/255*, and *t4=4/255* in annualized terms.

We then need the grid of EOD exchange rates *X (t)*. These are *X0*, *X1*, *X2*, *X3*, and *X4*. We already know the value of *X0*, which is the initial FX rate (as observed today), that is, *X0=1.33* on Monday. As we did earlier, before proceeding, we compute a vector of draws from the cumulative standard normal distribution to obtain the following:

$$\varepsilon_1 = +0.4423, \varepsilon_2 = -0.1170, \varepsilon_3 = +0.0291, \varepsilon_4 = +0.6872$$

We can then apply equation 3 iteratively to go from the value of Monday *X0* to the value of Tuesday *X1* as follows:

$$X_1 = X_0 \left(1 + (r_d - r_f)\Delta t + \sigma \varepsilon_t \sqrt{\Delta t} \right)$$

Alternatively, we can go from the value of Monday *S0* to the value of Tuesday *S1* with the numerical values as follows:

$$X_1 = (1.33)\left(1 + (0.05 - 0.03)(1/255) + (0.30)(+0.4423)\sqrt{(1/255)}\right) = 1.52$$

We can then calculate the values for the rest of the days as follows:

$$X_2 = (1.52)\left(1 + (0.05 - 0.03)(1/255) + (0.30)(+0.4423)\sqrt{(1/255)}\right) = 1.47$$

$$X_3 = (1.47)\left(1 + (0.05 - 0.03)(1/255) + (0.30)(+0.4423)\sqrt{(1/255)}\right) = 1.32$$

$$X_4 = (1.32)\left(1 + (0.05 - 0.03)(1/255) + (0.30)(+0.4423)\sqrt{(1/255)}\right) = 1.44$$

If we put together these sets of computed rates and plot them as a function of time, we obtain the following graph:

Simulated exchange rates
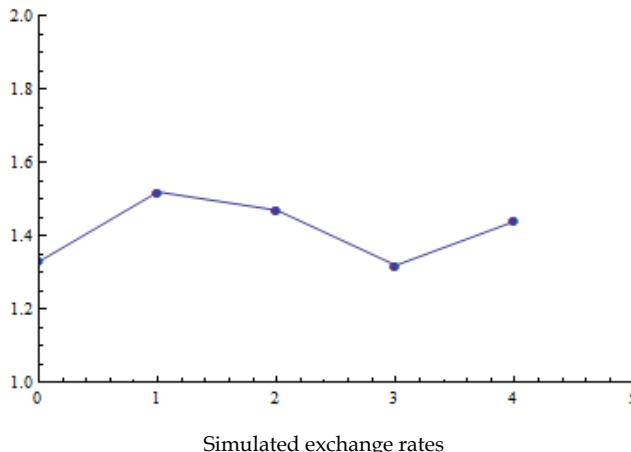
Interest rates

In the interest rate asset class, the underlying is an interest rate. Interest rates are quite complex. We can see this if we consider the question "what is the interest rate today?" The answer is certainly not simply "5 percent pa" because we also need to specify the maturity of the rates (*T*) we want to know. So interest has one more dimension than objects like equities or forex. While the currently observed equity value is a scalar quantity, that is, a single number, the current interest rate curve is a vector.

For example, let's consider the spot EURIBOR interest rates observed on May 13, 2013 for the maturities of 1 month, 3 months, 6 months, and 12 months (as published by `http://www.euribor-ebf.eu/`). We denote these spot rates quoted by *R(t,T)* as *R(0,3M)=EURIBOR 3M = 1 percent pa*, *R(0,6M)= EURIBOR 6M = 2 percent pa*, *R(0,9M)=EURIBOR 9M = 3 percent pa*, and *R(0,12M)=EURIBOR 12M = 4 percent pa*. Note that *t=0* because we consider May 13, 2013 as the current date.

How would each of these rates evolve into the future? In other words, how can we model *R(t,T)*? We have the following two choices reflecting the two modelling schools present in literature:

- Short rate models
- Market models

The first is the oldest, while the second is more recent.

In the first model, the key modelling variable is an idealization of the interest rate, the so-called short rate. It is an infinitesimal interest rate *dr* that applies to a very short time interval. To obtain the interest rate that applies to a full period, we ought to add or integrate the effect of all these small interest rates in the period. In the second model, the key modeling variable is an actual quoted or market interest rate, such as LIBOR. That's why these models are called market models in general, and its most famous version is called **Libor Market Model**.

Short rate models

In continuous time, the short rate can be represented by the following SDE developed by Vasicek as follows:

$$dr = \left(\theta - \lambda r\right)dt + \sigma dW$$

Equation 6

The preceding equation is a mean reverting process. The parameter θ is the mean reversion level, λ is the speed of the mean reversion, and σ is the volatility. The parameters θ, λ, and σ control how the stochastic process behaves. The value assigned to θ will be long-term interest rate level to which interest rates will tend, while λ will control how fast interest rates return to the long-term mean level. The volatility σ controls the magnitude of the "jumps" of the process. These parameters can be calibrated to market-quoted instruments, such as options on interest rate swaps (known as **swaptions**).

We can approximate the Vasicek process via the Euler-Murayama methodology described previously to obtain a discretized version of the stochastic process, as follows:

$$\Delta r = \left(\theta - \lambda r_t\right)\Delta t + \sigma \Delta W$$

$$r_{t+1} - r_t = \left(\theta - \lambda r_t\right)\Delta t + \sigma \Delta W$$

$$r_{t+1} = r_t + \left(\theta - \lambda r_t\right)\Delta t + \sigma \varepsilon_t \sqrt{\Delta t}$$

Equation 7

The preceding equation is a linear iterative equation, which we can compute iteratively by having a starting value of *r0* for a number of time steps *r1*, *r2*, …, *rN*. We only need the values of the parameters *theta*, *lambda*, and *sigma*, and a Gaussian random number generator for the value of *epsilon*.

For example, imagine that we would like to simulate the behavior of the short rate of interest for the next four days. The current value of the interest rate is 5 percent. The parameters λ= 1.0 and θ= 2.0, while the volatility is at 30 percent pa. How shall we proceed?

First we construct a time grid for the days in which we need the values. These are *t0*, *t1*, *t2*, *t3*, and *t4* in order to correspond to Monday (*t0*), Tuesday (*t1*), Wednesday (*t2*), Thursday (*t3*), and Friday (*t4*), in turn corresponding to *t0=0*, *t1=1/365*, *t2=2/365*, *t3=3/365*, and *t4=4/365* in annualized terms.

We then need the grid for the short rates *r(t)*. These are *r0*, *r1*, *r2*, *r3*, and *r4*. We will assign them to Monday, Tuesday, Wednesday, Thursday and Friday, respectively. As we already know the value of *r0*, which is the initial interest rate (as observed today), we have that *r0=5 percent* on Monday.

As we did earlier, before proceeding, we compute a vector of draws from the cumulative standard normal distribution to obtain the following:

$$\varepsilon_1 = +0.4423, \varepsilon_2 = -0.1170, \varepsilon_3 = +0.0291, \varepsilon_4 = +0.6872$$

We can then apply equation 3 iteratively to go from the value of Monday *r0* to the value of Tuesday *r1* as follows:

$$r_{t+1} = r_t + \left(\theta - \lambda r_t\right)\Delta t + \sigma \varepsilon_t \sqrt{\Delta t}$$

Alternatively, we can go from the value of Monday *S0* to the value of Tuesday *S1* with the numerical values as follows:

$$r_1 = r_0 + \left(\theta - \lambda r_0\right)\Delta t + \sigma \varepsilon_1 \sqrt{\Delta t}$$

$$r_1 = (0.05) + \left((1) - (2)(0.05)\right)(1/365) + (0.30)(0.4427)\sqrt{(1/365)}$$

We can then calculate the values for the rest of the days as follows:

$$r_2 = (0.05) + \left((1) - (2)(0.05)\right)(1/365) + (0.30)(0.4427)\sqrt{(1/365)}$$

$$r_3 = (0.05) + \left((1) - (2)(0.05)\right)(1/365) + (0.30)(0.4427)\sqrt{(1/365)}$$

$$r_4 = (0.05) + \left((1) - (2)(0.05)\right)(1/365) + (0.30)(0.4427)\sqrt{(1/365)}$$

Market models

Libor Market Model (**LMM)** is an advanced mathematical model used to price interest rate derivatives. Also known as the BGM model after its authors (Brace, Gatarek, Musiela, 1997), the LMM has become hegemonic in the financial markets worldwide. Literature offers a wide range of publications about the LMM, mostly its many variants and its complex advanced issues.

The LMM in reality can be understood not as a single model, but rather as a large family of models (Rebonato 1998) and (Brigo and Mercurio 2006). Its many variants include the number of factors considered, the type of volatility modeling used, the type of correlation modeling used, whether stochastic volatility or SABR are used, whether forward LIBOR rates or swap rates are used, and whether semi-analytical or numerical solution methods are used, among others.

Our methodology and notation closely follows that of (Pelsser 2000), which, even though succinct, provides a clear introduction to the LMM. From all the possible variations of the LMM, in this work, we chose the simplest implementation—embodied in the use of lognormal SDEs (GBM) for the forward rates and a single Wiener process driving the volatility in all rates (that is, a one factor case). Under these conditions, we further explore the use of flat volatility.

We first divide the term structure of interest rates N in a set of forward rates L and a set of reset times T as follows:

$$L_1(t), L_2(t), L_3(t), ..., L_N(t) \quad \text{and} \quad T_0, T_1, T_2, ..., T_N$$

Each of the preceding forward rates will have its own stochastic process driving them, which will result in N stochastic processes. Following BGM, we use Geometric Brownian Motion (BGM) following (Brace, Gatarek, and Musiela 1997) to describe each of these stochastic processes as follows:

$$dL_i = \mu_i L_i dt + \sigma_i L_i dW_i^Q \quad i = 1, N$$

Equation 8

We now further simplify the model and use a single factor driving all the forward rates. This simplification can be later relaxed into a multifactor LMM. It can be shown that by choosing the last rate as a terminal measure, the drift has the form as follows:

$$dL_i = -\sum_{k=i+1}^{N} \frac{\alpha_k \sigma_k(T_n) L_k(T_n)}{1+\alpha_k L_k(T_n)} \sigma_i(t) L_i(T_n) dT + \sigma_i(T_n) dW$$

Note that the drift in the preceding GBM equation is a function of the forward rate, thus not constant but state-dependent.

Given the complexity of the processes in the LMM, it is not possible to obtain a closed-form solution for all the forward rates. This is not a problem, however, as robust numerical methods can be used to solve the discretized version of the forward rate. One important method that is widely used for market models is **Monte Carlo simulation**. The forward rates *Li(Tn)* are the realizations of the spot LIBOR rates. In each column, the forward rate *Li(Tn+1)* is updated using the following discretization:

$$L_i(T_{n+1}) = L_i(T_n) - \sum_{k=i+1}^{N} \frac{\alpha_k \sigma_k(T_n) L_k(T_n)}{1 + \alpha_k L_k(T_n)} \sigma_i(t) L_i(T_n)(T_{n+1} - T_n)$$
$$+ \sigma_i(T_n)(W^{N+1}(T_{n+1}) - W^{N+1}(T_n))$$

Equation 9

The preceding equation, like the ones we have shown for equities and forex, ought to be solved iteratively using simulation. The forward rates can be arranged in an arithmetic table as follows:

$$
\begin{array}{llll}
L_1(T_0) & & & \\
L_2(T_0) & L_2(T_1) & & \\
L_3(T_0) & L_3(T_1) & L_3(T_2) & \\
L_4(T_0) & L_4(T_1) & L_4(T_2) & L_4(T_3)
\end{array}
$$

In the preceding table, the left column represents the term structure of interest rates at time *t=0*. Given a set of LIBOR rates realized on this for *T1, T2, …TN*, we can extract the future rates that we need for simulation as the ones present in the main diagonal of the preceding table.

For example, consider the following paying fixed-for-floating **Interest Rate Swap** (**IRS**) with a notional of 1 million EUR. This IRS pays 5 percent every 3 months and receives EURIBOR 3 months' rate every 3 months. The total maturity of the swap is one year, given the following current term structure of interest rates:

$$
\begin{array}{llll}
L_1(0) & & & \\
L_2(0) & L_2(3\,M) & & \\
L_3(0) & L_3(3\,M) & L_3(6\,M) & \\
L_4(0) & L_4(3M) & L_4(6\,M) & L_4(9\,M)
\end{array}
$$

Compute the present value of this swap using the LMM. How shall we proceed?

Term structure of spot rates of interest at *t=0* are *EURIBOR 3M = 1 percent pa*, *EURIBOR 6M = 2 percent pa*, *EURIBOR 9M = 3 percent pa*, and *EURIBOR 12M = 4 percent pa*.

First, we need to compute the initial forward rates by bootstrapping as observed from time *t=0*. We obtain this using the following equation:

$$L_1(0) = 1\%, L_2(0) = 3\%, L_3(0) = 5\%, L_4(0) = 7\%$$

Then using the iterative equation 9, we compute the forward the rates into the future as follows:

$L_1(0)$			
$L_2(0)$	$L_2(3\,\mathrm{M})$		
$L_3(0)$	$L_3(3\,\mathrm{M})$	$L_3(6\,\mathrm{M})$	
$L_4(0)$	$L_4(3M)$	$L_4(6\,\mathrm{M})$	$L_4(9\,\mathrm{M})$

corresponds to

1.0%			
3.0%	2.2%		
5.0%	4.3%	5.2%	
7.0%	6.6%	7.2%	6.9%

We populate the initial forward rates in the left-most column and advance column-by-column to the right until we have all the values we need, as shown on the right. For more details on the calculation, see the book by (Pelsser 2000). Note that even though an IRS can be priced "statically" (that is, without simulation), we use this example to give an idea of what are the steps that the LMM method requires for calculation.

Credit

In credit derivatives modeling, the underlying is credit risk. Modern methodologies of credit risk measurement can be grouped into two alternative approaches—the structural approach pioneered by (Merton 1974) and a reduced form approach utilizing intensity based models to estimate stochastic hazard rates, pioneered by various authors, including (Jarrow and Turnbull 1995), (Jarrow, Lando, and Turnbull 1997), and (Duffie and Singleton 1999).

Structural models

The structural approach to credit risk assumes that a firm defaults when the market value of its assets is less than the obligations or debt it has to pay. Structural models are, therefore, sometimes also referred to as **asset value models**. These models look at a company's balance sheet and its capital structure to assess its creditworthiness. However, one of the key problems with this approach is that the value of a company's assets is hard to observe directly. The annual report only provides an accounting version of the company's real assets and not their market value. For public companies, the equity is normally observable, as is its debt. (Merton 1974) starts with the assumption of an extremely simplified capital structure of the following form:

$$V(t) = E(t) + D(t)$$

Equation 10

In the preceding equation, *V* represents the value of the firm (the total of the assets of the firm), while *E* is its equity and *D* its debt. The equity *E* is understood as the total value of the equity of the firm, which is equal to the market value of a share (stock) multiplied by the number of shares in the market. For this simplified company, the debt is represented by a single zero coupon bond with maturity *T*.

At this point Merton asks the question "for a company with the preceding capital structure, when will it go in default?" Well, depends on our definition of default. If we take as default the fact that the company cannot pay its obligations at some specific future time *T*, then this condition will be satisfied if the value of the company at time *T*, that is, *V(T)* is larger than the face value of debt *D(T)*. At this moment in time, the bond holders will request payment and the company will be in position to cover it. On the contrary, if at maturity, the value of the firm is less than the value of the debt it has to pay, it, therefore, will not be able to honor its obligations and will be in default. These two scenarios can be defined mathematically as follows:

$$if\ V(T) > D(T)\ \textit{then no default}$$

$$if\ V(T) < D(T)\ \textit{then default}$$

What about equity holders? They are in possession of the company's stock in the end. Considering the preceding two scenarios, we then know that if the company goes in default, they receive nothing, while if the company continues to operate, they receive the difference between *V(T)* and *D(T)* at maturity, thus giving us the following equation:

$$\left. \begin{array}{l} if\ V(\mathrm{T}) > \mathrm{D}\ \ then\ E(T) = V(\mathrm{T}) - \mathrm{D} \\ \quad if\ V(\mathrm{T}) < \mathrm{D}\ \ then\ E(T) = 0 \end{array} \right\} E(T) = \max\left(V(\mathrm{T}) - \mathrm{D}, 0\right)$$

Note that the expression on the left can be written succinctly as the single expression on the right. The expression on the right-hand side represents the payoff to equity holders at maturity. Moreover, this expression has exactly the same form as the payoff of a European Call option with the underlying *V(t)* and strike *D*. Following Merton, we further assume that the dynamics of the firm follow a GBM as follows:

$$dV = rVdt + \sigma V dW^Q$$

Equation 11

And by doing this, we establish a bridge between credit risk and the pricing of equity derivatives. In fact, we can now use all the results from pricing equity derivatives to price structural models of credit risk. Note that the volatility σ is the firm's assets volatility and not the equity volatility. The value of the assets of the firm at time *t=0* can therefore be calculated using the Black-Scholes formula as follows:

$$E_0 = V_0 N(d_1) - D\exp(-rT)N(d_2)$$

Equation 12

$$with \quad d_1 = \frac{\ln(V_0/D) + (r + 0.5\sigma^2)T}{\sigma\sqrt{T}} \quad and \quad d_2 = d_1 - \sigma\sqrt{T}$$

In equation 12, *N()* is the cumulative standard normal distribution. This is a useful expression when we need to calculate problems like **Initial Public Offer** (**IPO**) of a firm and determine, based on the characteristics of a firm, what should be the fair price of its equity.

For example, imagine that we have a firm that has a total value of its assets of 100 million USD. The risk-free interest rate is 5 percent. The volatility of the firm's assets is assumed to be 20 percent. And the face value of its debt stands at 70 million USD payable as a zero coupon bond in four years. What should be the fair value of its equity at time *t=0*?

We use the framework of structural models of credit risk and proceed as follows. The parameters are $V_0 = 100,000,000$, $D = 70,000,000$, $T = 4$, $\sigma = 20\%$, and $r = 5\%$. Using the preceding formula we obtain the following equation:

$$d_1 = \frac{\ln(100/70) + (0.05 + 0.5 \times 0.2^2)4}{0.2\sqrt{4}} = 1.592$$

$$d_2 = 1.592 - 0.20 \times \sqrt{4} = 1.192$$

$$N(1.592) = 0.944, \quad N(1.192) = 0.883$$

$$E_0 = 100 \times 0.944 - 70 \exp(-0.05 \times 4) \times 0.883 = 43.79$$

Intensity models

The default is a random variable tau (τ) that denotes the time in which the company will go in default. Default can be described as bankruptcy, lack of payment, and so on. To model the arrival risk of a credit event, we need to model an unknown random point in time τ.

Intensity-based models focus directly on describing the conditional probability of default without the definition of the exact default event. Intensity models are based on the concept of survival probability. It is an idea borrowed from actuarial and biological sciences. Survival probability is a decaying exponential function, which describes the probability of how long a firm or a country (sovereign) will survive. This can be expressed mathematically in the simplest terms as follows:

$$P(\mathrm{t}, T) = \exp\left[-\lambda(T - t)\right]$$

Equation 13

But we want to model default, not survival. In fact, we can compute the probability of default in some future period *[S,T]* as shown from the current time *t*, simply as the difference between two consecutive survival probabilities. The probability of default *PD* in that future period then can be calculated using the following formula:

$$PD(\text{t}, \text{S}, T) = \exp\left[-\lambda(\text{S} - t)\right] - \exp\left[-\lambda(T - t)\right]$$

Equation 14

The preceding formulas are fundamental for the modeling of credit risk. With them, we can describe the likelihood of default, which is a probabilistic event that will happen in the future. The cashflows associated with this future event are thus contingent on the occurrence of this event. Following the classical framework of financial derivatives, thus credit derivatives can be understood as contingent claims, in which the event that triggers the future cash flow is default.

To be concrete, consider the simplified situation—we are expecting to receive a single cash flow from a company at some future time *T*. What is the present value of this cash flow?

If we don't consider credit risk, the **Present Value** (**PV**) of the cash flow is simply the discounted **Future Value** (**FV**) of the cash flow. If the discount factor in the intervening time is DF, then we can write the simplest formula in finance, which states the following:

$$PV = DF \times FV$$

We can go one step further and assuming continuous compounding with a constant risk-free rate *r*, we can obtain the following equation:

$$PV = \exp(-rT) \times FV$$

Equation 15

What happens now if we consider the credit risk of the company? The first thing to notice is that the cash flow is not any more certain, but it is uncertain. In mathematical terms, it is not any more deterministic, but it is probabilistic. Therefore, we ought to rewrite the previous equation. But because the future cash flow is unknown, it is in fact a random variable. We ought to write it as an expected value of this random variable as follows:

$$PV = \exp(-rT) \times E\left[FV\right]$$

This expectation thus introduces a probability of FV happening or not. But what is this probability? The probability that the company will be there in the future to make the payment. In other words, the probability that the company has survived until then, in other words its survival probability. So we can substitute $E[FV]$ with $FV \times P(0,T)$ in order to obtain the following formula:

$$PV = \exp(-rT) \times FV \times P(0,T)$$

Furthermore, we can explicitly represent the survival probability described previously as follows:

$$PV = \exp(-rT) \times FV \times \exp(-\lambda T)$$

We will receive the future cash flow only if the company is present then in the future (or has survived) to do it. The final expression for the present value of the cashflow, taking into account credit risk, is as follows:

$$PV = \exp(-(r+\lambda)T) \times FV$$

Equation 16

This last expression is interesting as it shows that credit risk is a form of "spread" that is added to the risk free rate that we normally use to discount future cashflows.

For example, imagine that we expect to receive 1 million USD from a counterparty that has a credit risk, quantified by its hazard rate, of $\lambda = 3\%$. Assuming that the risk-free interest rate $r = 5\%$, compute the present value of the cash flow firstly, without credit risk and secondly, with credit risk. Assume continuous time discounting. Assume that the recovery rate is $R = 100\%$. How shall we proceed?

By applying the preceding formulas, we obtain that without credit risk, the PV is as follows:

$$PV = \exp(-rT) \times FV$$

$$PV = \exp(-(0.05)(1)) \times (1,000,000) = 950,000$$

Now, if we consider the credit risk, we have the PV as follows:

$$PV = \exp\left(-(r+\lambda)T\right) \times FV$$

$$PV = \exp\left(-(0.05 + 0.03)(1)\right) \times (1,000,000) = 880,000$$

Note that the second cash flow is smaller than the first. The effect of the credit risk is to view the value of the cash flow today (its PV) as reduced because not being certain, it has to be multiplied by the chance that it happens (that is, its survival probability).

In the previous analysis, we have assumed that when default occurs, all of the future cash flow FV will be lost. However, in most real situations, some fraction of money is recovered. If we define this fraction as the recovery rate R, the present value PV of the risky future cash flow is still the same, as follows:

$$PV = \exp\left(-rT\right) \times E\left[FV\right]$$

But the expectation is now resolved in terms of two possible states, default or no default, and each of them is multiplied by its respective probabilities, as follows:

$$E\left[FV\right] = \underbrace{FV \times P(0,T)}_{NO\ DEFAULT} + \underbrace{FV \times R \times (1 - P(0,T))}_{DEFAULT}$$

In the preceding equation, we have taken into account the recovery rate R in the case of default. The present value of the risky cash flow PV is now as follows:

$$PV = \exp\left(-rT\right) \times FV \times \left(P(0,T) + R\left(1 - P(0,T)\right)\right)$$

Summary

This chapter gave an overview of the fundamental models used to price derivatives in the modern financial markets. We will now take these models as basis to model the underlying of the various asset classes we reviewed and apply a number of numerical methods to implement their calculations efficiently in a computer.

In the next chapter, we will concentrate on numerical methods.

3
Numerical Methods

In the previous chapter, we reviewed some of the key mathematical models used to describe the behavior of the underlying assets of financial derivatives. We saw, in particular, how these models are used to describe the future behavior of these assets based on the information we have today. These models are generally expressed in terms of SDEs and **Partial Differential Equations (PDEs)**.

In this chapter, we are going to describe the three main numerical methods used in the financial markets today in the context of financial derivatives. They are a way to use actual numerical values to the abstract mathematical formulas we saw in the previous chapter. These numerical methods are as follows:

- **Monte Carlo** (**MC**) simulation
- **Binomial Trees** (**BT**)
- **Finite Difference Methods** (**FDM**)

In the context of the Bento Box template, this chapter corresponds to box 3—numerical methods. There is a fourth family of methods, less frequently used, called **quadrature methods**, which are used for numerical integration. These will not be discussed here.

The Monte Carlo simulation method

Monte Carlo simulation is named after the famous casino in the principality of Monaco. It is the most widely used numerical method to price financial derivatives in the industry because of its simplicity, flexibility, and extensibility.

The basic idea of the method is to construct a simulation engine that will allow us to predict a number of possible ways (or trajectories) in which the underlying assets can evolve in the future. These trajectories can be thought of as potential economic or financial scenarios. With MC simulation, we attempt to answer questions such as "given the observed price of Vodafone stock today, what could be the likely prices of the stock each day for the next month?"

As we cannot be certain of the future evolution of prices, our result needs to be based on probability, and, thus, we need large number of samples. Using the stochastic models that we saw in the previous chapter to simulate one possible trajectory, with MC simulation, we are going to simulate many possible trajectories and, for each, compute the payoff that the contract would have had if the prices had followed that specific path in future. Afterwards, we are going to take all these possible payoffs and compute their expected value, that is, the mean or average value. This will give us an estimate of how much this contract will be worth in the future.

MC simulation then allows us to compute the fair price of a financial derivative as its expected discounted payoff. This concept stems from the financial principle of fair pricing, which states that the price that a contract should have if the sum of the cash flows that we expect to receive are the same as the sum of the cash flows that we expect to pay. For more details on MC simulation, you are invited to refer to *Monte Carlo Methods in Financial Engineering*.

In order to have an intuition of why this is the case, consider the following simple example:

Imagine that you have bought a plain vanilla European Call option contract at time *t=0*. This contract will give you a payoff of $H(S_T) = \max(S_T - K, 0)$ at maturity *t=T*. Because the value of the underlying at maturity is uncertain, that is, *S_T* is a random variable, the payoff function *H(S_T)* is also uncertain. We can write that the expected value of the payoff function $H(S_T)$ is the expectation $E[H(S_T)]$. In addition, in a European Call contract, we pay a premium today in order to have the right to exercise the option or not at maturity *t=T*. How much should we pay for the premium for this contract today?

As we said before, in a fair value setting, what we expect to receive should be equal to what we expect to pay. By putting all these cashflows together (positive indicating to be received, negative to be paid) we can write the following:

- Amount paid at *t=0* is written as $-\pi$
- Amount to be received at *t=T* is written as $+E\big[H(S_T)\big]$

If we now compute the present value of these cash flows, we get the following equation:

$$-\pi + DF_T \times E\big[H(S_T)\big] = 0$$

In other words, it can be summarized as follows:

$$\pi = DF_T \times E\big[H(S_T)\big]$$

The object of the MC simulation method is precisely to help us compute the expectation of the payoff $E[H(S_T)]$; once you compute this, discount this value to obtain the premium of the derivative.

This same idea can be generalized to more complex settings with many complex payoffs and underlyings.

Algorithm of the MC method

For European-type derivatives, MC simulation is composed of the following three steps:

1. The first step is to generate trajectories.

    Simulate *M* trajectories for the evolution of the underlying from *t=0* to maturity *t=T*. In this step, we use the discretized version of SDE that describes the evolution of the underlying. In our case, we use GBM as SDE, which will allow us to take the value of the stock from its current value *S_0* to the value at maturity *t=T*. A discretized version is essentially an approximate version applicable to finite time steps rather than continuous time steps. For more details, please refer to *Monte Carlo Methods in Financial Engineering*. We discretize the life of the option contract in *N* steps, each of size *dt*, which can be succinctly written as follows:

    $$\big\{ S_i^j \big\} \quad i = 0...N \quad j = 1...M$$

At the end of this step, we should have a vector of *M* values for *S_T*, as follows:

$$\left\{ S_T^i \right\} \quad i = 1...M$$

These represent a set of possible scenarios for the value of the underlying *S* at time *t=T*. We use GBM to generate multiple paths that will serve a prediction of where the value of *S_T* will be at maturity.

2. The next step is to compute the expectation.

    Once we have the set of values of the underlying at maturity, we now need to compute the expectation of the payoff at maturity. So we take each of these values and compute the payoff for each value as follows:

$$H\left(S_T^i \right) \quad i = 1...M$$

The preceding equation will give us a vector of payoffs. In order to compute the expectation, we need to simply take the average of the payoffs as follows:

$$E\left[H\left(S_T^i \right) \right] = \frac{1}{M} \sum_{i=1}^{M} H\left(S_T^i \right)$$

3. Now discount the expectation to the present.

    The final step is to discount the value of the payoff from maturity to the present time. In order to do this, we will use the following formula:

$$\pi = DF_T \times E\left[H(S_T) \right]$$

Alternatively, we can also use continuous compounding, as follows:

$$\pi = \exp(-rT) \times E\left[H(S_T) \right]$$

The preceding equation will give us the value of the derivative π. Note that in this case, we have assumed that there is no correlation between the interest rates and the price of stock. That is why we can neatly separate the two effects in the preceding equation. If the interest rates and the price of stock were correlated, then we will not be able to separate the discount factor and the expectation. This no-correlation assumption is standard for simple pricing models.

Example of the MC method

Consider the example where we would like to price a six-month European Call option on Vodafone equity (VOD.L). The current equity price of Vodafone is £100.00, with a volatility of 20 percent and a strike of £100. We assume that the stock pays no dividends. The current risk-free rate is 5 percent pa. How do we proceed to solve this problem using MC simulation? We proceed using the following three phases:

1. The first step is to generate trajectories.

    We apply GBM to simulate the value of VOD.L stock from the spot price today *S_0 = £100.00*. For simplicity, we choose to discretize the life of the option from *t=[0,T]* into *N=5* time steps and to do *M=5* simulations using GBM in discrete terms, as follows:

    $$S_{i+1} = S_i \left(1 + r\Delta t + \sigma \varepsilon_i \sqrt{\Delta t} \right)$$

    The five trajectories will thus be as follows:

    $$S_0^1 \to S_1^1 \to S_2^1 \to S_3^1 \to S_4^1 \to S_5^1$$

    $$S_0^2 \to S_1^2 \to S_2^2 \to S_3^2 \to S_4^2 \to S_5^2$$

    $$S_0^3 \to S_1^3 \to S_2^3 \to S_3^3 \to S_4^3 \to S_5^3$$

    $$S_0^4 \to S_1^4 \to S_2^4 \to S_3^4 \to S_4^4 \to S_5^4$$

    $$S_0^5 \to S_1^5 \to S_2^5 \to S_3^5 \to S_4^5 \to S_5^5$$

    The prices of the stock at maturity will be as follows:

    $$\left\{ S_4^1 = 103.37, S_4^2 = 94.40, S_4^3 = 105.27, S_4^4 = 108.24, S_4^5 = 117.73 \right\}$$

2. The next step is to compute the expectation.

    For each of the values of the underlyings, we now compute the payoffs as follows:

    $$\left\{ H(S_5^1), H(S_5^2), H(S_5^3), H(S_5^4), H(S_5^5) \right\}$$

We now use the specific form of the payoff to describe a European Call option as follows:

$$\left\{\max(S_4^1 - K, 0), \max(S_4^2 - K, 0), \max(S_4^3 - K, 0), \max(S_4^4 - K, 0), \max(S_4^5 - K, 0)\right\}$$

We apply the following numbers to the preceding equation to get the following result:

$$\{3.37, 0.00, 5.27, 8.24, 17.73\}$$

The expected value of the preceding calculation is as follows:

$$E\left[H\left(S_T^i\right)\right] = \frac{1}{5}(3.37 + 0.00 + 5.27 + 8.24 + 17.73) = 6.92$$

3. Now discount the expectation to the present.

We now use the following continuous compounding to discount the expected payoff we just calculated in step 2 in order to determine the value of the premium:

$$\pi = \exp(-rT) \times E[H(S_T)] = \exp(-(0.05)(0.5)) \times (6.92) = 6.75$$

In this example, we have used only five scenarios for our MC price. In practice, hundreds or even thousands of scenarios are required in order to obtain an acceptable error. Clearly, the more scenarios you use, the more accurate the approximation. It is possible to derive some error-bound formulas for the MC method and show the speed of convergence. For more details, the reader is invited to refer to *Monte Carlo Methods in Financial Engineering*. Putting together all the trajectories for the five MC scenarios, we obtain the table shown in the following screenshot. Here, we see that all the trajectories start at *S0=100* and lead to some final value *S5*. For each trajectory, we compute the payoff *H*, which is then averaged to compute its expected value. The result is then discounted to obtain the present value of the derivative.

S0	S1	S2	S3	S4	S5	H(S_T)
100,00	101,06	107,10	115,81	113,49	103,37	3,37
	0,0884	0,8656	1,2070	-0,3961	-1,4890	
100,00	105,81	108,18	101,43	103,05	94,40	0,00
	0,8390	0,2757	-1,0661	0,1735	-1,4057	
100,00	100,17	102,71	99,83	105,65	105,27	5,27
	-0,0515	0,3215	-0,5230	0,8436	-0,1360	
100,00	108,82	113,29	112,32	117,29	108,24	8,24
	1,3157	0,5703	-0,2142	0,6198	-1,2992	
100,00	103,79	102,30	101,87	112,90	117,73	17,73
	0,5200	-0,3055	-0,1464	1,6339	0,5973	
6,75	←				mean	6,92

Example of the Monte Carlo simulation

The Binomial Trees method

Binomial Trees (**BT**) can be traced to the work of (Cox, Ross, and Rubinstein 1979). Like MC methods, they are based on the idea of how the discretization of stock prices can jump up or down. Unlike the MC methods, BT are not based on simulation of many possible paths, but on the construction of a single path of possible future prices that bifurcates at every node. These prices, as well as their associated probabilities, constitute the tree. Once this tree is built, the prices of the underlying at maturity can be determined, and the the payoff at maturity can be then computed and discounted to the present time in order to determine the premium of the derivative.

Algorithm of the BT method

The BT method when applied to price derivatives is composed of three phases: the construction of the tree of prices (forward phase), the computation of the payoffs (maturity phase), and the discounting of the payoffs to the present time (backward phase). We will now explain the BT method in the simplified context of a two-step BT. This can be easily generalized to an *N* step tree.

To start with, we assume that the underlying can only go up or down in the next time step. So we specify the up factor *u* to describe how much the value today changes to a higher value and the down factor *d* to describe how much the value today changes to a lower value, such that the up value is *S(T)= u S(0)*, and the down value is *S(T)= d S(0)*. Furthermore, refer to "Option pricing: A simplified approach". The formula for the up and down values can be shown as follows:

$$u = \exp(\sigma\sqrt{\Delta t})$$
$$d = \exp(-\sigma\sqrt{\Delta t})$$

The following is the formula for the probabilities of going up:

$$p = \frac{\exp(r\Delta t) - d}{u - d}$$

The probability of going down is $1 - p$. We can now proceed to construct our binomial tree in the following three phases:

1. The first phase is the forward phase.

   Here we construct the tree. Like in MC simulation, time is discretized in steps *dt* from *t=0* to *t=T*. From one step *tp*, the next price of the underlying can either go up or down by a factor *u* or *d* as shown in the following formulas:

   $$\text{At } t = 0 : S_0$$

   $$\text{At } t = t_1 : uS_0 \text{ or } dS_0$$

   $$\text{At } t = t_2 : u^2 S_0, udS_0, d^2 S_0$$

   Thus, the values of the tree at maturity are as follows:

   $$S_T^1 = u^2 S_0, S_T^2 = udS_0, S_T^3 = d^2 S_0$$

   In general case, we proceed in a similar manner until we arrive at the maturity *T*, and we have *N+1* values for the variable *S*. We will calculate these values with the help of the following equation:

   $$\left\{ S_T^k \right\} \quad k = 1...N + 1$$

The preceding equation in our case can be summarized as follows:

$$\left\{ S_T^1, S_T^2, S_T^3 \right\}$$

This entire process is illustrated in the following diagram:
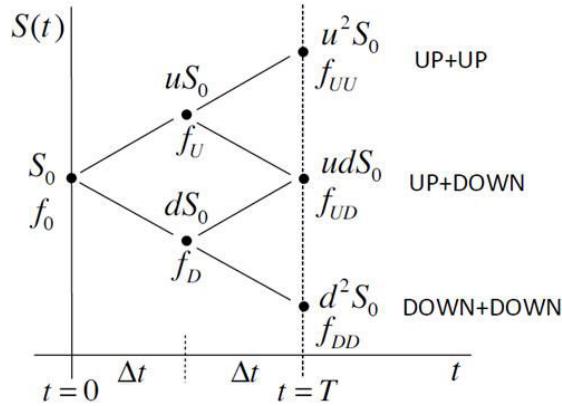
2. The second phase is the payoff phase.

   In this phase, we use the values of the underlying at maturity and for each of them, we compute the value of the payoff, as follows:

   $$\left\{ H\left(S_T^k \right) \right\} \quad k = 1...N+1$$

   In our case, the equation can be summarized as follows:

   $$\left\{ H(S_T^1), H(S_T^2), H(S_T^3) \right\}$$

   The following in turn are the values of the option at maturity T:

   $$\left\{ V_T^k = H\left(S_T^k \right) \right\} \quad k = 1...N+1$$

   In our case, the preceding equation can be summarized as follows:

   $$\left\{ V_T^1 = H(S_T^1), V_T^2 = H(S_T^2), V_T^3 = H(S_T^3) \right\}$$

3. The third phase is the backward phase.

   In this final phase, we take the values of the payoff at maturity and proceed in a backward manner. We move from the last node to the previous nodes, by computing the option value as the discounted expected payoff in the previous nodes using the weighted probabilities, as follows:

$$V_{T-1}^k = \exp(-r\Delta t)\left[pV_T^k + (1-p)V_T^k \right]$$

In our case, in the second step, the equations are as follows:

$$V_2^1 = \exp(-r\Delta t)\left[pV_3^1 + (1-p)V_3^2 \right]$$

$$V_2^2 = \exp(-r\Delta t)\left[pV_3^2 + (1-p)V_3^3 \right]$$

And, in the first step, the equation is as follows:

$$V_1^1 = \exp(-r\Delta t)\left[pV_2^1 + (1-p)V_2^2 \right]$$

The premium of the derivative, the option price, is the value $\pi = V_1^1$.

Example of the BT method

Consider the example where we would like to price a six-month European Call option on Rolls Royce equity (RR.L). The current equity price of the stock is £100.00, with a volatility of 30 percent p.a. and a strike of £90. We assume the stock pays no dividends. The current risk-free rate is 5 percent pa. How do we proceed to solve this problem using BT?

To start with, we divide the life of the option in two steps, thus *dt=0.25*. The tables in the following screenshot illustrate the numerical values for each of the three steps applied to this problem:

STEP 1				STEP 2				STEP 3			
			134,99				44,99				44,99
		116,18								27,30	
S0	100,00		100,00				10,00	V0	16,04		10,00
		86,07								4,98	
			74,08				0,00				0,00
			ST				H(ST)				
TIME	0,00	0,25	0,50	TIME	0,00	0,25	0,50	TIME	0,00	0,25	0,50

Example of Binomial Trees pricing.

We first compute the up and down factors as well as the up probability p. In numerical terms, these are calculated using the following equations:

$$u = \exp(\sigma\sqrt{\Delta t}) = \exp(0.30\sqrt{0.25}) = 1.16$$
$$d = \exp(-\sigma\sqrt{\Delta t}) = \exp(-0.30\sqrt{0.25}) = 0.86$$

The following are the probabilities of going up and down respectively:

$$p = \frac{\exp(r\Delta t) - d}{u - d} = \frac{\exp(0.05 \times 0.25) - 0.86}{1.16 - 0.86} = 0.50$$

$$(1 - p) = 0.50$$

With all these parameters, we can now proceed to construct our tree in three phases, as follows:

1. The first phase is the forward phase.

   We can now construct the two levels of the tree as follows.

   $$A \ t = 0 : S_0 = 100$$

   $$\text{At } t = t_1 : uS_0 = 116,18 \text{ or } dS_0 = 86.07$$

   $$\text{At } t = t_2 : u^2 S_0 = 134.99, udS_0 = 100, d^2 S_0 = 74.08$$

   Thus, the values of the tree at maturity are as follows:

   $$\left\{ S_T^1 = 134.99, S_T^2 = 100, S_T^3 = 74.08 \right\}$$

2. The second phase is the payoff phase.

   In this phase, we use the values of the underlying at maturity, and for each of them, we compute the value of the payoff, as follows:

   $$\left\{ H\left(S_T^k \right) \right\} \quad k = 1...N + 1$$

   In our case, the equation can be summarized as follows:

   $$\left\{ H(S_T^1) = 44.99, H(S_T^2) = 10, H(S_T^3) = 0 \right\}$$

The following in turn are the values of the option at maturity *T*:

$$\left\{ V_T^k = H\left(S_T^k \right) \right\} \quad k = 1...N+1$$

In our case, the preceding equation can be summarized as follows:

$$\left\{ V_T^1 = 44.99, V_T^2 = 10, V_T^3 = 0 \right\}$$

3. The third phase is the backward phase.

    In this final phase, we take the values of the payoff at maturity and proceed in a backward manner. We move from the last node to the previous nodes, by computing the option value as discounted expected payoff in the previous nodes using the weighted probabilities, as follows:

    $$V_{T-1}^k = \exp(-r\Delta t)\left[pV_T^k + (1-p)V_T^k \right]$$

    In our case, in the second step, the equations are as follows:

    $$V_2^1 = \exp(-r\Delta t)\left[pV_3^1 + (1-p)V_3^2 \right] = 27.30$$

    $$V_2^2 = \exp(-r\Delta t)\left[pV_3^2 + (1-p)V_3^3 \right] = 4.98$$

    And in the first step, the equation is as follows:

    $$V_1^1 = \exp(-r\Delta t)\left[pV_2^1 + (1-p)V_2^2 \right] = 16.04$$

The Finite Difference method

The Finite Difference (FD) method is a numerical technique that focuses directly on the approximate solution of a differential equation. As shown by (Black and Scholes 1973) for equity financial derivatives (contingent claims), the problem is expressed in terms of a **Partial Differential Equation** (**PDE**).

The basic idea of FDM is to discretize a differential equation. The method transforms the derivatives in the differential equation into quantities or ratios that approximate the derivatives. These quantities are not any more infinitesimal but finite, that is, they have a finite length. This is the origin of the name of finite differences. For more details, the reader can refer to *The Mathematics of Financial Derivatives: A Student Introduction*.

Consider the following illustration where a continuous function *f(X)* and the first derivative of the function is defined as follows:

$$f(x) = \frac{df}{dx} \approx \frac{\Delta f}{\Delta x} = \frac{f_{i+i} - f_i}{x_{i+i} - x_i} = \frac{f_{i+i} - f_i}{\Delta x}$$

The preceding function is also known as the **slope,** which is the ratio between the growth (or decrease) in the function with respect to the step size *dx*. Using the preceding finite difference allows us to calculate the slope of the *f(x)* function in terms of algebraic quantities.

In quantitative finance, we encounter various types of PDEs. The most important is the Black-Scholes PDE, which is expressed as follows:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

We now consider solving this equation in a rectangular domain in the *S* and *t* axes. In the *S* axis, the domain is [a,b]. In the *t* axis the domain is [0,T]. This can be written mathematically as the domain $\Omega = \{[a,b] \times [0,\mathrm{T}]\}$. In the case of a European Call, it has a final condition as follows:

$$V(S,T) = \max(S - K, 0)$$

And the following are the boundary conditions:

$V(\mathrm{a,t}) = 0$ and $V(\mathrm{b,t}) = S$.

Rather than solving the Black-Scholes PDE directly (that is, using variables *S* and *t*) we will be following (Wilmott et al. 1995), and we are going to propose a change of variables. This will transform the original PDE into an equivalent PDE, which is easier to solve and in fact is the classical equation of heat diffusion. The change of variables is as follows:

$$S = K \exp(x)$$

$$t = T - \tau / (\tfrac{1}{2}\sigma^2)$$

The preceding equations transform the Black-Scholes PDE into the classical equation of heat diffusion, as follows:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} \quad -\infty < x < \infty, \ \tau > 0$$

And the European Call payoff is transformed into the following equation:

$$u(x,0) = \max\left(\exp(\tfrac{1}{2}(k+1)x - \exp(\tfrac{1}{2}(k-1)x, 0\right)$$

where the parameter *k* is: $k = r / (\tfrac{1}{2}\sigma^2)$.

Algorithm of FDM

The application of FDM to the preceding PDE requires the first derivative with respect to time and the second derivative with respect to *x*, which leads to the following equations:

$$\frac{\partial u}{\partial \tau} \approx \frac{\Delta u}{\Delta \tau} = \frac{u_{i,j+1} - u_{i,j}}{\Delta \tau}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{\Delta}{\Delta x}\left(\frac{\Delta u}{\Delta x}\right) = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

The preceding approximations can be derived from a Taylor series expansion. See (Wilmott et al. 1995) as we did in the preceding section.

In order to do this, we need to discretize the domain of the function to a discrete set of nodes. In the case of the BS equation, there will be *N* division's (or *N+1*) nodes in the spatial dimension and *M* division's (or *M+1*) nodes in the temporal dimension.

If we now put together our previous approximations, we will obtain the following formula:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta \tau} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

Solving for the term on the LHS of the preceding equation, we finally obtain the following discretized version of the PDE:

$$u_{i,j+1} = \alpha u_{i+1,j} + (1 - 2\alpha)u_{i,j} + \alpha u_{i-1,j}$$

Where in the preceding equation $\alpha = \frac{\Delta \tau}{(\Delta x)^2}$.

The discretized version of the PDE can be solved iteratively in time, using the **explicit** or **forward** finite difference method (FDM) as it's the simplest possible implementation of finite difference techniques for pricing options. We are now ready to follow the next phases to apply the FDM, which are as follows:

1. First, discretize the domain.

   Perform this step both in space and time dimensions with time steps $\Delta \tau, \Delta x$.

2. Now approximate each of the derivatives with finite differences.

   Just as we have shown in the preceding section, we will apply the principle of transforming the continuous derivatives of the PDE into a finite approximation. This finite approximation will lead to algebraic equations. In literature, this set of equations is called a **stencil**.

3. Next collocate the stencil to all the nodes of the domain.

   We now apply the stencil to all the nodes in the domain with the exception of the nodes that represent the initial and boundary conditions. For these nodes, we know the value is a priori, and, hence, it does not need to be computed.

4. Iterate the solution in time with the stencil until we cover the full domain.

    In explicit FDM, you simply advance and compute the values for the unknown function u. Note that in other forms of FDM (such as implicit FDM), we need to solve a system of equations via a matrix problem. Please refer to (Wilmott et al. 1995) for further details on implicit methods.

Example of the FD method

Consider the example where we would like to price a six-month European Call option on Barclays equity (BARC.L). The current equity price of BARC is £75, with a volatility of 30 percent p.a. and a strike of £75. We assume the stock pays no dividends. The current risk-free rate is 5 percent pa. How do we proceed to solve this problem using the FDM?

We know that equity financial derivatives satisfy the Black-Scholes PDE when the stock is modelled using GBM. So we solve the heat diffusion equation we described in the previous section. As we did earlier, we apply the following four phases to solve our FDM problem:

1. First discretize the domain.

    We divide the domain into N space divisions dS and M time divisions dt, thus $N=5$ and $M=4$. We first apply these values both in space and time dimensions with time steps $\Delta\tau, \Delta x$.

    Thus, we obtain six points in time as follows:

    $$\tau = \{0.0225, 0.0180, 0.0135, 0.0090, 0.0045, 0.0000\}$$

    Five points in space is shown as follows:

    $$x = \{-1, -0.5, 0.0, +0.5, +1\}$$

2. Now approximate each of the derivatives with finite differences as follows.

$$u_{i,j+1} = \alpha u_{i+1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i-1,j}$$

In the preceding equation, $\alpha = 0.018$.

3. Collocate the stencil to all the nodes of the domain.
The following is the initial condition:

$$u(x,0) = \max\left(\exp(\tfrac{1}{2}(k+1)x - \exp(\tfrac{1}{2}(k-1)x, 0\right)$$

Alternatively, the following is the condition with numerical values:

$$u(x,0) = \{0.00, 0.00, 0.00, 0.67, 1.82\}$$

In the preceding equation, $k = 1.11$.
The following is the final boundary condition:
$u(-1,\tau) = 0$ and $u(+1,\tau) = 1.82$.

4. Iterate the solution in time with the stencil until we cover the full domain.

The following are the internal nodes:

$$u_{i,j+1} = \alpha u_{i+1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i-1,j}$$

$$i = 2,3,4; j = 1 : u_{i,j+1} = \alpha u_{i+1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i-1,j}$$

$$i = 2,3,4; j = 2 : u_{i,j+1} = \alpha u_{i+1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i-1,j}$$

$$i = 2,3,4; j = 3 : u_{i,j+1} = \alpha u_{i+1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i-1,j}$$

We can arrange the numerical results from our algorithm as shown in the table in the following screenshot, using the transformed variables (upper table) or the original variables (lower table), where we can find that for *S=75* and *t=0*, the option price is *£4,20*:

x/tau					
	-1	-0,5	0	0,5	1
0,0225	0,00	0,00	0,06	0,71	1,82
0,0180	0,00	0,00	0,05	0,70	1,82
0,0135	0,00	0,00	0,04	0,69	1,82
0,0090	0,00	0,00	0,02	0,68	1,82
0,0045	0,00	0,00	0,01	0,68	1,82
0,0000	0,00	0,00	0,00	0,67	1,82
S/t					
	27,59	45,49	75,00	123,65	203,87
0,0	0,00	0,15	4,20	50,49	125,93
0,1	0,00	0,09	3,41	50,14	126,56
0,2	0,00	0,05	2,60	49,79	127,20
0,3	0,00	0,02	1,76	49,42	127,84
0,4	0,00	0,00	0,90	49,04	128,48
0,5	0,00	0,00	0,00	48,65	128,87

Example of Finite Difference pricing.

Summary

In this chapter, we reviewed the basics of the three key numerical methods used to price financial derivatives today. For each of them, we have provided an algorithm and a numerical example. Further, more advanced features of these methods can be found in excellent textbooks by (Glasserman 2003), (Kloeden and Platen 1992), and (Wilmott et al. 1995) as mentioned in all the previously discussed sections.

Not all methods are applicable in all situations, just like the tools in a toolbox. Some methods are more effective to solve some specific problems. For example, with a binomial tree, it is simple to evaluate American options also, while for Monte Carlo, it is not so straightforward. Monte Carlo is more powerful in high-dimensional problems, while finite differences can be used effectively for low-dimensional problems.

4

Equity Derivatives in C++

In the previous two chapters, we described the key mathematical models used to simulate the behavior of the underlying assets of financial derivatives (*Chapter 2, Mathematical Models*) and the main numerical methods used to price them (*Chapter 3, Numerical Methods*).

In this chapter, we apply these ingredients to the pricing of equity derivatives. We consider two examples: the pricing of a plain vanilla European Call option (basic example) and the pricing of an equity basket on the maximum of two assets (advanced example). We provide the full working C++ implementation for both. Note that if you are new to OOP, it is suggested that you first study the implementation in C followed by the implementation in C++, available in the code bundle of the chapter.

Basic example – European Call

In this first example, we consider the pricing of a plain vanilla European Call option. This example is exceedingly simple but crucial; it will serve as the building block for the rest of the option pricing problems to be solved with the Monte Carlo simulation.

The full characteristics of the contract, the choice of the mathematical model, and its numerical method are shown below in the Bento Box template:
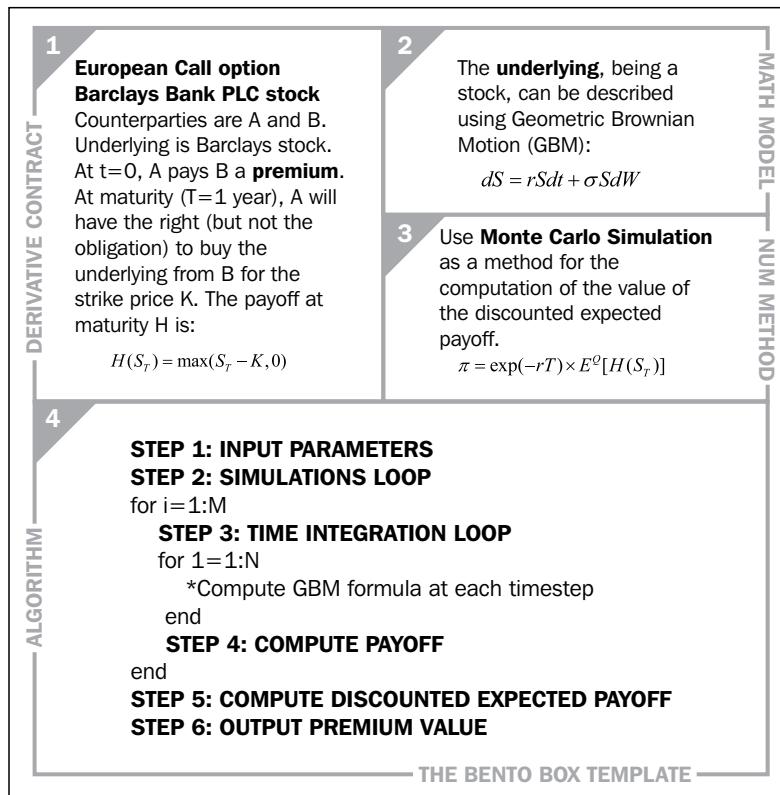
Bento Box template for basic example: European Call

Our objective is to calculate the premium of this financial derivative. We proceed by completing the contents of the Bento Box in clockwise sense, starting from the top-left corner. We first fill all the data of the contract, in particular the payoff function, which for a simple European Call is as follows:

$$H(S_T) = \max(S_T - K, 0)$$

Secondly, we ought to select the mathematical model for the underlying, which in the case of equities is GBM. Third, we select the numerical method to be used as Monte Carlo simulation. Fourth, we construct the algorithm that will integrate these calculations as a series of calculation steps, which will serve us as blueprint for implementing it in C++.

The algorithm is shown in box 4 of the Bento Box template. The implementation of the algorithm in C++ is shown in code snippets 1, 2, and 3. Code snippet 1 (`maineq1.cpp`) is the pricing algorithm proper, while code snippets 2 and 3 are auxiliary functions. The algorithm is composed of six steps, which take us from the input parameters (**STEP 1**) to the output of the premium value (**STEP 6**).

An important feature of this algorithm is the function in code snippet 2 (`random.cpp`). This implements the Box-Muller method to obtain random samples from the standard normal (Gaussian) distribution that are required for the GBM. Code snippet 3 (`random.h`) is simply the header file of code snippet 2 (`random.cpp`). The Box-Muller method takes two independent samples from a uniform distribution and transforms them into a single sample from a Gaussian distribution; this value needs to be assigned to the variable epsilon in the code. Certainly, a more efficient implementation is possible. The Box-Muller method in fact transforms a couple of uniform variables into a couple of normal variables. It would be better to also use the second normal sample, generated in the process, in order to be computationally more effective. Please refer to the book website for details of this more efficient implementation and to the original paper for further details (*A Note on the Generation of Random Normal Deviates*).

As part of the input parameters, we ought to define *N* and *M*. Here *N* represents the number of time steps to be used in the GBM calculation, while *M* represents the number of Monte Carlo simulations to be used. In our example, we consider the pricing of a European Call option on Barclays stock (BARC.L), whose spot is £100, strike £100, risk-free interest rate 5 percent p.a., an annualized volatility of 10 percent, and a maturity of one year. We use *N=500* and *M=10,000*. In my computer, the option premium is £6.81 with an execution time of 1.34 seconds. The value of the premium and the execution time will vary from computer to computer.

Note that this code can be easily modified to price other payoffs by simply changing **STEP 4** in the algorithm. In terms of a C++ implementation, this concept can be incorporated using a class to define the payoff. Also, **STEP 4** can be slightly modified to include an estimate of the accuracy in the Monte Carlo approximation. Please refer to the website for downloadable implementations containing these features. An excellent textbook describing this example is *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*.

Code 1 – EQ1 – Monte Carlo European Call

The following is the code snippet for `EQ1_main.cpp` file:

```cpp
// maineq1.cpp
// requires random.cpp
#include "random.h"
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;
int main()
{
  cout << "\n *** START EQ1: Monte Carlo European Call *** \n";
  // STEP 1: INPUT PARAMETERS
  double T=1; // maturity
  double K=100; // strike
  double S0=100; // spot
  double sigma=0.10; // volatility
  double r=0.05; // interest rate
  int N=500; // number of steps
  int M=10000; // number of simulations
  double S[N+1];
  double sumpayoff=0;
  double premium=0;
  double dt = T / N;

  // STEP 2: MAIN SIMULATION LOOP
  for (int j=0; j < M; j++)
  {
    S[0]=S0; // initialize each path for simulation

    // STEP 3: TIME INTEGRATION LOOP
    for (int i=0; i < N; i++)
    {
      double epsilon = SampleBoxMuller();  // get Gaussian draw
      S[i+1] = S[i]*(1+r*dt+sigma*sqrt(dt)*epsilon);
    }

    // STEP 4: COMPUTE PAYOFF
    sumpayoff += max(S[N]-K,0.0); // compute and ad payoff
  }

  // STEP 5: COMPUTE DISCOUNTED EXPECTED PAYOFF
```

```
    premium =  exp(-r*T)*(sumpayoff / M);

    // STEP 6: OUTPUT RESULTS
    cout <<"premium =  " << premium << "\n";
    cout << "\n *** END EQ1: Monte Carlo single asset *** \n";

    return 0;
}
```

Code 2 – random.cpp file

The following is the code snippet for random.cpp file:

```
// random.cpp
// Computing Gaussian deviates using Box-Muller method

#include "Random.h"
#include <cstdlib>
#include <cmath>
using namespace std;

double SampleBoxMuller()
{
  double result;
  double x;
  double y;

  double xysquare;
  do
  {
    x = 2.0*rand()/static_cast<double>(RAND_MAX)-1;
    y = 2.0*rand()/static_cast<double>(RAND_MAX)-1;
    xysquare = x*x + y*y;
  }
  while
  ( xysquare >= 1.0);
  result = x*sqrt(-2*log(xysquare)/xysquare);
  return result;
}
```

Code 3 – random.h header file

The following is the code for `random.h` file:

```
// random.h
double SampleBoxMuller();
```

After compiling and running the code, you should obtain the following screenshot:
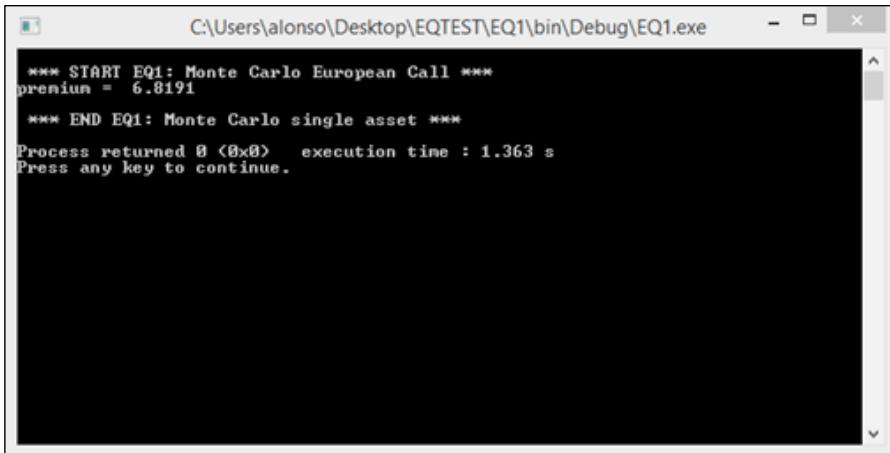
Advanced example – equity basket

In this second, more advanced example, we will illustrate the implementation of the pricing of an equity basket option, with the payoff being the largest between two asset values at maturity. The details of the approach are shown in the following Bento Box template:

1

Equity Basket option on the maximum of two assets. Counterparties are A and B. Underlying is Barclays stock and Rolls Royce stock. At t=0, A pays B a **premium**. At maturity (T=1 year), A will receive from B the payoff H: i

$$H(S1_T, S2_T) = \max(S1_T, S2_T)$$

DERIVATIVE CONTRACT

2

The **underlyings**, being a stocks, can be described using GBM process each:

$$dS1 = rS1dt + \sigma_1 S1 dW_1$$
$$dS2 = rS2dt + \sigma_2 S2 dW_2$$

MATH MODEL

3

Use **Monte Carlo Simulation** as a method for the computation of the value of the discounted expected payoff.

$$\pi = \exp(-rT) \times E^Q[H(S1_T, S2_T)]$$

NUM METHOD

4

STEP 1: INPUT PARAMETERS
STEP 2: SIMULATIONS LOOP
for i=1:M
   STEP 3: TIME INTEGRATION LOOP
   for 1=1:N
      *Compute GBM for asset 1 at each timestep
      *Compute GBM for asset 2 at each timestep
    end
   STEP 4: COMPUTE BASKET PAYOFF
end
STEP 5: COMPUTE DISCOUNTED EXPECTED BASKET PAYOFF
STEP 6: OUTPUT BASKET PREMIUM VALUE

ALGORITHM

— THE BENTO BOX TEMPLATE —

Bento Box template for advanced example: equity basket

Our aim is to compute the option premium as before.

The details of the contract are in Box 1, particularly the payoff function is as follows:

$$H(S1_T, S2_T) = \max(S1_T, S2_T)$$

Note that this being a basket option with two assets, we will now need two GBM processes to describe the evolution of the underlying. This is reflected in **STEP 3** that has been updated in the algorithm in box 4. We can then use the same MC numerical method to compute the expectation of the payoffs.

The C++ implementation of this algorithm can be found in code snippet 4. There are only two slight differences with Code 1: first, regarding the input parameters (**STEP 1**) and second regarding the calculation of the GBM (**STEP 4**). We now need to specify the parameters for both processes, including their spot prices and volatilities. As we need to compute two correlated stochastic processes, the two Gaussian samples that are required are now computed as follows:

$$\varepsilon_1 = \overline{\varepsilon}_1$$
$$\varepsilon_1 = \rho\overline{\varepsilon}_1 + \sqrt{1 - \rho^2 \overline{\varepsilon}_2}$$

In the preceding equation, $\overline{\varepsilon}_1, \overline{\varepsilon}_2$ are two independent samples from the Gaussian distribution, while $\varepsilon_1, \varepsilon_2$ are the two correlated samples that incorporate the effect of the correlation ρ. Epsilon_1 and epsilon_2 remain normal variables, as they have unitary variance and the expected value of their product $\varepsilon_1, \varepsilon_2$ is equal to ρ.

As we did earlier, we can easily modify the payoff in **STEP 5** and incorporate other more complicated payoffs.

For example, consider the price of the following basket option:

We have two assets Barclays PLC (BARC.L) and Rolls Royce (RR.L). We want to price an option that pays the maximum of the value of these two assets at maturity, which is year. The current spot price of Barclays is £120 and that of Rolls Royce is £100. Their annualized volatilities are 10 percent and 15 percent respectively. We choose to discretize time in 300 time steps and use 10,000 simulations. The premium for this option under these conditions is £120.48 with an execution time of 2.22 seconds.

For further details about basket equity derivatives, you are invited to consult *Paul Wilmott on Quantitative Finance, 2nd Edition*.

Code 4 – EQ2 – Monte Carlo equity basket

The following is the code snippet for `EQ2_main.cpp` file:

```
// maineq2.cpp
// requires random.cpp
#include "random.h"
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

int main()
```

```
{
  cout << "\n *** START EQ2: Monte Carlo equity basket *** \n";
  // STEP 1: INPUT PARAMETERS
  double T=1; // maturity
  double r=0.05; // interest rate
  double S10=120; // spot equity 1
  double S20=100; // spot equity 2
  double sigma1=0.10; // volatility
  double sigma2=0.15; // volatility
  double rho=0.5; // correlation
  int N=300; // number of steps
  int M=10000; // number of simulations
  double S1[N+1];
  double S2[N+1];
  double sumpayoff=0;
  double premium=0;
  double dt = T / N;

  // STEP 2: MAIN SIMULATION LOOP
  for (int j=0; j < M; j++)
  {
    S1[0]=S10;
    S2[0]=S20;
    // STEP 3: TIME INTEGRATION LOOP
    for (int i=0; i < N; i++)
    {
      double epsilon1 = SampleBoxMuller();
      double epsilon2 = SampleBoxMuller();
      S1[i+1] = S1[i]*(1+r*dt+sigma1*sqrt(dt)*epsilon1);
      epsilon2 = epsilon1*rho+sqrt(1-rho*rho)*epsilon2;
      S2[i+1]=S2[i]*(1+r*dt+sigma2*sqrt(dt)*epsilon2);
    }
    // STEP 4: TIME INTEGRATION LOOP
    sumpayoff += max(S1[N],S2[N]);
  }
  // STEP 5: COMPUTE DISCOUNTED EXPECTED PAYOFF
 premium =  exp(-r*T)*(sumpayoff / M);

  // STEP 6: OUTPUT RESULTS
  cout <<"premium =  " << premium << "\n";
  cout << "\n *** END EQ2: Monte Carlo equity basket *** \n";
  return 0;
}
```

After compiling and running the code, you should obtain the following screenshot:

Summary

We have solved two pricing problems in equity derivatives. We have seen a very simple example (what we called the basic) and a more complex one, which included an equity basket option. For each, we have provided the complete C++ implementation.

We will now proceed to the next asset class, foreign exchange derivatives, where we will also solve two problems, a simple and an advanced one, following the the Bento Box template approach in the next chapter.

5

Foreign Exchange Derivatives with C++

We now move to the world of currency or foreign exchange derivatives and how to price them using C++. We consider two examples: the pricing of a European Call option (basic example) and the pricing of an up-and-out barrier call option (advanced example). We provide the full-working C++ implementation for both. We follow the model for the evolution of foreign exchange currencies as found in "Foreign Currency Option Values". A simpler C implementation (without the OO features) can be found in the code bundle of this chapter. If you are new to OOP, it is recommended that you first study the implementation in C followed by the implementation in C++.

Basic example – European FX Call (FX1)

In this example, we demonstrate the pricing of a plain vanilla European Call option on foreign exchange. Our aim here is to calculate the premium of this financial derivative.

The full details of the contract, including the choice of mathematical model and its numerical method, are shown in the following Bento Box template for European Call FX option (FX1).
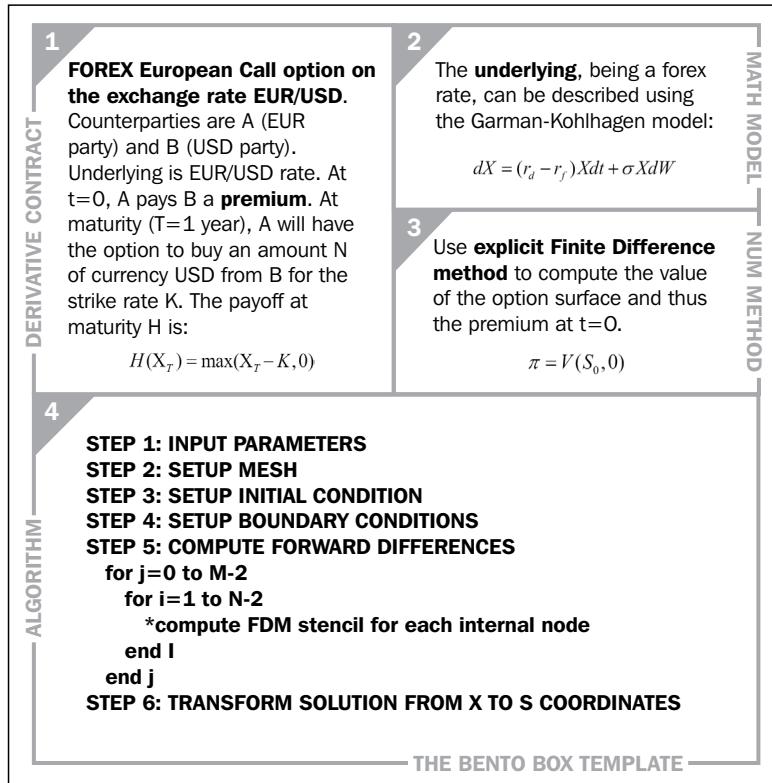
Bento Box template for European Call FX option (FX1)

We proceed by completing the contents of the Bento Box in clockwise sense, starting from the top-left corner. The following are the steps to do so:

1. **Derivative contract**: We first fill all the data of the contract, in particular the payoff function, which in our case is as follows:

$$H(X_T) = \max\left(X_T - K, 0\right)$$

Equation 1

2. **Math model**: We ought to select the mathematical model for the underlying, which in the case of currencies is the Garman-Kohlhagen model.

3. **Numerical method**: We select the numerical method to be used and in this case, we choose the finite difference method.

4. **Algorithm**: We construct the algorithm that will put together these calculations as a series of calculation steps, which will serve us as blueprint for implementing it in C++.

Note that a finite difference algorithm, in contrast to a Monte Carlo simulation, does not require a random number generator to operate. All computations are deterministic.

An important feature of finite difference methods is that they require the definition of a mesh. This mesh is essentially the collection of coordinates in which the **Partial Differential Equation** (**PDE**) will be approximated. In the case of equities, the Black-Scholes PDE, for example, is defined in terms of two independent variables: the stock price *S* and time *t*. In the case of currencies, the Garman-Kohlhagen PDE is defined in terms of two variables: the exchange rate *X* and time *t*. The solution domain is therefore the area defined by all the possible values that the pairs *X* and *t* can take in the *X* and *t* plane. For example, if we are considering a European Call currency option with strike 1.0 EUR/GBP, spot price 1.0 EUR/GBP and a maturity of one year, our solution domain gamma could be defined as the range of values for *X* between 0.5 and 1.5, and the range of values for *t* between 0 and 1. In mathematical terms, this can be represented as follows:

$$\Omega = \left\{ (X,t) \forall X \in [0.5, 1.5] \times t \in [0,1] \right\}$$

Equation 2

This rectangular domain then needs to be divided or discretized. This means that we have to transform it from a continuous into a discrete domain. Usually in finite differences, what we do is divide it into a number *N* of equidistant steps in the *X* axis and into a number *M* of equidistant steps in the *t* axis. The result is a grid that resembles a mesh and thus the origin of the name.

Note that we present the implementation of the explicit finite difference method, as described in *Chapter 3*, *Numerical Methods*, using a transformation of variables. This is done to transform the original PDE into an equivalent but simplified dimensionless PDE, which describes the diffusion of heat. This dimensionless version of the PDE is easier to solve using FDM.

Because of this transformation, the solution domain is not changed into two new variables *x* and τ. And so the PDE is solved in the domain defined by the following equation:

$$\Omega = \left\{ (x, \tau) \right\}$$

Equation 3

We consider an example of a European Call option on currency with strike 0.75 EUR/USD with a spot price of 0.75 EUR/USD. The option has six months to maturity. We divide the *x* axis in *N=5* steps and the tau axis in *M=6* steps. The premium under these conditions is 4.36 EUR/USD.

The upcoming code snippets implement the algorithm from the Bento Box template.

Code 9 – FX1_main.cpp (finite difference FX European Call)

The following is the code snippet for FX1_main.cpp file:

```cpp
// FX1_main.cpp
// requires FX_source.cpp, FX_print.cpp

#include "FX.h"

using namespace std;

int main()
{
  cout << "\n *** START FX1: Finite Difference European Call ***
    \n\n";

  // STEP 1: INPUT PARAMETERS
  auto T = 0.5; // maturity
  auto K = 75.0; // strike
  auto S0 = 75.0; // spot
  auto sigma = 0.30; // volatility
  auto r = 0.05; // interest rate
  auto dx = 0.5; // space step
  auto dt = 0.1; // time step
  auto N = 5; // number of space steps
  auto M = 6; // number of time steps

  // Construct a FX_EQ1 object from the input parameters:

  FX fx_eq1(T, K, S0, sigma, r, dt, dx, N, M);

  // Ask the object to evaluate the FX data for European Call:

  auto result = fx_eq1.get_data_and_premium();

  // STEP 7: OUTPUT RESULTS

  cout << result;
```

```
cout << "\n *** END FX1: Finite Difference European Call ***
  \n";

return 0;
}
```

Code 10 – FX1_source.cpp (finite difference FX European Call)

The following is the code snippet for FX1_source.cpp file:

```cpp
// FX1_source.cpp
#include "FX.h"
#include "matrix.h"
#include <algorithm>
using namespace std;

result_data FX::evaluate_data_and_premium() const
{
  double dtau, alpha, k;

  vector<double> t, tau, S, x;

  matrix<double> u, v;

  matrix_resize(u, N, M);

  matrix_resize(v, N, M);

  // Therefore, both the matrices u, v are resized to N by M
  // Now, let us resize the vectors t, tau, S and x:
  t.resize(M);
  tau.resize(M);
  S.resize(N);
  x.resize(N);

  dtau = dt * (0.5*sigma*sigma);
  alpha = dtau / (dx*dx);
  k = r / (0.5*sigma*sigma);
  double xmin = -1;
  double xmax = +1;

  // STEP 2: SETUP MESH (x and tau grids)
```

```
for (int i = 0; i < N; i++)
{
  x[i] = xmin + i*dx;
  S[i] = K*exp(x[i]);
}

for (int j = 0; j < M; j++)
{
  t[j] = j*dt;
  tau[j] = (T - t[j]) / (0.5*sigma*sigma);
}

// STEP 3: SETUP INITIAL CONDITION
for (int i = 0; i < N; i++)
{
  u[i][0] = max(exp(0.5*(k + 1)*x[i]) - exp(0.5*(k - 1)*x[i]),
    0.0);
}

// STEP 4: SETUP BOUNDARY CONDITIONS
for (int j = 1; j < M; j++)
{
  u[0][j] = 0.0;
  u[N - 1][j] = u[N - 1][0];
}

// STEP 5: COMPUTE FORWARD DIFFERENCES
for (int j = 0; j < M - 1; j++)
{
  for (int i = 1; i < N - 1; i++)
  {
    u[i][j + 1] = alpha*u[i + 1][j] + (1 - 2 * alpha)*u[i][j] +
      alpha*u[i - 1][j];
  }
}

// STEP 6: TRANSFORM SOLUTION FROM X TO S COORDINATES (u and v)
for (int j = 0; j < M; j++)
{
  for (int i = 0; i < N; i++)
  {
```

```
    v[i][j] = pow(K, (0.5*(1 + k)))*pow(S[i], (0.5*(1 -
        k)))*exp(1.0 / 8.0*(k + 1)*(k + 1)*sigma*sigma*(T -
        t[i]))*u[i][j];
    }
  }

  result_data result(alpha, dtau, k, x, S, t, tau, u, v);

  return result;

}
```

> [💡 For code snippet 11 `FX.h`, code snippet 12 `FX_print.cpp`,
> and code snippet 13 `matrix.h`, please refer to the code bundle
> of the book.]

To compute the basic example (FX1), you need to compile and run code
snippets 9,10,11,12, and 13 (which include a matrix and printing utility);
afterwards, you should obtain the following screenshot:

Basic example (FX1): FX European Call screenshot with results

Advanced example – FX barrier option (FX2)

In this second example, we consider the pricing of an exotic option: an up-and-out barrier with a call payoff. The details of the approach are shown in the following Bento Box template for FX Barrier Up and Out option (FX2):

1 DERIVATIVE CONTRACT

FOREX Up-and-Out Barrier Call option on the exchange rate EUR/USD. Counterparties are A (EUR party) and B (USD party). Underlying is EUR/USD rate. At t=0, A pays B a **premium**. At maturity (T=1 year), A will have the option to buy an amount N of currency USD from B for the strike rate K. The payoff at maturity H is:

$$H(X_T) = \max(X_T - K, 0)$$

2 MATH MODEL

The **underlying**, being a forex rate, can be described using the Garman-Kohlhagen model:

$$dX = (r_d - r_f)Xdt + \sigma XdW$$

3 NUM METHOD

Use **explicit Finite Difference method** to compute the value of the option surface and thus the premium at t=0.

$$\pi = V(S_0, 0)$$

4 ALGORITHM

STEP 1: INPUT PARAMETERS
STEP 2: SETUP MESH
STEP 3: SETUP INITIAL CONDITION
STEP 4: SETUP BOUNDARY CONDITIONS
STEP 5: COMPUTE FORWARD DIFFERENCES
   for j=0 to M-2
     for i=1 to N-2
       *compute FDM stencil for each internal node
     end I
   end j
STEP 6: TRANSFORM SOLUTION FROM X TO S COORDINATES

THE BENTO BOX TEMPLATE

Bento Box template for FX Barrier Up and Out option (FX2)

Note that there is a great advantage of using **Finite Difference Methodology** (**FDM**) with respect to **Monte Carlo** (**MC**) in pricing a continuously monitored barrier option. This is because MC is rather complex to incorporate the continuously monitored features, leaving us with little choice but to increase the number of fixing/observation points in the MC program. However, this will significantly increase the computation time in MC. We do not need to do this in FDM, making it more efficient.

Our target is to compute the option premium as we did earlier.

An up-and-out barrier is just like a standard European Call option but with one crucial difference—if the underlying crosses the limiting upper barrier, the option has a value of zero. The pricing algorithm and its implementation are thus almost identical, but with the difference that the upper boundary condition will now set the value to zero.

Barrier options are useful in finance because their premium is smaller than those of standard European options. They are cheaper to the investor, because he/she is taking the risk of not exercising it if the level of the underlying is too high (up-and-out barriers) or too low (down-and-out barriers).

We consider the same example as we did earlier, but with a barrier *B = 1.5* EUR/USD. The premium for this option under these conditions is 4.11 EUR/USD with an execution time of 2.22 seconds.

The upcoming code snippets implement the algorithm from the Bento Box template.

Code 14 – FX2_main.cpp (FDM FX barrier option)

The following is the code snippet for `FX2_main.cpp` file:

```cpp
// FX2_main.cpp
// requires FX2_source.cpp, FX_print.cpp

#include "FX.h"
#include <iostream>

using namespace std;

int main()
{
  cout << "\n *** START FX2: Finite Difference"
    << " European Up-and-Out Barrier Call *** \n\n";

  // STEP 1: INPUT PARAMETERS
  auto T = 0.5; // maturity
  auto K = 75.0; // strike
  auto S0 = 75.0; // spot
  auto sigma = 0.30; // volatility
  auto r = 0.05; // interest rate
  auto dx = 0.5; // space step
  auto dt = 0.1; // time step
  auto N = 5; // number of space steps
```

```
    auto M = 6; // number of time steps

    // Construct a FX object from the input parameters:

    FX fx_eq2(T, K, S0, sigma, r, dt, dx, N, M);

    // Ask the object to evaluate the FX data
    // for European Up-and_Out Barrier Call:

    auto result = fx_eq2.get_data_and_premium();

    // STEP 7: OUTPUT RESULTS
    cout << result;

    cout << "\n *** END FX2: Finite Difference"
      << " European Up-and-Out Barrier Call *** \n";

    return 0;
}
```

Code 15 – FX2_source.cpp (FDM FX barrier option)

The following is the code snippet for FX2_source.cpp file:

```
// FX2_source.cpp

#include "FX.h"
#include "matrix.h"
#include <algorithm>

using namespace std;

result_data FX::evaluate_data_and_premium() const
{
  double dtau, alpha, k
  vector<double> t, tau, S, x
  matrix<double> u, v
  auto resz = [this](matrix<double>& u, int N, int M) {

    // to make number of rows =
    u.resize(N);

    // to make number of columns =
    for (auto& row : u)
    row.resize(M);
  };

  resz(u, N, M)
```

```
resz(v, N, M);

// Therefore, both the matrices u, v are resized to N by M
// Now, let us resize the vectors t, tau, S and x:
t.resize(M);
tau.resize(M);
S.resize(N);
x.resize(N);

dtau = dt * (0.5*sigma*sigma);
alpha = dtau / (dx*dx);
k = r / (0.5*sigma*sigma);

double xmin = -1;
double xmax = +1;

// STEP 2: SETUP MESH (x and tau grids)
for (int i = 0; i < N; i++)
{
  x[i] = xmin + i*dx;
  S[i] = K*exp(x[i]);
}

for (int j = 0; j < M; j++)
{
  t[j] = j*dt;
  tau[j] = (T - t[j]) / (0.5*sigma*sigma);
}

// STEP 3: SETUP INITIAL CONDITION
for (int i = 0; i < N; i++)
{
  u[i][0] = max(exp(0.5*(k + 1)*x[i]) - exp(0.5*(k - 1)*x[i]),
    0.0);
}

// STEP 4: SETUP BOUNDARY CONDITIONS
for (int j = 1; j < M; j++)
{
  u[0][j] = 0.0;
  u[N - 1][j] = 0.0;
}

// STEP 5: COMPUTE FORWARD DIFFERENCES
for (int j = 0; j < M - 1; j++)
{
  for (int i = 1; i < N - 1; i++)
  {
```

```
      u[i][j + 1] = alpha*u[i + 1][j] + (1 - 2 * alpha)*u[i][j] +
        alpha*u[i - 1][j];
    }
  }

  // STEP 6: TRANSFORM SOLUTION FROM X TO S COORDINATES (u and v)
  for (int j = 0; j < M; j++)
  {
    for (int i = 0; i < N; i++)
    {
      v[i][j] = pow(K, (0.5*(1 + k)))*pow(S[i], (0.5*(1 - k)))
      *exp(1.0 / 8.0*(k + 1)*(k + 1)*sigma*sigma*(T -
        t[i]))*u[i][j];
    }
  }

  result_data result(alpha, dtau, k, x, S, t, tau, u, v);

  return result;

}
```

To compute the advanced example (FX2), you need to compile and run code snippets 14 and 15 plus the previous 11, 12, and 13; afterwards, you should obtain the following screenshot:

Advanced example (FX2): FX up-and-out barrier call screenshot with results

Summary

In this chapter, we have solved two pricing problems in forex derivatives. We have seen a basic example and a more complex one (plain vanilla) and an advanced example (exotic) including a barrier option. For each, we have provided the complete C++ implementation.

We will now proceed to the next asset class and interest rate derivatives.

6

Interest Rate Derivatives with C++

This chapter illustrates the application of C++ to the pricing of interest rate derivatives. We will consider two examples: the pricing of a plain vanilla **Interest Rate Swap** (**IRS**) (basic example) and the pricing of a Cap (advanced example). We provide the full working C++ implementation for both. Both the examples are solved using one factor **Libor Market Model** (**LMM**) and Monte Carlo simulation. A simpler C implementation (without the OO features) can be found in the accompanying book website. The LMM is described in "The Market Model of Interest Rate Dynamics". An excellent description of the Monte Carlo simulation can be found in "Efficient Methods for Valuing Interest Rate Derivatives".

Basic example – plain vanilla IRS (IR1)

In this example, we will demonstrate the pricing of a plain vanilla IRS. The full details of the contract, including the choice of the mathematical model and its numerical method, are shown in the following Bento Box template:
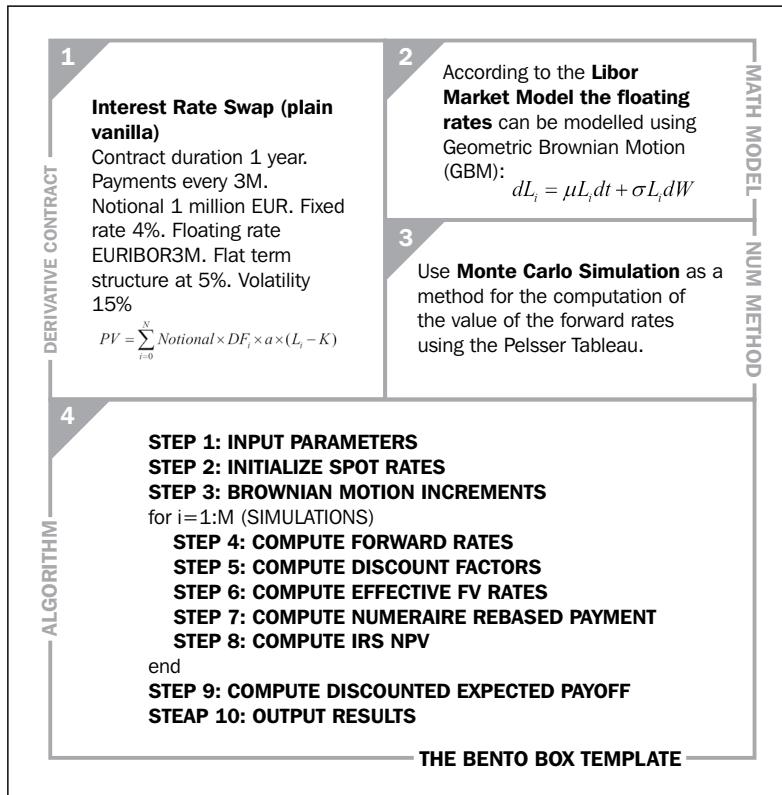
Bento Box template for basic example (IR1)

Our aim here is to calculate the net present value of this IRS, in particular a paying fixed-for-floating IRS. In this contract, the holder pays the fixed rate and receives the floating rate at regular intervals.

We proceed by completing the contents of the Bento Box in clockwise sense, starting from the top-left corner. First, we will fill all the data of the contract, in particular the payoff function, which in our case is as follows:

$$PV = \sum_{i=0}^{N} Notional \times DF_i \times \alpha \times \left(L_i - K \right)$$

Equation 1

The present value of the IRS is the sum of the discounted future payments of the IRS. Being a paying IRS, we pay the fixed rate *K* and we receive the future floating rate *L*. This rate is fixed (that is, determined) at the beginning of the period and it runs up to the maturity date (when the payment is made). DF_i stands for the respective discount factors. Each payment is multiplied by the notional and day count fraction α.

Second, we ought to select the mathematical model for the underlying. In the case of interest rates, we can choose between short rate models (such as the Vasicek and the Hull and White) or the market models (such as LMM). In this chapter, we select the LMM to solve these problems. Third, we select the numerical method to be used and in this case, we choose the Monte Carlo method. This method will allow us to simulate the random behavior of the forward rates. Fourth, we construct the algorithm that will put together these calculations as a series of calculation steps, which will serve as a blueprint for implementing it in C++.

There are many variations of the LMM, in terms, the number of rates used (multifactor), or the underlying used. (The swap LMM uses the swap rate instead of the forward rate as fundamental unknown.) In this chapter, we will consider only one factor, (lognormal forward) LMM.

The algorithm is shown in box 4 of the Bento Box template. The implementation of the algorithm in C++ is shown in code snippets 16, 17, and 18. Code snippet 16 is the main code block, code snippet 17 is its associated source, while code snippet 18 contains the header file. The algorithm is composed of 10 steps, which take us from the input parameters (**STEP 1**) to the output of the present value of the IRS (**STEP 10**).

Note that Monte Carlo simulation requires a random number generator to operate. We will take advantage of the random number generator, which we developed in *Chapter 3*, *Numerical Methods*, to price equity derivatives (the Box-Muller algorithm).

We consider an example of a plain vanilla IRS on a notional of one million EUR. The length of the contract is one year and the frequency of payments is every three months. The floating rates are therefore indexed to EURIBOR3M. The fixed rate is 4 percent p.a.

We use LMM with Monte Carlo simulation with 10,000 simulations. We assume an initial flat term structure of interest rates at 5 percent p.a. We also assume a volatility of 15 percent for the forward rates (this value is usually calibrated from observed swaptions in the market).

The upcoming code snippets implement the algorithm from the Bento Box template.

Code 16 – IR1_main.cpp (IRS with Monte Carlo LMM)

The following is the code snippet for IR1_main.cpp file:

```
// IR1_main.cpp
// requires random.cpp IR1_source.cpp

#include "IR.h"
#include <iostream>
using namespace std;

int main()
{

  cout << "\n *** START IR1: IRS Monte Carlo Libor Market Model 1F
    * ** \n\n";

  // Plain Vanilla IRS, pays fixed, receives floating
  // freq payments every 3M, maturity 1 year

  // STEP 1: INPUT PARAMETERS
  double notional = 1000000; // notional
  double K = 0.04; // fixed rate IRS
  double alpha = 0.25; // daycount factor
  double sigma = 0.15; // fwd rates volatility
  double dT = 0.25;
  int N = 3; // number forward rates
  int M = 1000; // number of simulations

  // Construct a IR object from the input parameters:

  IR ir1(notional, K, alpha, sigma, dT, N, M);

  // Obtain the value of premium from member function
    "get_premium()":
```

```
    auto results = ir1.get_simulation_data();

    // STEP 10: OUTPUT RESULTS
    auto sz = results.datapoints.size();
    for (decltype(sz) nsim = 0; nsim < sz; ++nsim)
    {
      cout << "simIRS[" << nsim << "] = " <<
        results.datapoints[nsim] << endl;
    }

    cout << "\n *** IRS PV = " << results.Value << endl;
    cout << "\n *** END IR1: IRS Monte Carlo Libor Market Model 1F
      *** \n";

    return 0;
}
```

Code 17 – IR1_source.cpp (IRS with Monte Carlo LMM)

The following is the code snippet for IR1_source.cpp file:

```
// IR1_source.cpp

#include "IR.h"
#include "random.h"
#include "matrix.h"
#include <algorithm>
#include <iostream>

using namespace std;

IR_results IR::run_LIBOR_simulations() const
{
  matrix<double> L; // forward rates
  matrix_resize(L, N + 1, N + 1);
  matrix<double> D; // discount factors
  matrix_resize(D, N + 2, N + 2);
  vector<double> dW(N + 1); // discount factors
  vector<double> FV(N + 2); // future value payment
  vector<double> FVprime(N + 2); // numeraire-rebased FV payment
  vector<double> V(M); // simulation payoff

  // Composing the SampleBoxMuller class:

  SampleBoxMuller normal;

  double df_prod = 1.0;
  double drift_sum = 0.0;
```

```cpp
   double sumPV = 0.0;
   double PV = 0.0;

   // STEP 2: INITIALISE SPOT RATES
   L[0][0] = 0.05;
   L[1][0] = 0.05;
   L[2][0] = 0.05;
   L[3][0] = 0.05;

   // start main MC loop

   for (int nsim = 0; nsim < M; ++nsim)
   {

     // STEP 3: BROWNIAN MOTION INCREMENTS
     dW[1] = sqrt(dT)*normal();
     dW[2] = sqrt(dT)*normal();
     dW[3] = sqrt(dT)*normal();

     // STEP 4: COMPUTE FORWARD RATES TABLEAU
     for (int n = 0; n < N; ++n)
     {
       for (int i = n + 1; i < N + 1; ++i)
       {
         drift_sum = 0.0;
         for (int k = i + 1; k < N + 1; ++k)
         {
           drift_sum += (alpha*sigma*L[k][n]) / (1 +
             alpha*L[k][n]);
         }
         L[i][n + 1] = L[i][n] * exp((-drift_sum*sigma -
           0.5*sigma*sigma)*dT + sigma*dW[n + 1]); // cout <<"L: i=
           " << i <<", n+1 = " << n+1 " << L[i][n+1] << "\n";
       }
     }
     // STEP 5: COMPUTE DISCOUNT RATES TABLEAU
     for (int n = 0; n < N + 1; ++n)
     {
       for (int i = n + 1; i < N + 2; ++i)
       {
         df_prod = 1.0;
         for (int k = n; k < i; k++)
         {
           df_prod *= 1 / (1 + alpha*L[k][n]);
         }
         D[i][n] = df_prod;
         // cout <<"D: i = " << i <<", n = " << n <<", D[i][n] = "
           << D[i][n] << "\n";
```

```
    }
  }

  // STEP 6: COMPUTE EFFECTIVE FV PAYMENTS
  FV[1] = notional*alpha*(L[0][0] - K);
  FV[2] = notional*alpha*(L[1][1] - K);
  FV[3] = notional*alpha*(L[2][2] - K);
  FV[4] = notional*alpha*(L[3][3] - K);

  // STEP 7: COMPUTE NUMERAIRE-REBASED PAYMENT
  FVprime[1] = FV[1] * D[1][0] / D[4][0];
  FVprime[2] = FV[2] * D[2][1] / D[4][1];
  FVprime[3] = FV[3] * D[3][2] / D[4][2];
  FVprime[4] = FV[4] * D[4][3] / D[4][3];

  // STEP 8: COMPUTE IRS NPV

  V[nsim] = FVprime[1] * D[1][0] + FVprime[2] * D[2][0] +
    FVprime[3] * D[3][0] + FVprime[4] * D[4][0];
}
// end main MC loop

// STEP 9: COMPUTE DISCOUNTED EXPECTED PAYOFF
sumPV = 0.0;
for (int nsim = 0; nsim < M; nsim++)
{
  sumPV += V[nsim];
}

PV = sumPV / M;

IR_results results(V, PV);

return results;
}
```

For code snippet 18 IR.h, please refer to the code in the code bundle.

To compute the basic example (IR1), you will have to compile and run code snippets 16, 17, 18, 4, 5, and 13 (which include the header, matrix, and random functions). Afterwards, you should obtain the following screen:

Basic example (IR1) screenshot with results

Advanced example – IRS with Cap (IR2)

In this second example, we consider the pricing of an IRS with a cap The details of the approach are shown in the following Bento Box template advanced example:
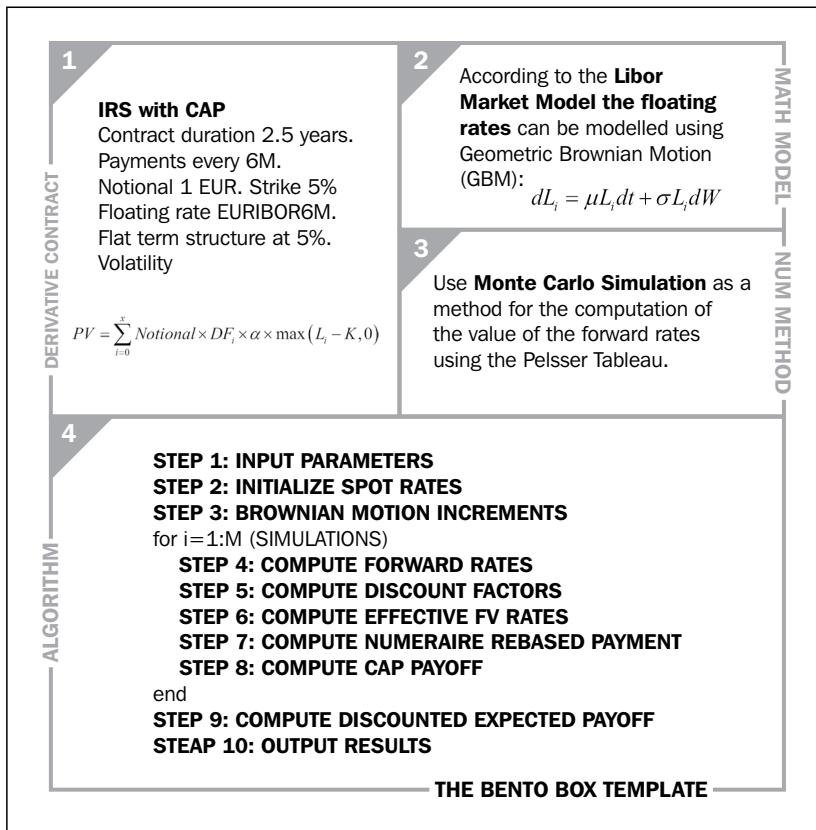
Bento Box template for advanced example (IR2)

Our target is to compute the net present value of the swap as we did earlier.

An IRS cap is just like a standard IRS but with one key difference—at each payment date, we compute the maximum between (a) the difference between the floating rate and the strike and (b) zero. With this difference, we compute the value of a caplet; the cap is simply the sum of the caplets contained in the IRS.

The algorithm is shown in box 4 of the Bento Box template. The implementation of the algorithm in C++ is shown in code snippets 19 and 20. Code snippet 19 is the main code block, while code snippet 20 is its associated source.

Code 19 – IR2_main.cpp (cap with Monte Carlo LMM)

The following is the code snippet for `IR2_main.cpp` file:

```cpp
// IR2_main.cpp
// requires random.cpp IR2_source.cpp

#include "IR.h"
#include <iostream>
using namespace std;

int main()
{
  std::cout << "\n *** START IR2: CAP Monte Carlo Libor Market
    Model 1F * ** \n\n";

  // STEP 1: INPUT PARAMETERS
  double K = 0.05; // strike caplet
  double alpha = 0.5; // daycount factor
  double sigma = 0.15; // fwd rates volatility
  double dT = 0.5;
  int N = 4; // number forward rates
  int M = 1000; // number of simulations

  // Construct a IR object from the input parameters:
  IR ir2(K, alpha, sigma, dT,  N, M);

  // Obtain the value of premium from member function
    "get_premium()":

  auto results = ir2.get_simulation_data();

  // STEP 10: OUTPUT RESULTS
  auto sz = results.datapoints.size();

  for (decltype(sz) nsim = 0; nsim < sz; ++nsim)
  {
    cout << "Vcap[" << nsim << "] = " << results.datapoints[nsim]
      << endl;
  }

  cout << "\n *** IRS cap = " << results.Value << "\n";

  cout << "\n *** END IR2: CAP Monte Carlo Libor Market Model 1F *
    ** \n";

  return 0;
}
```

Code 20 – IR2_source.cpp (cap with Monte Carlo LMM)

The following is the code snippet for `IR2_source.cpp` file:

```cpp
// IR2_source.cpp

#include "IR.h"
#include "random.h"
#include "matrix.h"
#include <algorithm>
#include <iostream>

using namespace std;

IR_results IR::run_LIBOR_simulations() const
{
  matrix<double> L; // forward rates
  matrix_resize(L, N + 1, N + 1);
  matrix<double> D; // discount factors
  matrix_resize(D, N + 2, N + 2);
  vector<double> dW(N + 1); // discount factors
  vector<double> V(N + 2); // caplet payoff
  vector<double> Vprime(N + 2); // numeraire-rebased caplet payoff
  vector<double> Vcap(M); // simulation payoff

  // Composing the SampleBoxMuller class:
  SampleBoxMuller normal;

  double df_prod = 1.0;
  double drift_sum = 0.0;
  double sumcap = 0.0;
  double payoff = 0.0;

  // STEP 2: INITIALISE SPOT RATES
  L[0][0] = 0.05;
  L[1][0] = 0.05;
  L[2][0] = 0.05;
  L[3][0] = 0.05;
  L[4][0] = 0.05;

  // start main MC loop

  for (int nsim = 0; nsim < M; ++nsim)
  {
    // STEP 3: BROWNIAN MOTION INCREMENTS
    dW[1] = sqrt(dT)*(normal());
    dW[2] = sqrt(dT)*(normal());
```

```
      dW[3] = sqrt(dT)*(normal());
      dW[4] = sqrt(dT)*(normal());

      // STEP 4: COMPUTE FORWARD RATES TABLEAU
      for (int n = 0; n < N; ++n)
      {
        for (int i = n + 1; i < N + 1; ++i)
        {
          drift_sum = 0.0;
          for (int k = i + 1; k < N + 1; ++k)
          {
            drift_sum += (alpha*sigma*L[k][n]) / (1 +
              alpha*L[k][n]);
          }
          L[i][n + 1] = L[i][n] * exp((-drift_sum*sigma -
            0.5*sigma*sigma)*dT
          + sigma*dW[n + 1]);
          // cout <<"L: i = " << i <<", n+1 = " << n+1 <<", = " <<
            L[i][n+1] << "\n";
        }
      }

      // STEP 5: COMPUTE DISCOUNT RATES TABLEAU
      for (int n = 0; n < N + 1; ++n)
      {
        for (int i = n + 1; i < N + 2; ++i)
        {
          df_prod = 1.0;
          for (int k = n; k < i; k++)
          {
            df_prod *= 1 / (1 + alpha*L[k][n]);
          }
          D[i][n] = df_prod;
          // cout <<"D: i = " << i <<", n = " << n <<", D[i][n] = "
          //               << D[i][n] << "\n";
        }
      }

      // STEP 6: COMPUTE CAPLETS
      double diff;
      diff = L[0][0] - K;
      V[1] = max(diff, 0.0);
      diff = L[1][1] - K;
      V[2] = max(diff, 0.0);
      diff = L[2][2] - K;
```

```
    V[3] = max(diff, 0.0);
    diff = L[3][3] - K;
    V[4] = max(diff, 0.0);
    diff = L[4][4] - K;
    V[5] = max(diff, 0.0);

    // STEP 7: COMPUTE NUMERAIRE-REBASED CAPLETS
    Vprime[1] = V[1] * D[1][0] / D[5][0];
    Vprime[2] = V[2] * D[2][1] / D[5][1];
    Vprime[3] = V[3] * D[3][2] / D[5][2];
    Vprime[4] = V[4] * D[4][3] / D[5][3];
    Vprime[5] = V[5] * D[5][4] / D[5][4];

    // STEP 8: COMPUTE CAP PAYOFF
    Vcap[nsim] = Vprime[1] + Vprime[2] + Vprime[3] + Vprime[4] +
      Vprime[5];
  }
  // end main MC loop

  // STEP 9: COMPUTE DISCOUNTED EXPECTED PAYOFF
  sumcap = 0.0;

  for (int nsim = 0; nsim < M; ++nsim)
  {
    sumcap += Vcap[nsim];
  }

  payoff = D[N + 1][0] * sumcap / M;

  IR_results results(Vcap, payoff);

  return results;
}
```

We consider the example of an IRS having a cap with a strike 5 percent and a maturity of 2.5 years. We assume a flat term structure of 5 percent with a forward volatility of 15 percent. Payments are every six months and a notional of 1 EUR. The floating rate is EURIBOR6M.

To compute the advanced example (IR2), you need to compile and run code snippets 19, 20, 18, 4, 5, and 13 (which include the a header, matrix, and random functions). Afterwards, you should obtain the following screenshot:
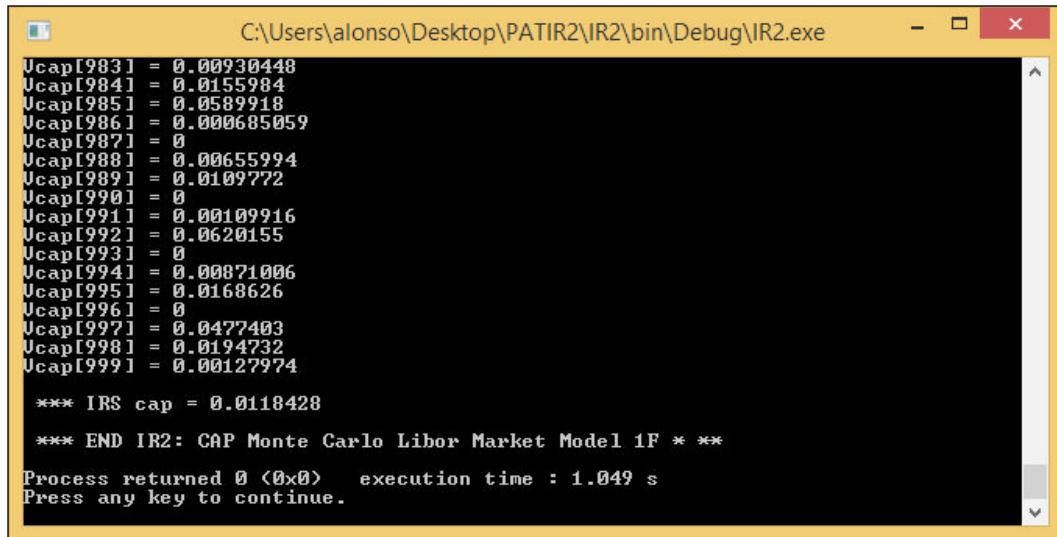

```
C:\Users\alonso\Desktop\PATIR2\IR2\bin\Debug\IR2.exe                    –  □  ×
Vcap[983] = 0.00930448
Vcap[984] = 0.0155984
Vcap[985] = 0.0589918
Vcap[986] = 0.000685059
Vcap[987] = 0
Vcap[988] = 0.00655994
Vcap[989] = 0.0109772
Vcap[990] = 0
Vcap[991] = 0.00109916
Vcap[992] = 0.0620155
Vcap[993] = 0
Vcap[994] = 0.00871006
Vcap[995] = 0.0168626
Vcap[996] = 0
Vcap[997] = 0.0477403
Vcap[998] = 0.0194732
Vcap[999] = 0.00127974

 *** IRS cap = 0.0118428

 *** END IR2: CAP Monte Carlo Libor Market Model 1F * **

Process returned 0 (0x0)   execution time : 1.049 s
Press any key to continue.
```

Advanced example (IR2) screenshot with results

Summary

In this chapter, we have solved two pricing problems in interest rate derivatives. We have seen a basic example (plain vanilla IRS) and an advanced example. For each, we have provided the complete C++ implementation.

We will now proceed to the last asset class, credit derivatives, in the next chapter.

7

Credit Derivatives with C++

In this last chapter, we focus on the application of C++ to the pricing of credit derivatives. We consider two examples: the use of the Merton model to price a defaultable firm's equity plus the firm's default probability (basic example) and the pricing of **Credit Default Swap** (**CDS**) (advanced example). The first example is based on the structural approach to credit risk, while the second is based on the intensity approach. We provide the full working C++ implementation for both the examples. A simpler C implementation (without the OO features) can be found in the accompanying book website.

Basic example – bankruptcy (CR1)

In this example, we will study the default (bankruptcy) of a firm using the (Merton 1974) model. For more information, see "On the Pricing of Corporate Debt: The Risk Structure of Interest Rates". In this model, the dynamics of the firm are described using **Geometric Brownian Motion** (**GBM**). The capital structure of the firm is assumed to be very simple: the firm's assets (V) composed entirely of equity (E) and debt (D). For a given maturity T, default happens if the firm's assets at maturity $V(T)$ are less than the value of the debt (D) that the firm has to pay at that time.

In this context, we can study the probability of default of a firm using the Monte Carlo simulation. By generating a number of possible trajectories in which the firm can evolve and counting the times that the firm satisfies the default condition, we can estimate the likelihood of bankruptcy of a firm.

Our aim is then to calculate the number of times that *V(T)* are less than the value of the debt (*D*) and use this as an estimate of the **Probability of Default** (**PD**). In addition to using the same Monte Carlo computations, we can also estimate the value of the equity of the firm at *time=0*.

We now proceed by completing the contents of the Bento Box in clockwise sense, starting from the top-left corner. First, we fill all the data of the bankruptcy analysis, in particular default condition, which in our case is as follows:

$$V_T < D$$

Second, we ought to select the mathematical model for the underlying, which in the case of the Merton model is GBM. Third, we select the numerical method to be used and in this case, we choose the Monte Carlo simulation. Fourth, we construct the algorithm that will put together these calculations as a series of calculation steps, which will serve us as blueprint for implementing it in C++.

The algorithm is shown in Box 4 of the Bento Box template for bankruptcy. The implementation of the algorithm in C++ is shown in code snippet 1. The algorithm is composed of seven steps, which will take us from the input parameters (**STEP 1**) to the output of the premium value (**STEP 6**).

The Monte Carlo simulation requires a random number generator to operate and therefore, the `random.cpp` file (studied in *Chapter 4*, *Equity Derivatives in C++*) is re-used.

We consider an example of a firm whose capital structure is composed of the total firm's assets at *t=0*, *V(0)=100* million EUR, and a debt composed of a single zero coupon bond with a face value of *D=70* million EUR. The volatility of the firm's assets is assumed to be 20 percent. The maturity is four years. The risk-free interest rate is 5 percent pa.

Running the C++ code snippets as shown in the following figure, with 500 steps and 10,000 simulations, we estimate the probability of default in the four year period to be 88.63 percent and the equity value to be *E(0)=43.95* million EUR:
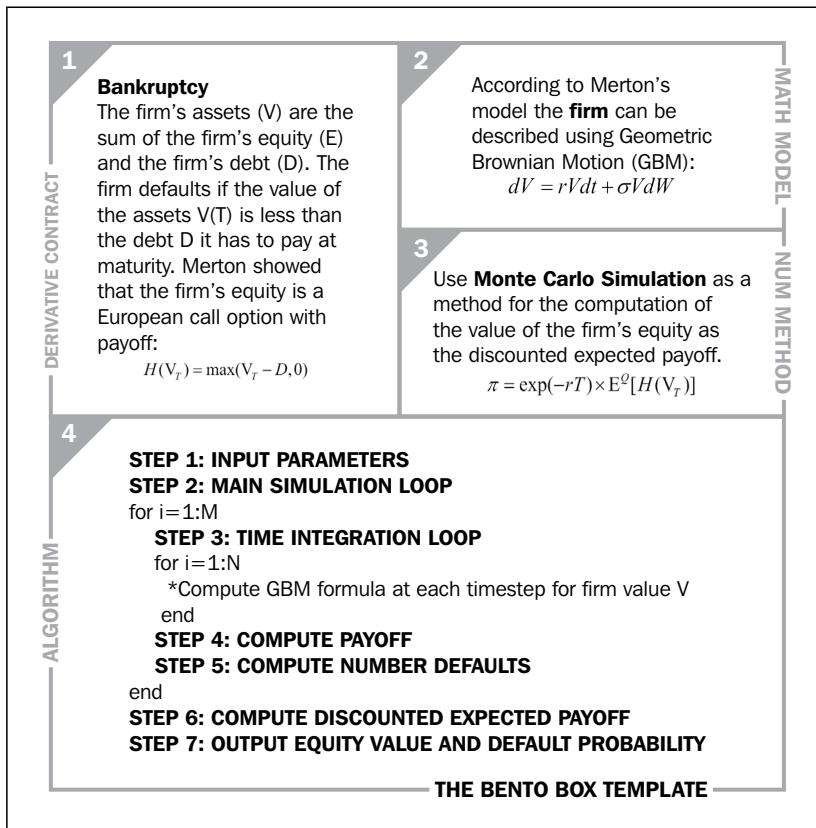
Bento Box template for firm's bankruptcy (CR1)

The upcoming code snippets implement the algorithm from the Bento Box template.

Code 21 – CR1_main.cpp (Bankruptcy using Merton model)

The following is the code snippet for CR1_main.cpp file:

```
// CR2_main.cpp

// It requires CR2_source.cpp
#include "CR2.h"

#include <iostream>

using namespace std;

int main()
```

```
{
    cout << "\n *** START CR2: Credit Default Swap *** \n";

    // STEP 1: INPUT PARAMETERS

    auto T = 1.0; // maturity
    auto N = 4; // number of payments per year
    auto notional = 100.0; // notional
    auto r = 0.05; // risk free interest rate
    auto h = 0.01; // hazard rate
    auto rr = 0.50; // recovery rate

    // Construct a CR2 object from the input parameters:

    CR2 cr2(T, N, notional, r, h, rr);

    // Obtain the value of premium from member function
       "get_premium()":

    auto cr2_results =
        cr2.get_pv_premium_and_default_legs_and_cds_spread();

    // STEP 6: OUTPUT RESULTS

    cout << "\n PV premium leg =  "
        << cr2_results.pv_premium_leg << "\n";

    cout << "\n PV default leg =  "
        << cr2_results.pv_default_leg << " \n";

    cout << "\n cds_spread =  "
        << cr2_results.cds_spread_in_bps << "  bps \n";

    cout << "\n *** END CR2: Credit Default Swap *** \n";

    return 0;
}
```

Code 22 – CR1_source.cpp (Bankruptcy using Merton model)

The following is the code snippet for CR1_source.cpp file:

```
// CR2_source.cpp

#include "CR2.H"
#include <vector>
```

```cpp
#include <cmath>

using namespace std;

CR2_results CR2::find_pv_premium_and_default_legs_and_cds_spread()
  const
{
  auto pv_premium_leg = 0.0; // sum premium leg
  auto pv_default_leg = 0.0; // sum default leg
  auto t = 0.0; // current time
  auto cds_spread = 0.0;
  auto array_size = static_cast<int>(N*T + 1);
  vector <double> DF(array_size);
  vector <double> P(array_size);
  P[0] = 1.0;
  auto dt = T / N;

  // STEP 2: LOOP FOR ALL PAYMENTS
  for (int j = 1; j < array_size; j++)
  {
    t = j*dt;
    DF[j] = exp(-r*t);
    P[j] = exp(-h*t);

    // STEP 3: COMPUTE PREMIUM PAYMENTS
    pv_premium_leg = pv_premium_leg + DF[j] * notional*dt*P[j];

    // STEP 4: COMPUTE DEFAULT PAYMENTS
    pv_default_leg = pv_default_leg + DF[j] * (1.0 -
      rr)*notional*(P[j - 1] - P[j]);
  }

  // STEP 5: COMPUTE CDS SPREAD
  cds_spread = pv_default_leg / pv_premium_leg;

  // Composing the CR2_results class:
  CR2_results results;
  results.pv_premium_leg = pv_premium_leg;
  results.pv_default_leg = pv_default_leg;
  results.cds_spread_in_bps = cds_spread * 10000;
  return results;
}
```

For code snippet 23 CR1.h, please refer to the code in the code bundle.

To compute the basic example (CR1), you need to compile and run code snippets 21, 22, 23, 4, and 5 (which include a header and random functions); afterwards, you should obtain the following screenshot:
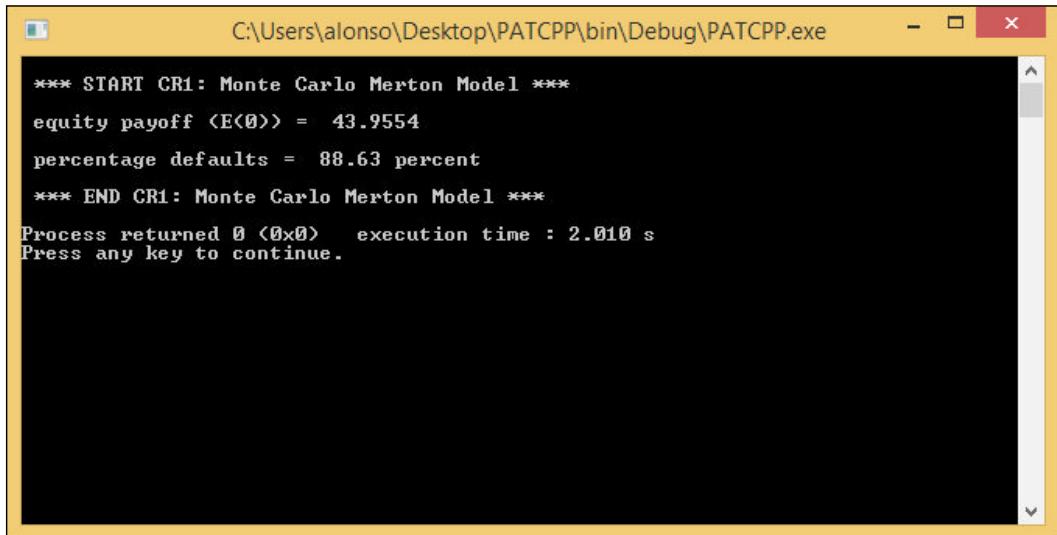
Firm's bankruptcy (CR1) screenshot with results

Advanced example – CDS (CR2)

In this second example, we consider the pricing of CDS. The details of the approach are shown in the following Bento Box template for the CDS:
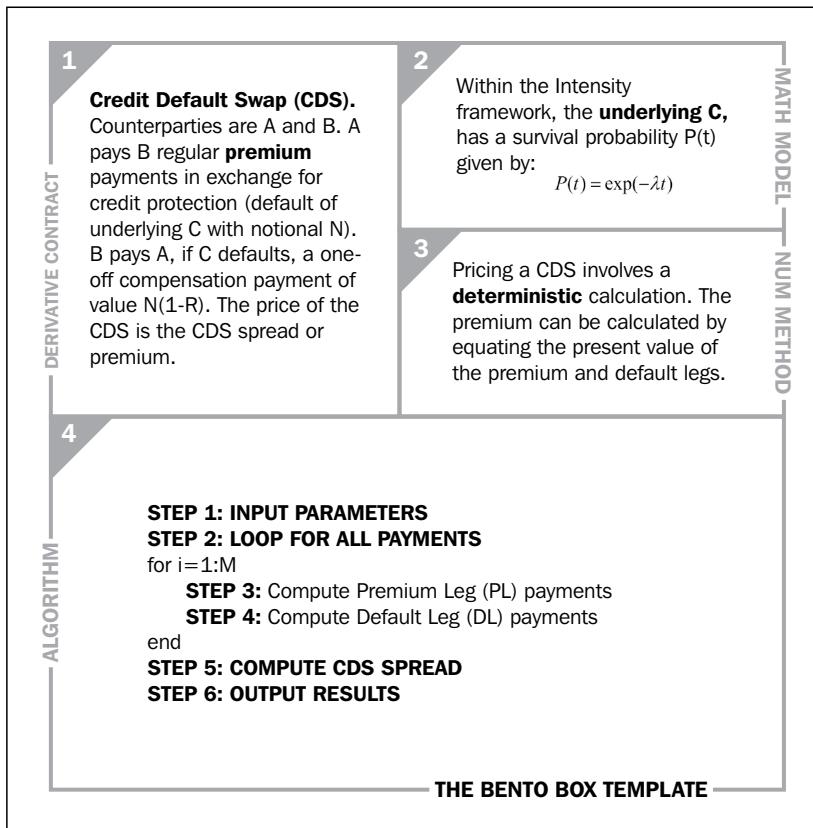
Bento Box template for CDS (CR2)

A CDS is a financial contract between two counterparties A and B, in which one party pays to the other party to buy credit protection against the possible default of an underlying C.

In structure, the CDS is similar to the plain vanilla IRS, as it is composed of an exchange of cash flows between the parties. In a typical CDS with duration of five years, counterparty A pays B a series of premium payments at regular intervals upon an agreed notional. These payments will be made as long as underlying C "survives" (that is, doesn't go in default).

Counterparty B pays A a single contingent payment at the time of default of underlying C. The amount paid is equal to the notional minus the recovery rate. In mathematical terms, it can be expressed as follows:

$$N(1-R)$$

Like in an IRS, the "price" of the contract is obtained by computing the present value of each leg (the sum of the expected premium payments called **Premium Leg** (**PL**) and the sum of the expected default payment called **Default Leg** (**DL**)). In mathematical terms, PL and DL are expressed as follows:

$$PL = \sum_{i=1}^{N} \pi \times N \times P(T_i) \times \Delta t \times DF_i$$

$$DL = \sum_{i=1}^{N} N \times (1-R) \times \left[P(T_{i-1}) - P(T_i) \right] \times DF_i$$

In the preceding equations, *P(T)* is the survival probability at time *t*, *N* is the notional, *R* is the recovery rate, and *DF(t)* is the discount factor at time *t*. For fair pricing, these legs must be equal and with this, we can determine what should be the fair value of the premium paid (also called CDS spread). The value of this spread, denoted by the Greek letter π, is regarded as the price of the CDS contract, In mathematical terms, it can be expressed as follows:

$$\pi = \frac{\sum_{i=1}^{N} \pi \times N \times P(T_i) \times \Delta t \times DF_i}{\sum_{i=1}^{N} N \times (1-R) \times \left[P(T_{i-1}) - P(T_i) \right] \times DF_i}$$

The pricing algorithm we present in the Bento Box template for the CDS attempts to compute the premium from the preceding equation.

As we can see, this calculation is deterministic and therefore no Monte Carlo simulation is required. The credit model we illustrate here is an example of "intensity models" based on the work of "Pricing Derivatives on Financial Securities Subject to Credit Risk".

Code snippet 2 illustrates the implementation of this pricing in C++.

We will consider the example where the contract duration is one year, quarterly payments (that is, four payments per year), notional = 100 million USD, risk-free rate = 5 percent pa, hazard rate of underlying = 1 percent pa, recovery rate = 50 percent. For these inputs, the CDS spread is 50.0626 basis points.

In the following figure, we find the Bento Box framework applied to our CDS problem:
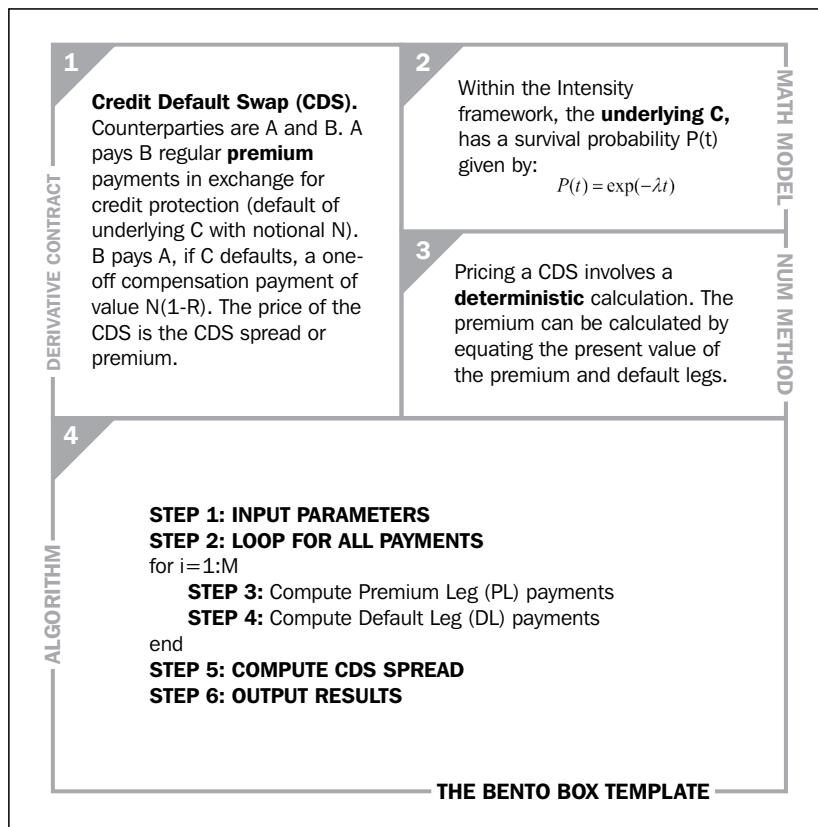
Bento Box template for CDS (CR2)

The upcoming code snippets implement the algorithm from the Bento Box template.

Code 24 – CR2_main.cpp (CDS)

The following is the code snippet for `CR2_main.cpp` file:

```cpp
// CR2_main.cpp

// It requires CR2_source.cpp
#include "CR2.h"
#include <iostream>

using namespace std;

int main()
{
  cout << "\n *** START CR2: Credit Default Swap *** \n";

  // STEP 1: INPUT PARAMETERS
  auto T = 1.0; // maturity
  auto N = 4; // number of payments per year
  auto notional = 100.0; // notional
  auto r = 0.05; // risk free interest rate
  auto h = 0.01; // hazard rate
  auto rr = 0.50; // recovery rate

  // Construct a CR2 object from the input parameters:
  CR2 cr2(T, N, notional, r, h, rr);

  // Obtain the value of premium from member function
    "get_premium()":

  auto cr2_results =
    cr2.get_pv_premium_and_default_legs_and_cds_spread();

  // STEP 6: OUTPUT RESULTS
  cout << "\n PV premium leg =  "
    << cr2_results.pv_premium_leg << "\n";

  cout << "\n PV default leg =  "
    << cr2_results.pv_default_leg << " \n";

  cout << "\n cds_spread =  "
    << cr2_results.cds_spread_in_bps << "  bps \n";

  cout << "\n *** END CR2: Credit Default Swap *** \n";

  return 0;
}
```

Code 25 – CR2_source.cpp (CDS)

The following is the code snippet for `CR2_source.cpp` file:

```cpp
// CR2_source.cpp

#include "CR2.H"
#include <vector>
#include <cmath>

using namespace std;

CR2_results CR2::find_pv_premium_and_default_legs_and_cds_spread()
  const
{
  auto pv_premium_leg = 0.0; // sum premium leg
  auto pv_default_leg = 0.0; // sum default leg
  auto t = 0.0; // current time
  auto cds_spread = 0.0;
  auto array_size = static_cast<int>(N*T + 1);
  vector <double> DF(array_size);
  vector <double> P(array_size);

  P[0] = 1.0;

  auto dt = T / N;

  // STEP 2: LOOP FOR ALL PAYMENTS
  for (int j = 1; j < array_size; j++)
  {
    t = j*dt;
    DF[j] = exp(-r*t);
    P[j] = exp(-h*t);

    // STEP 3: COMPUTE PREMIUM PAYMENTS
    pv_premium_leg = pv_premium_leg + DF[j] * notional*dt*P[j];

    // STEP 4: COMPUTE DEFAULT PAYMENTS
    pv_default_leg = pv_default_leg + DF[j] * (1.0 -
      rr)*notional*(P[j - 1] - P[j]);
  }

  // STEP 5: COMPUTE CDS SPREAD
  cds_spread = pv_default_leg / pv_premium_leg;

  // Composing the CR2_results class:
  CR2_results results;
  results.pv_premium_leg = pv_premium_leg;
```

```
    results.pv_default_leg = pv_default_leg;
    results.cds_spread_in_bps = cds_spread * 10000;
    return results;
}
```

> For code snippet 26 `CR2.h`, please refer to the code in the code bundle.

To compute the advanced example (CR2), you will have to compile and run code snippets 24, 25, and 26; afterwards, you should obtain the following screenshot:
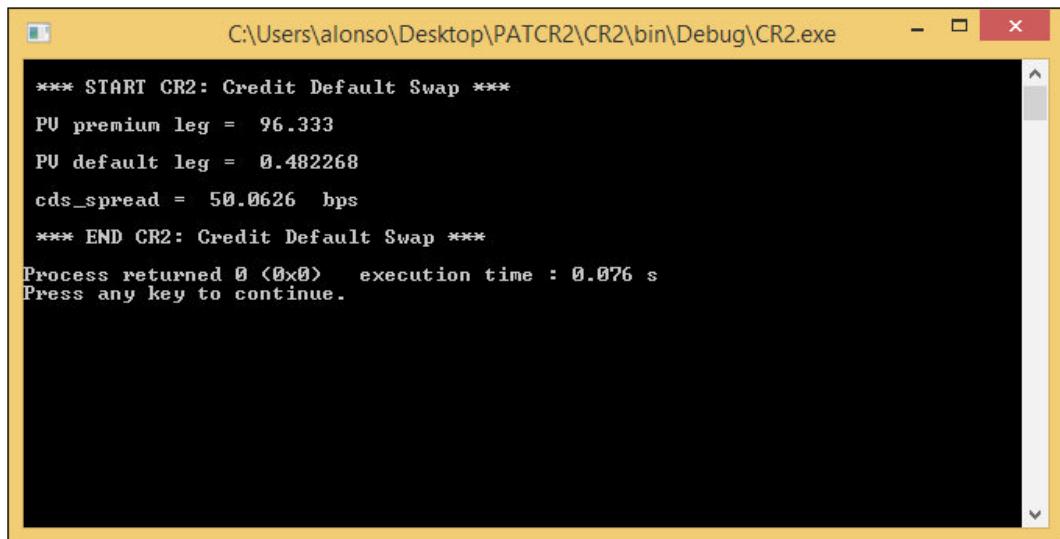
CDS (CR2) screenshot with results

Summary

In this chapter, we have solved two pricing problems in credit derivatives. We have seen a basic example (using structural models) and a more advanced one (using intensity models). There are many possible variations and more complex contracts, but these two are the main families that will give you an idea of how to go forward in this fascinating asset class. This concludes our survey of examples implementing different types of financial derivatives in C++.

A
C++ Numerical Libraries for Option Pricing

Implementing financial derivatives in C++ could be a complex task. As we have shown in this book, it requires knowledge not only of the mathematical models and numerical methods required for their implementation in the forms of C++ code, but it also requires the use of reliable support mathematical and financial libraries. For example, when you need to obtain random samples from a standard normal distribution or when you need to invert a matrix. In these cases, instead of implementing these algorithms ex novo, what we can do is make use of numerical libraries that exists for this purpose. These contain algorithms that have been used for many years and therefore have been validated by many users before. Using these libraries will significantly accelerate our implementation of advanced pricing models. Some examples of these libraries are mentioned in the upcoming sections.

Numerical recipes

License: Commercial.

Website: `http://www.nr.com`.

A collection of widely used and reliable set of C++ numerical routines can be found in the book "Numerical Recipes: The Art of Scientific Computing, 3rd Edition". These set of routines are regarded as the "gold standard" by the top universities and research institutions around the world. There is an excellent associated website that can be found at `http://www.nr.com/`. The book contains the description of the theoretical background of the routines and it gives access to the C++ code. There are more than 400 C++ numerical routines covering topics, such as *Solution of Linear Algebraic Equations*, *Matrix Algebra*, *Interpolation and Extrapolation*, *Integration*, and *Random Numbers*.

Financial numerical recipes

License: Free/GNU.

Website: `http://finance.bi.no/~bernt/gcc_prog/.`

This website contains a large number of very useful C++ numerical and financial programs that have been developed by Bernt Arne Odegaard. They follow the ANSI C++ standard and have a large accompanying manual named *Circa* (250 pages) with the formulas used and the references involved. This library can be found at `http://finance.bi.no/~bernt/gcc_prog/.`

The QuantLib project

License: Free/GNU.

Website: `http://quantlib.org/.`

The QuantLib project is a large project offering software for Quantitative Finance. It has been used for modeling, trading, and risk management in the financial sector. The software is written in C++ and has been subsequently exported to various languages such as C#, Objective Caml, Java, Perl, Python, GNU R, Ruby, and Scheme. QuantLib has many useful tools including yield curve models, solvers, PDEs, Monte Carlo (low-discrepancy), exotic options, VAR, and so on.

The Boost library

License: Free/GNU.

Website: `www.boost.org.`

The Boost project offers peer-reviewed portable C++ source libraries that are freely available under GNU GPL. These libraries have been created with the intention of making them useful and usable across a broad spectrum of applications. Ten Boost libraries are included in the C++ Standards Committee's Library Technical Report (TR1) and in the new C++11 Standard. Examples include Accumulators, Array, Chrono, Filesystem, Geometry, Math, Math/Statistical Distributions, and MPI.

The GSL library

License: Free/GNU.

Website: `www.gnu.org/s/gsl/`.

The **GNU Scientific Library** (**GSL**) is a numerical library for C and C++. The library provides a large variety of mathematical numerical routines, including random number generators, special functions, and least-squares fitting. There are over 1000 functions in total. Examples of the subject areas covered by the library include Complex Numbers, Roots of Polynomials, Special Functions, Vectors and Matrices, Permutations, Linear Algebra, Eigensystems, Fast Fourier Transforms, Quadrature, Random Numbers, Quasi-Random Sequences, Statistics, Histograms, and Monte Carlo Integration.

B

References

Chapter 2

- Garman, M.B., and S.W Kohlhagen. "Foreign Currency Option Values". Journal of International Money and Finance 2, 231-237. 1983.

- Rebonato R. *Interest-Rate Option Models: Understanding, Analysing and Using Models for Exotic Interest-Rate Options*. 1998. Wiley.

- D., Brigo, and Mercurio F. "Interest Rate Models: Theory and Practice, 2nd Edition". Springer Finance. 2006.

- Pelsser, Antoon. *Efficient Methods for Valuing Interest Rate Derivatives*. Springer. 2000.

- Brace, A., D. Gatarek, and et M Musiela."The Market Model of Interest Rate Dynamics". Mathematical Finance. Vol. 7, No. 2, 127-154. 1997.

- Jarrow, Robert A., and Stuart Turnbull. "Pricing Derivatives on Financial Securities Subject to Credit Risk". The Journal of Finance. Vol. 50. March 1995.

- Jarrow, R., D. Lando, and S. Turnbull. "A Markov Model of the Term Structure of Credit Risk Spreads". Review of Financial Studies. Vol. 10. 481–523. 1997.

- Duffie, D., and K. Singleton. "Modeling Term Structures of Defaultable Bonds". Review of Financial Studies. Vol. 12. 687-720. 1999.

- Merton, Robert C. "On the Pricing of Corporate Debt: The Risk Structure of Interest Rates". The Journal of Finance. Vol. 29, No. 2, pp. 449-470. May 1974.

Chapter 3

- Glasserman, Paul. *Monte Carlo Methods in Financial Engineering*. Springer. 2003.

- Wilmott, Paul, Sam Howison, and Jeff Dewynne. *The Mathematics of Financial Derivatives: A Student Introduction*. Cambridge University Press. 1995.

- Cox, J. C., Ross S. A., and Rubinstein M. "Option pricing: A simplified approach". Journal of Financial Economics 7 (3), 229. 1979.

- Black, F., and M. Scholes. "The Pricing of Options and Corporate Liabilities". Journal of Political Economy 81 (3), 637-654. 1973.
- Kloeden P.E., and Platen E. Eckhard. Numerical Solution of Stochastic Differential Equations (Stochastic Modelling and Applied Probability). Springer. 1992.

Chapter 4

- Black, Fischer, and Myron Scholes. "The Pricing of Options and Corporate Liabilities". Journal of Political Economy 81 (3), 637-654. 1973.
- Box, G. E. P., and Mervin E. Muller. *A Note on the Generation of Random Normal Deviates*. The Annals of Mathematical Statistics. Vol. 29, No. 2, pp. 610–611. 1958.
- Higham, Desmond. *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*. Cambridge University Press. 2004.
- Wilmott, Paul. *Paul Wilmott on Quantitative Finance, 2nd Edition*. Wiley. 2006.

Chapter 5

- Garman, M.B., and S.W Kohlhagen. "Foreign Currency Option Values". Journal of International Money and Finance 2, 231-237. 1983.

Chapter 6

- Brace, A., D. Gatarek, and et M Musiela."The Market Model of Interest Rate Dynamics". Mathematical Finance. Vol. 7, No. 2, 127-154. 1997.
- Pelsser, Antoon. *Efficient Methods for Valuing Interest Rate Derivatives*. Springer, 2000.

Chapter 7

- Merton, Robert C. "On the Pricing of Corporate Debt: The Risk Structure of Interest Rates". The Journal of Finance. Vol. 29, No. 2, pp. 449-470. May 1974.
- Jarrow, Robert A., and Stuart Turnbull. "Pricing Derivatives on Financial Securities Subject to Credit Risk". The Journal of Finance. Vol. 50. March 1995.

Appendix A

- Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. "Numerical Recipes: The Art of Scientific Computing, 3rd Edition". Cambridge University Press. 2007.

Index

Thank you for buying
Advanced Quantitative Finance with C++

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.
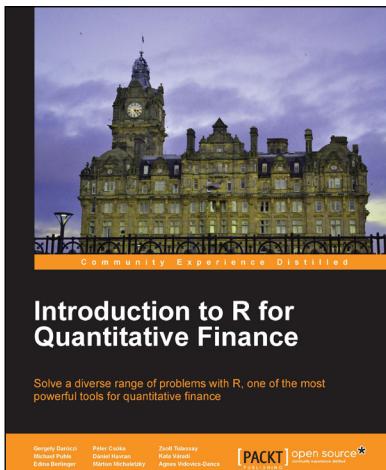
Introduction to R for Quantitative Finance

ISBN: 978-1-78328-093-3   Paperback: 164 pages

Solve a diverse range of problems with R, one of the most powerful tools for Quantitative finance

1. Use time series analysis to model and forecast house prices.

2. Estimate the term structure of interest rates using prices of government bonds.

3. Detect systemically important financial institutions by employing financial network analysis.

F# for Quantitative Finance

ISBN: 978-1-78216-462-3   Paperback: 286 pages

An introductory guide to utilizing F# for quantitative finance leveraging the .NET platform

1. Learn functional programming with an easy-to-follow combination of theory and tutorials.

2. Build a complete automated trading system with the help of code snippets.

3. Use F# Interactive to perform exploratory development.

4. Leverage the .NET platform and other existing tools from Microsoft using F#.

Please check **www.PacktPub.com** for information on our titles
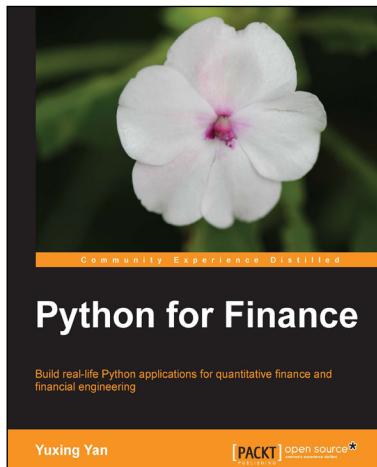
Python for Finance

ISBN: 978-1-78328-437-5   Paperback: 408 pages

Build real-life Python applications for quantitative
finance and financial engineering

1. Estimate market risk, form various portfolios,
   and estimate their variance-covariance matrixes
   using real-world data.

2. Explains many financial concepts and trading
   strategies with the help of graphs.

3. A step-by-step tutorial with many Python
   programs that will help you learn how to
   apply Python to finance.
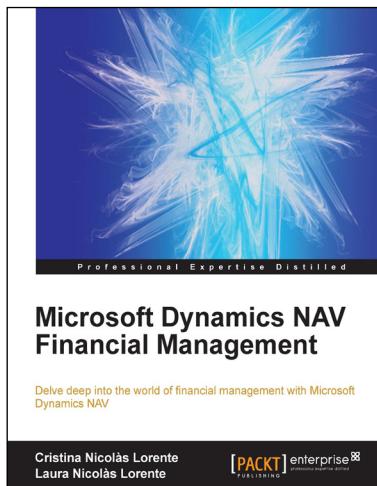
Microsoft Dynamics NAV Financial Management

ISBN: 978-1-78217-162-1   Paperback: 134 pages

Delve deep into the world of financial management
with Microsoft Dynamics NAV

1. Explore the features inside the sales and
   purchases areas as well as functionalities
   including payments, budgets, cash flow,
   fixed assets, and business intelligence.

2. Discover how the different aspects of Dynamics
   NAV are related to financial management.

3. Learn how to use reporting tools that will help
   you to make the right decisions at the right time.

Please check **www.PacktPub.com** for information on our titles