impa **Instituto Nacional de Matemática Pura e Aplicada**

Convertible Bond Pricing: A Monte Carlo Approach

Author: **Leandro Amato Loriato**

Advisor: Maria Rodríguez Nogueiras
Co-Advisor: Jorge Passamani Zubelli

Rio de Janeiro
May, 2014

Para minha amada, Lenita.

Agradecimentos

Em primeiro lugar, agradeço a Deus, por todas as bençãos que me concedeu e concede todos os dias da minha vida.

Agradeço aos meus pais, Umbelino e Ana, pelo seu amor incondicional e por sempre frisarem a importância dos estudos.

Agradeço aos meus familiares e amigos, que souberam entender os momentos de ausência em diversos eventos, encontros e confraternizações, quando tive de abrir mão da companhia maravilhosa de vocês para cumprir as tarefas e obrigações do curso. Agradeço em especial à minha namorada Lenita, quem mais sofreu com minhas ausências e compromissos do curso, mas continuou me amando muito, sempre me apoiando e incentivando ao longo do curso, tanto nos momentos felizes quanto nos difíceis.

Agradeço aos meus orientadores Maria Nogueiras e Jorge Zubelli, por me orientarem e ajudarem na confecção deste trabalho.

Agradeço aos companheiros do IMPA, em especial aos amigos mais próximos, Romeu Dellazeri, Lucas Barcellos, Wanderson Costa, Douglas Vieira e Diogo Gobira, pelas constantes conversas, trocas de ideias, momentos de descontração e apoio mútuo. A convivência com vocês nestes anos aliviou bastante este árduo curso.

Agradeço aos professores mais próximos, Hugo De La Cruz, Vinícius Albani, Welington de Oliveira e Luca Mertens, pelo incentivo e pelas diversas dicas teóricas e práticas nos vários tópicos relacionados a este curso, Matemática Pura e Finanças em geral.

Agradeço ao Instituto Militar de Engenharia (IME), minha alma mater, pela sólida formação em matemática e computação, que certamente me ajudou muito durante este mestrado.

Por fim, agradeço ao Instituto Nacional de Matemática Pura e Aplicada (IMPA), pela honra de poder cursar o mestrado em tão nobre e renomada instituição.

Abstract

Convertible Bonds are interesting hybrid instruments with debt- and equity-like features that have received increasing attention for the last years, especially after the sub-prime mortgage crisis in 2008. This work aims at presenting the main concept behind those instruments, its related features and pricing issues, exhibiting in a constructive manner, from simple products to complex ones, how one may model and price them.

To deal with the possibility of American exercises, we implement least-squared and hedged Monte Carlo pricing methods. A clear, flexible, extensible and ready-to-use code implementation for the proposed pricing framework is provided together with some examples of contracts. A discussion of attained numerical results is also presented.

Keywords: convertible bond, least-squared Monte Carlo, hedged Monte Carlo

Resumo

Debêntures Conversíveis são interessantes instrumentos híbridos com características de títulos de dívida e de ações que têm recebido atenção crescente nos últimos anos, especialmente após a crise imobiliária americana em 2008. Esse trabalho tem por objetivo apresentar o conceito principal por trás desses instrumentos, suas características e dificuldades de precificação, exibindo de forma construtiva, de produtos simples a outros mais complexos, como alguém consegue modelar e precificá-los.

Para lidar com a possibilidade de exercícios Americanos, implementamos os métodos de precificação de Monte Carlo com mínimos quadrados e com cobertura de risco. Uma implementação clara, flexível, extensível e pronta para uso para o framework de precificação proposto é apresentada com alguns exemplos de contratos. Uma discussão de resultados numéricos encontrados também é apresentada.

Palavras-chave: debêntures conversíveis, Monte Carlo com mínimos quadrados, Monte Carlo com cobertura de risco

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In the last few years, specially after american subprime mortgage crisis and its following repercussions all over the world in the last quarter of 2008, the tide of uncertainty and risk aversion has led to a rising interest in fixed income instruments[1], specially the so-called **Convertible Bonds**.

Convertible Bonds are hybrid instruments with debt- and equity-like features. Although there is no uniformity in this type of instrument and the exact features depend upon contract terms, they are bonds that, besides paying periodical coupons, also grant the investor the right to convert it, in pre-determined time periods, into an equivalent amount of the underlying equity. Some put and call optionalities are also commonly observed, granting the investor and issuer each the right to, in certain pre-determined periods, end the contract before its maturity.

Despite the growing interest and the numerous works about equities and bonds separately, surprisingly there are few works about Convertible Bonds.

1.1 Motivation

The problem that we will focus in this work is the modelling and fair pricing of Convertible Bonds, i.e. its price under arbitrage free assumptions.

The study of this problem is motivated by the following reasons:

- The correct pricing of these instruments allows one to identify, avoid, limit or even explore arbitrage oportunities[2];

[1] In a 2011 report [Com11], McKinsey & Company highlights that outstanding global debt had reached US$93 trillion, almost twice the capitalization of global equity market, US$54 trillion.

[2] There's a common belief that Convertible Bonds market practiced prices are, in average, below those generated by theoretical models, possibly generating arbitrage opportunities. Some works, like [AKW08], however, do not support this belief.

- The new international accounting standard, International Financial Reporting Standards (IFRS)[IFR], that is gradually been adopted by many countries around the world[3], requires companies to mark their assets by their market value, effectively demanding them to know how to precify these instruments.

As we shall see later in this work, Convertible Bond pricing has its own subtleties and mathematical challenges, depending on the product and its chosen model specific features.

1.2 Goals and Structure of the Work

The goals of this work are:

- To present the concept of a Convertible Bond, its related features and terms and fair pricing issues;

- To exhibit, in a constructive manner, how one may, beginning from more simple products to others more complex, model and price Convertible Bonds using a stock value-based model and Monte Carlo methods;

- To provide a concrete, clear and ready-to-use code implementation for the proposed pricing framework, which can be used and extended by others to treat more general Convertible Bond contracts and also other types of exotic securities.

We shall focus in this work primarily on Monte Carlo methods for pricing, because, as we shall see later, they are reasonably simple to implement, of easy interpretation and easily extendable to cope with more complex products, like those found in market.

This work is organized as follows:

- In Chapter 2, we introduce the Convertible Bond instrument, its main concepts, features and associated issues;

- In Chapter 3, we present Tree methods and Monte Carlo methods, the pricing techniques that will be used in this work;

- In Chapter 4, we discuss the strategy adopted in this work for pricing Convertible Bonds, starting from a more simple product and simplified model, working on towards more complex settings with use of Monte Carlo methods, presenting related results along the text;

- In Chapter 5, we summarize the attained results and conclude the work;

[3] In Brazil, Federal Law 11.638/07 instituted as mandatory IFRS use in financial statements of publicly held companies, keeping it as optional for private companies. Through Statement 14.259/06, Central Bank of Brazil demanded that, from 2010 on, all financial institutions must issue their statements in accordance with IFRS.

- In Appendix A, the implementation approach and developed code are presented, along with the discussion of implementation specific issues

Chapter 2

Convertible Bond Concepts

In this chapter, we introduce the Convertible Bond security, presenting its main features and associated issues.

In Section 2.1, we present the definition of a Convertible Bond, present some historic perspective behind it, an insight into nowadays market and some of the compelling reasons parties tend to look for it.

Next, in Section 2.2, the most relevant features/terms of a Convertible Bond's contract are presented.

Finally, in Section 2.3, we discuss the issues found while trying to model and price Convertible Bond's, some of which we will address in the models presented later in Chapter 4.

2.1 Basics

To grasp the basic idea behind a Convertible Bond, one must first recall the concept of a **(corporate) bond**: a (corporate) bond is a security issued by a firm (**issuer**) that entitles the **holder** (sometimes, called the **investor**) with the right to receive the bond's **face value** at a final date (**maturity time**) and interest rate payments (**coupons**) over the face value in a predetermined (not necessarily, but usually regular) fashion until maturity.

From holder's point of view, it's a **fixed income instrument**: an instrument that he expects to receive periodic payments, vinculated to interest rates. From this remark, it's pretty obvious that the bond payments, and as a consequence, its value are influenced by interest rate term structure.

From issuer's point of view, a bond is a **debt security**: it's used to captate money from the market, as a kind of a loan. In exchange, the issuer promises to repay the face value as well as the agreed coupons. From this remark, it's quite easy to see that the bond payments and

value are also influenced by the issuer's credit risk[1].

After introducing the concept of a bond, the concept of a **Convertible Bond** is fairly straightforward and given by its essential feature: it's a bond for which the holder has the right to, at prespecified times and, possibly, under some prespecified restrictions, give up future coupons in exchange for a certain number of issuer's stocks (**parity**). This procedure is called **conversion** of the bond, which gives name to the security[2]. Because of these equity-like and debt-like features, it's called an hybrid security.

The investor's decision to convert is related to whether or not he/she expects firm's stock price to rise farther in the future than the value to be received from bond's coupons. From this remark, it's clear that the convertible bond's value, besides of other factors that affect bond's value, is also affected by the stock price and its volatility.

According to a Credit Suisse's 2012 white paper [AG12], the market of Convertible Bonds reached in 2011 a total of US$ 500 billion, been today primarily dominated by USA issuers (43%), followed by European issuers (29%). There has also been observed a trend of issue growth in Asia and America.

Convertible Bonds are also not a recent security, have now been around for over 150 years. They were first issued in mid-nineteenth century by american railroad companies, in a time where US economy was growing at a fast pace, but capital was not easy to obtain. To captate money to finance their growth, they attracted investors with the possibility of high equity gains as companies tended to grow quickly, but with the safeness of a traditional bond.

Nowadays, Convertible Bonds continue to be an attractive security, not only to investors but also to issuers, for the following reasons:

- **For the investor**:
  - In worst-case scenarios, it provides the safety of a traditional bond. In the best-case scenarios, it can also provide a equity-like rentability;
  - It provides higher yields than those attainable via traditional stock purchases.
- **For the issuer**:
  - Because of its implicit conversion optionality, its paid coupons are usually smaller than those paid in traditional bonds, thus constituting a cheaper captating method than traditional bonds. For a small firm, facing budget constraints, this is a very good feature;

[1] In some countries, including Brazil, people use the term *debenture* interchangebly with *bond*. In many other countries, however, the term *debenture* is used only for bonds with long-term maturity and/or whose firm's credit risk is negligible. To avoid confusion, we will only use the term *bond* throughout this work.

[2] In the market, some call this type of convertible bond a **Vanilla Convertible Bond**, as there is another type of convertible bond, called **Mandatory Convertible Bond**, which obliges the holder to convert it at maturity. In this work, we shall deal only with Vanilla Convertible Bonds.

– As opposed to traditional stock issue, it allows a firm to delay diluition, as stocks may be emitted along the bond's lifetime and only when/if the firm grows in a steady pace and confirms good future prospects. For small, private companies, this is critically important, as it allows the firm to formalize its governance structure before stock issuance.

From a portfolio optimization and risk perspective, having a Convertible Bond in a portfolio seems to also offer the following advantages [AG12]:

- It provides good risk diversification;
- It reduces the portfolio's Expected Shortfall.

Historical data also indicates, in the last few years, a higher rate of return with lower volatility for Convertible Bonds when compared to stock shares, as shown in the following graph of Swiss market [AG12]:

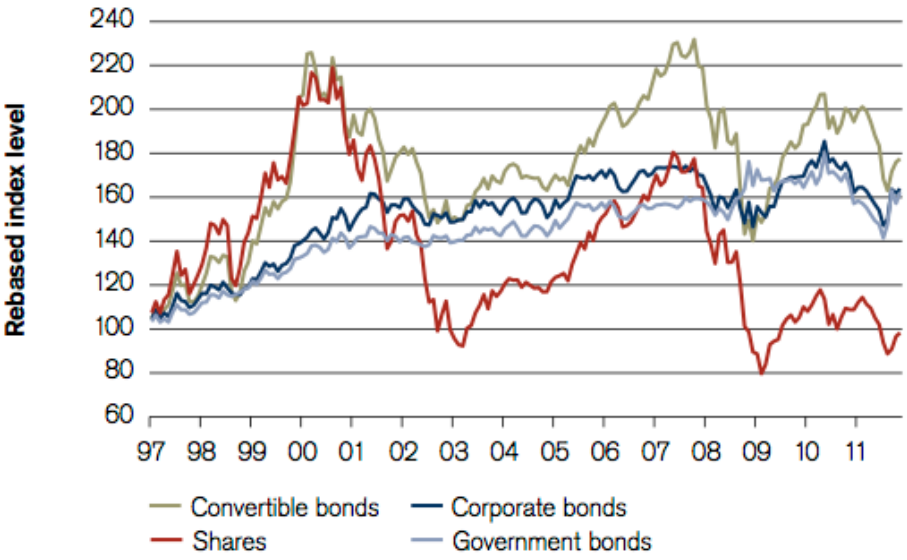

Figure 2.1: Historical returns in Swiss market in the last few years.
Source: Bloomberg, Credit Suisse AG, 2011

For all this, we can no doubtly say that Convertible Bonds are regarded nowadays are a creative and very attractive security, and will continue to be so and have an important role in market in the forthcoming years.

2.2 Features/Terms

As Convertible Bond's contracts are not standardized ones, but in fact very heterogeneous in practice, the specific features of a given convertible depends upon the exact terms of its prospectus. Besides the common bond's terms (interest rate, coupon frequency, first

coupon date, nominal value, maturity date), there are, though, features/terms which are more commonly observed in these contracts. Below we enlist the most relevant ones:

Conversion Provisions

A **conversion provision** is an obligatory clause which specifies at which conditions may take place. The following terms are commonly found in conversion provisions:

Conversion Dates

The **conversion dates** term specifies in which period the investor may exercise the conversion. In practice, any sparse structured exercise period may be specified.

Usually, this term is not informed and it is assumed that conversion is american-styled, i.e. it may take place at any date between issuance and maturity.

Conversion Restrictions

Besides conversion dates, it has become more and more common nowadays to include other **conversion restrictions**, which must be met in order to allow the investor to execute conversion, giving some kind of protection for the issuer against conversions in situations where, whilst it would be very attractive to the investor, it would not be so good for the firm. This specially useful, for example, for small companies, as it may delay stock dilution to a time when the firm has achieved good growth and good governance structure.

In its most common form, a conversion restriction is composed of a level which the firm's stock price must reach in order to trigger conversion. Usually it is a percentage level in terms of the stock price initial value, but could also be a fixed value.

Whenever conversion restrictions are specified, we call the convertible bond a **Contingent Convertible Bond** (sometimes abbreviated as **CoCo Bonds**).

Conversion Ratio/Conversion Price

The **conversion ratio** specifies the number of stock shares into which each bond unity (bought at a specified **nominal value**) can be converted. Typically, it remains fixed throughout all bond's lifetime, but may be adjusted by certain clauses (see **reset clause** below).

Instead of specifying a conversion ratio, the prospectus may specify a **conversion price**. This is the price for which stock shares would be bought upon conversion.

Specifying a conversion ratio or a conversion price is equivalent and they are related as follows:

$$conversion\ price = (nominal\ value / conversion\ ratio)$$

Conversion ratio x conversion price equivalence Consider a Convertible Bond with a $nominal\ value = 100$ and $conversion\ ratio = 2.25$. We would then have:

$$conversion\ price = 100/2.25 = 44.44$$

At a given time, a Convertible Bond may also be classified with respect to its conversion price as follows:

- If $stock\ price > conversion\ price$, it's said to be **in the money**, because one would rather choose to convert it than buy stocks at the market;
- If $stock\ price < conversion\ price$, it's said to be **out of the money**, because one would rather choose buy stocks at the market than convert it.

As we have seen, conversion ratio and conversion price are dual terms: specifying one implicitly specifies the other. Hence, only one of them must be informed in the prospectus. Without loss of generality, throughout this work, we shall assume that a conversion ratio term is specified.

Conversion Ratio Adjustment Clause

A **Conversion ratio adjustment clause** is an optional clause that specifies that the conversion ratio may be adjusted upwards or downards when certain events occur.

Traditionally, they are created to protect the investor from events that may harm the firm's value and consequently lower its stock price, effectively diminishing the possibilities of conversion. But they could also be used to hedge the issuer in cases in which conversion would be excessively attractive and harm issuer's interests.

The following types of conversion ratio adjustment clauses are more commonly found in Convertible Bond prospectus:

Reset Clause A **reset clause** is the most commonly found and more general conversion ratio adjustment clause. It is an optional conversion ratio adjustment clause that takes place

if the firm's performance happens to be far from what was expected in issue time, with its stock price remaining outside of a specified region, defined in terms of its initial value, for a specified amount of time.

In its most common variant, it provides an lower-bounded region, i.e. it adjusts the conversion ratio upwards if the stock price remains under a specified level, protecting the investor from downward fluctuations. But it could also be used to provide an upper-bounded or box (upper- and lower-bounded) region to also protect the issuer against situation's where stock price rises too much and conversion becomes excessively attractive, harming issuer's interests.

Being defined in terms of the stock price as a proxy for the firm's value and perception in the market, it may account not only for price formation fluctuations but also other firm-related events, such as changes of credit rating, which ultimately will be reflected in the stock price.

Dilution Protection Clause A **dilution protection clause** is a more specific conversion ratio adjustment clause that protects the investor against the event of loss on stock price due to issuer's capital dilution, adjusting the conversion ratio upwards.

Takeover Protection Clause A **takeover protection clause** is a more specific conversion ratio adjustment clause that protects the investor against variations on stock price due to a merger involving the issuer.

Redemption Ratio

The **redemption ratio** specifies how much of the bond's face value may be redeemed at maturity. Usually, we have $redemption\ ratio = 1$, but sometimes a convertible bond may be issued with $redemption\ ratio > 1$, when it's said to be issued **at premium**.

Call Provision

A **call provision** is a clause which gives the issuer the right (but not the obligation) to, in certain prespecified periods, end the Convertible Bond's contract before maturity time, forcing the investor either to redeem it for an **strike price**, which may be a fixed amount of cash or may be set to include the principal and any accrued interest. Also, upon call exercise the investor may be granted the option to convert the bond. If the payoff earned by converting the bond is greater than the strike price earned with the call when not converting it, since the investor is considered to be rational, he/she will be practically forced by the issuer into

converting the Convertible Bond. Because of this, this type of conversion is called a **forced conversion**, in contrast with its **voluntary** counterpart.

As a call provision grants the issuer a right to prematurely end the contract, effectively denying the investor the safety of a regular bond, this clause is often badly perceived by investors. To protect the investor, **call protection clauses** have been also commonly found in call provisions:

Hard Call Clause

A **hard call clause** is a call protection clause which specifies an period of time, usually at beginning of the Convertible Bond's lifespan, in which the issuer may not exercise the call. After this period, the issuer may then exercise the call normally (respecting the prespecified call dates).

Soft Call Clause

A **soft call clause** is call another protection clause which specifies quantitative restrictions, usually the stock price reaching a percentage level of its initial value, which must be met in order to trigger the possibility of call by the issuer. Unless these restrictions are met, the issuer may not exercise the call.

Besides call protection clauses, it is also very common to require the issuer to notify the investor of its intention to call the bond before the call itself takes place. The amount of time prior to the call itself that the notice must be made is called **call notice period**.

Put Provision

A **put provision** is the investor-equivalent of a call provision clause: it gives the investor the right (but not the obligation) to, in certain prespecified periods, end the Convertible Bond's contract before maturity time, forcing the issuer to buy the bond back for an **strike price**, usually a fixed amount of cash.

Contraty to call provision clauses, put provision clauses are very attractive features to the investor. To protect issuers from an excessively attractive put provision, usually exercises dates are restricted to narrow or sparse periods.

To better illustrate the above defined terms, below is presented an example of Convertible Bond's contract:

Example 2.2.1. Convertible Bond's contract example

Regular bond features/terms	
Issue date	30 Jun 2006
Maturity	31 Jul 2013
Face value	ZAR 100
First coupon date	31 Jan 2007
Coupon frequency	2 (semi-annual)
Coupon ratio	5.7%
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: - (american-styled) **Conversion restrictions**: - **Conversion ratio**: 3.64964 **Conversion ratio protection clauses**: -
Call provisions	**Strike price**: principal amount plus accrued interest **Call dates**: - (american-styled) **Hard call clause**: 3 years **Soft call clause**: 130% initial stock price value trigger
Put provisions	- (non-puttable)

Table 2.1: Steinhoff 2013 Convertible Bond

This example was adapted from a Convertible Bond contract from south-african market, presented in [Zad10].

2.3 Pricing Issues

As seen in Sections 2.1 and 2.2, Convertible Bonds are amazingly versatile and flexible debt-equity hybrid instruments. Unfortunately, all this flexibility comes at a price of difficult and troublesome pricing. Below we present the main issues found whilst pricing Convertible Bonds:

Contract uniqueness

Convertible Bonds are instruments very difficult to price because their contracts are often unique: they are not standardized instruments, but highly heterogeneous. They may or may not contain specific clauses and its exacts terms vary wildly, as they are taylored to meet the exact issuer's money captation needs, whilst providing attractive features for the investor. This heterogeneity in Convertible Bond contracts makes it practically impossible to exhibit a generic closed-form solution for their prices, whilst also making very difficult to employ numerical PDE pricing methods, because its pricing problem and boundary conditions are

very difficult to state in a sufficiently general way to encompass all possibilites[3].

A Convertible Bond is also unique in another sense: it is issued at a specific point in time, with specified maturity, when a specific conjuncture is found. Whilst, although very rare, there exists the possibility that the same firm does issue another Convertible Bond with same features and terms, its pricing will not be the same because the main factors affecting its price - stock price, credit risk issues, volatility of the stock and interest rates - will naturally not be the same, as the firm itself would have changed over time, affected by the convertible's issue itself, and so would the market. Thus, it is practically impossible to price Convertible Bond's by similarities with other convertibles, even with the ones issued by the same firm.

Complex features/terms

As seen, any of the possible optionalities of a Convertible Bond contract - conversion, call, put - may have flexible, american-, canary-, verde-, bermudan-styled or any other sparse structured exercise dates. This again makes it practically impossible to exhibit generic closed-form solution for its price and very difficult to use PDE methods. Also, as we shall see later on Chapter 3, regular Monte Carlo methods cannot correctly cope with these types of exercise. Only specially modified Monte Carlo and tree methods may then be used for its pricing.

Also, many of its clauses are very liberal when it comes to its exact terms and may assume complex, path-dependent forms. For example, a Convertible Bond contract may have a reset clause that states that "*the conversion ratio will be adjusted to 0.9 whenever the mean stock price of the last 3 weeks remain above 130% of initial stock price*". This kind of time-dependency makes it quite to difficult to use tree methods, leaving only simulation-based, specially Monte Carlo, methods as flexible enought to cope with these complex terms.

Credit risk issues

One of the main factors affecting the Convertible Bond's price ([BS77]), credit risk issues are quite difficult to model and treat using PDE and tree methods. Some successful approaches make use of Poisson processes and jump-diffusion SDEs in Monte Carlo methods, requiring a more refined stochastic calculus theory.

All these difficulties have made credit risk issues the greatest research topic in the last few years, even more after the Subprime Mortgage Crisis and its repercussions around the world, leading to stricter credit risk regulations in financial market.

[3] As may easily be seen in many works on Convertible Bonds, although in its most general form Convertible Bond's pricing is very difficult to solve with closed-form solutions and PDE methods, there are restricted, simplified and well-defined versions of this problem which do have closed-form and/or PDE solutions.

Firm-related events

Some of the Convertible Bond contract's clauses may depend upon events that are firm related. Even if firm-based models are employed instead of stock-based ones (see 4.1), firm-related events are really difficult to model, specially ones related to changes in its capital structure, like mergers and stock dilution.

To try to mend this, one may choose to treat these events are exhogeneous ones and try to emulate them using jump processes. This method naturally works as good as how well calibrated the rates of events are. So, they usually do not work very well, because these events are very localized in time and do not repeat in the same manner, resulting in little data available for calibration.

Chapter 3

Pricing Methods

Before stepping into Convertible Bond models, we briefly present the main pricing techniques we will be using in this work: Tree methods and Monte Carlo methods.

In Section 3.1, we present the classic Black-Scholes stochastic model, the base model we shall use throughout this work for the asset's stock price dynamics.

In Section 3.2, we present the general idea behind Tree methods and the specific method we will be using throughout this work, the Cox-Ross-Rubinstein tree.

Finally, in Section 3.3, we present the general idea behind Monte Carlo methods, the motivation behind the introduction of backward induction/projection techniques, and the techniques of Least-squared Monte Carlo (LSMC) and Hedged Monte Carlo (HMC), that we will use later in this work.

3.1 Black-Scholes Model

This work shall use the Black-Scholes model as the base model for pricing Convertible Bonds. Initially proposed by Black and Scholes in their seminal paper [BS73] and later formalized by Robert Merton in [Mer73] using Stochastic Calculus, the Black-Scholes Model became one of the most known and used models for fair pricing financial instruments, laying strong and solid foundations for more refined models and shaping the Finance industry in the past 40 years. It was a so important scientific contribution, authors Scholes and Merton earned a Economics Nobel prize in 1997 for it[1].

Whenever using the Black-Scholes Model, it should always be kept in mind that it relies on the following important market assumptions:

- Market is assumed to contain exactly one riskless asset, a zero-coupon bond, with

[1] Black was ineligible because it had died in 1995, but was mentioned as one of its contributors.

an associated risk-free interest rate, and one risky asset, a stock, with an associated uncertainty source. Any money which is not invested in stock must be invested in the zero-coupon bond.

- Investors are not able foresse market events;
- Investors act rationally and in a self-financing way;
- The actions of a single investor do not influence stock prices;
- Investors may borrow and/or lend any ammount of money at risk-free interest rate;
- Investors may buy or sell any ammount of stocks, including fractional ones;
- Short-selling by investors is allowed;
- Market is frictionless: there are no fees or costs involved in transactions;
- Market is complete: every contingent claim is attainable, i.e. may be replicated using a self-financing strategy envolving the riskless and risky assets. As a consequence (see [KK01]), market is arbitrage-free (i.e. there are no arbitrage opportunities) and each contingent claim has only one (fair) price.

Under the aforementioned assumptions, the **Black-Scholes Model** (sometimes also called **Black-Scholes-Merton Model**) is formulated as follows:

Let $S(t) > 0$ be the (risky) asset's stock price at time t, with associated constant volatility σ and paying a constant continuous dividend yield q. Let r also be the deterministic, constant risk-free interest rate. Since the market is complete, it may be proved [KK01] that there is only one Equivalent Martingale Measure, called **Risk-neutral Measure**, denoted $\mathbb{Q}$[2]. Under this probability measure $\mathbb{Q}$, stock price dynamics is governed by the following drift-diffusion Stochastic Differential Equation (SDE)

$$\begin{cases} dS(t) = & S(t)((r-q)dt + \sigma dW^{\mathbb{Q}}(t)) \\ S(0) = & S_0 \end{cases} \tag{3.1.1}$$

where $\{W^{\mathbb{Q}}(t)\}_t$ is a Wiener process modelling the uncertainty source associated with this asset. The interpretation of SDE 3.1.1 is, under risk-neutral perspective, the stock price tends to rise deterministically with constant rate but is disturbed by a random noise scaled by its constant volatility.

It must be noted that SDE 3.1.1 is a fairly simple one and thus has an explicit solution ([KK01]) given by

[2]This is in contrast with the **Real-world Probability Measure** $\mathbb{P}$.

$$S(t) = S_0 exp\left(\left((r-q) - \frac{\sigma^2}{2}\right)t + \sigma W^{\mathbb{Q}}(t)\right) \tag{3.1.2}$$

Thus, we see that the stock price process $S(t)$ is modelled as a **Geometric Brownian Motion** in Black-Scholes Model.

Let $C(t, S(t))$ be the fair price at time t of an european contingent claim that pays $B = h(T, S(T))$ at time $t = T$. This fair price may be stated, under this stochastic pricing framework, as the discounted expected value of the future payoff under risk-neutral measure ([KK01]):

$$\begin{aligned} C(t,S(t)) &= \mathbb{E}^{\mathbb{Q}}\left(\frac{P_0(t)}{P_0(T)}B|\mathcal{F}_t\right) = \frac{P_0(t)}{P_0(T)}\mathbb{E}^{\mathbb{Q}}(B|\mathcal{F}_t) \Rightarrow \\ C(t,S(t)) &= e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}(B|\mathcal{F}_t) \end{aligned}$$

with $P_0(t) = e^{rt}$ being the zero-coupon bond (which happens to be the associated numeráire for $\mathbb{Q}$) and $\mathcal{F}_t = \sigma(W^{\mathbb{Q}}(t))$.

At time $t = 0$, in particular, we have:

$$\begin{aligned} C^* = C(0,S(0)) &= \mathbb{E}^{\mathbb{Q}}\left(\frac{P_0(0)}{P_0(T)}B|\mathcal{F}_0\right) = \frac{1}{P_0(T)}\mathbb{E}^{\mathbb{Q}}(B) \Rightarrow \\ C^* &= e^{-rT}\mathbb{E}^{\mathbb{Q}}(B) \end{aligned} \tag{3.1.3}$$

For some special cases of contingent claims, the above mentioned equation yields a closed-form formula for the fair price of the contingent claim. For more general cases of contingent claims, specially for those with path-dependent payoffs, no closed-form formula is available

A classical and noteworthy example of closed-form formula derived from Equation 3.1.3 is that obtained for an **european call option** with strike price K. Its fair price is formulated as:

$$C^* = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left(S(T) - K\right)^+$$

It can be then showed ([KK01]) that, under the Black-Scholes Model, the above equation gives place to the following closed-form formula, known as the notorious **Black-Schole Formula** for the price of european call option:

$$C^* = S_0 e^{-qT}\Phi(d_1) - Ke^{-rT}\Phi(d_2) \tag{3.1.4}$$

with

$$
\begin{aligned}
d_1 &= \frac{1}{\sigma\sqrt{T}}\left(log(\frac{S_0}{K} + ((r-q) + \frac{\sigma^2}{2})T\right) \\
d_2 &= d_1 - \sigma\sqrt{T} \\
\Phi(x) &= \int_{-\infty}^{x} e^{-\frac{1}{2}z^2} dz
\end{aligned}
$$

This brief overview of Black-Scholes Model shall be sufficient for understanding its use in this work. For a more formal discussion of this model and general security pricing under it, refer to [KK01], [Shr04] and [MR06] .

3.2 Tree Methods

3.2.1 Main Idea

The main idea behind **tree methods** is to approximate the continuous state of a stochastic variable (veat least the asset's price) in a time span ranging from initial time t_0 to final time T using a discrete structure, a **tree**, with each node level representing states distant a Δt time step from previous node level[3].

Starting from an initial node (representing initial time t_0), the possible outcomes of any given node are modelled in the n succeeding nodes, each one of them with an assigned probability to occur.

After building the tree structure, it can then be used to calculate prices of securities. For european-styled securities, the method works as follows:

1. On leaf nodes (those representing final time T), security value is calculated as just the payoff of the security;

2. At any preceding node, the value of the security is the expected value (usually under the risk neutral measure $\mathbb{Q}$) of the possible security values in succeeding nodes.

3. Finally, the value of the security is its value at the initial node.

For american-styled securities (or even bermudan, canary, verde, or any other security with sparse exercise structure), the structure is pretty much the same, except on step , if the

[3] In fact, if one's model only has one stochastic variable, the structure is a 2-dimensional structure, a tree. If more than one stochastic variable is involved, each node maps a possible outcome for the entire set of stochastic variables, which result in a more general discrete structure, a kind of multidimensional tree known as **lattice**.

security may be exercised at that moment, its value will be maximum between the averaged value of succeeding nodes and the intrinsic value obtained if exercise takes place.

Tree methods yields as a result an exact value for the security price. Approximating the continuous structure of stochastic variables state incurs in an discretization error. Smaller Δt values (or equivalently, bigger number of time discretization steps) lead to more precise values but with also more computational cost. As we will be see as a very recurring trend, this poses a challenge whilst choosing Δt, to balance precision level with computational cost.

3.2.2 Binomial Trees

Although the main idea is pretty much the same behind every tree method, they differ usually in how many n succeeding nodes each node has and what are the probabilities of ocurrence of each node.

The most popular tree in mathematical finance is the **binomial tree**, in which one have only two nodes succeding each node:

- One for which the stochastic variable's value would rise from S in current node to uS with a probability p;
- Other for which the stochastic variable's value would fall from S in current node to dS with a probability $1 - p$

A commonly found and useful (but not required) property of a binomial tree is that $udS = duS$, that is, the notes after an upward movement and downard movement in any particular order are exactly the same. This kind of tree are called **recombinant trees**.

Assuming a recombinant binomial tree, using combinatorial reasoning is not difficult to show that, after some $k\Delta t$ time steps from a node with initial value S, an specific node after i upward movements and $j = k - i$ downward movements would have a value

$$S_{i,j} = u^i d^j S = u^i d^{k-i} S$$

Thus, in a recombinant binomial tree, any node may be uniquely identified by this notation $S_{i,j}$. Moreover, we see that, having u, d and p well defined, it is quite easy to build a recombinant binomial tree.

Suppose we follow the Black-Scholes Model (see Section 3.1). Let $C_{i,j}$ be the price of an european-style security at node with stock price $S_{i,j}$. It is also uniquely identified by this notation as its underlying node is unique identified. The security price $C_{i,j}$ at an specific node is then given by

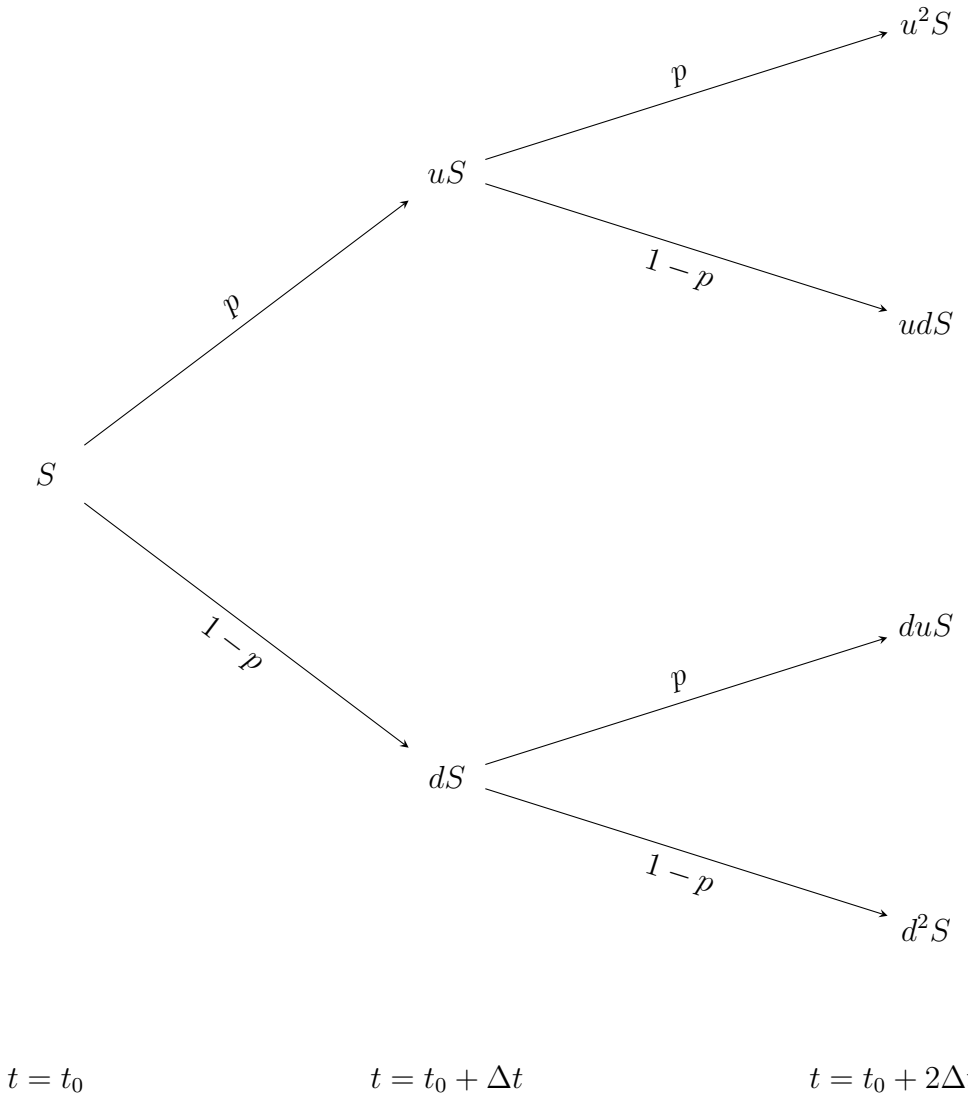

Figure 3.1: 2-period non-recombinant binomial tree

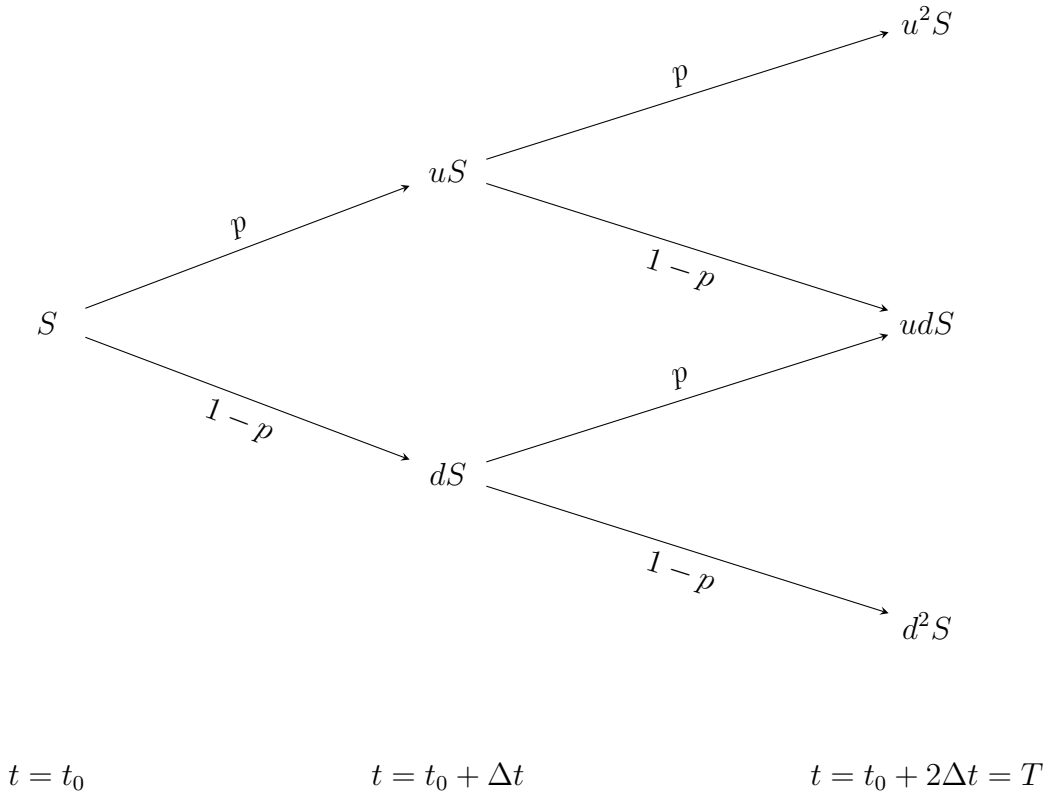

Figure 3.2: 2-period recombinant binomial tree

$$C_{i,j} = e^{-r\Delta t}(pC_{i+1,j} + (1-p)C_{i,j+1})$$

that is, an risk-neutral mean of the prices if the stock goes up or goes down. For an american-style security, the price $C_{i,j}$ would then be

$$C_{i,j} = max(e^{-r\Delta t}(pC_{i+1,j} + (1-p)C_{i,j+1}, h(S_{i,j}))$$

which is, as seen, a maximum between the intrinsic value obtained by exercising it at current node and the risk-neutral mean of the prices of succeeding nodes.

For a more detailed description on binomial trees, refer to [Hul09].

We move on to present the most commonly used binomial tree, the Cox-Ross-Rubinstein Binomial Tree, which will be the one we will use in this work.

Cox-Ross-Rubinstein Binomial Tree (CRR Tree)

Introduced in the seminal article [CRR79], the **Cox-Ross-Rubinstein Binomial Tree (CRR Tree)** is a very popular tree method in mathematical finance. It consists in a binomial tree in

which parameters u, d and p as set as

$$u = e^{\sigma\sqrt{\Delta t}}$$
$$d = \frac{1}{u}$$
$$p = \frac{(e^{r\Delta t} - d)}{(u - d)}$$

This parameters were chosen ([CRR79]) so as to better approximate, in terms of mean and variance, the dynamics given by SDE (3.1.1).

It has also been shown in [CRR79] that the price given by a CRR tree asymptoptically converges to the ones given by original Black-Scholes framework for vanilla options. This useful property explains why this pricing method is one of the most popular one amongst financial industry.

The CRR tree can also be extended to deal with important features found in the market, such as continuous and discrete dividend yields. For a more detailed description on this extensions, refer to [Hul09].

3.2.3 Other Tree Types

Besides binomial trees, more general trees can also be found in mathematical finance context.

Trinomial trees (which have 3 succeeding nodes for each node) and non-regular trees (which have variable number of succeeding nodes for any number) are very useful for capturing events such as credit risk defaults, which cannot be modelled using a simple a binomial trees. But this flexibility also turns these trees in general more difficult to build than binomial ones.

Examples of such more general trees include **Hull Trees** and **Hung-Wang Trees**. As we shall not use these other tree types in this work, we will not dwelve much into these tree models. For more details, see [Zad10] and [Hul09].

3.3 Monte Carlo Methods

3.3.1 Regular Monte Carlo

First, we present the definition of a (regular) Monte Carlo method:

Definition 3.3.1 (**Monte Carlo method**). Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and X be a random variable defined in this space, with known distribution.
Let $\{x_i\}_{i=1}^{M}$ be a sequence of random samples from X, i.e.

$$x_1, x_2, \cdots, x_M \sim X$$

Let $g(.)$ be a functional into $\mathbb{R}$ of random variables defined over the above probability space. A **Monte Carlo method** is a method to approximate $\mathbb{E}(g(X))$. From Law of Large Numbers, if $\mathbb{E}(g(X))) < \infty$ this can be done as:

$$\mathbb{E}(g(X)) \simeq \frac{1}{M}\sum_{i=1}^{M} g(x_i) = \bar{g}_M$$

Moreover, from Central Limit Theorem, we have that:

$$\bar{g}_M \xrightarrow{d} N(\mathbb{E}(g(X)), \frac{1}{M}\mathsf{Var}(g(X)))$$

In this work, we are interested in using Monte Carlo methods to obtain the fair price of securities. To note how this can be done, consider the following scenario:

Suppose we have a security which pays $h(T, S(T))$ at time $t = T$. Suppose also that we follow the Black-Scholes Model (see Section 3.1). Using Equation 3.1.3 and noting that

$$W(t + \Delta t) - W(t) \sim N(0, \Delta t) \sim \sqrt{\Delta t} N(0, 1) \quad ,$$

since $W(t)$ is a Brownian motion process, we may calculate the price of this security by using Monte Carlo method to estimate $E^{\mathbb{Q}}(B) = E^{\mathbb{Q}}(h(T, S(T)))$ as follows ([Iac09]):

1. Choose adequate parameters N_{MC} (number of simulation paths) and N_t (number of discretization steps used in each path);

2. Compute $\Delta t = T/N_t$;

3. For $i = 1, \cdots, N_{MC}$ do:

   (a) For $j = 1, \cdots, N_t$ do:

i. Generate a (pseudo)random number sample z from standard Gaussian distribution $N(0,1)$;

ii. Set $t_j = t_{j-1} + \Delta t$;

iii. Set $W(t_j) = W(t_{j-1}) + z\sqrt{\Delta t}$;

iv. Compute $S(t_j)$ from a discretized version of Equation 3.1.1, using some **finite difference discretization scheme** (see below)

(b) Compute $B_i = h(t_{N_t}, S(t_{N_t}))$

4. Finally, compute $C^* = e^{-rT}\mathbb{E}^{\mathbb{Q}}(B) \simeq \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} B_i$

As seen previously (3.3.1), C^* is better aproximated as $N_{MC} \to \infty$. But choosing bigger values for N_{MC} leads to simulating more paths, which increases the computational cost of Monte Carlo simulation. Therefore, N_{MC} must be chosen as to balance computational cost and precision level on C^* estimation.

As both stochastic processes $S(t)$ and $W(t)$ in Equation 3.1.1 are continuous in t, by discretizing this equation[4] with N_t steps, we are incurring in an error. Choosing bigger values for N_t leads to more fine-grained simulations, with less error but more computational cost. Like with N_{MC}, choosing N_t also involves balancing computational cost and precision level in the simulation.

For discretizing Equation 3.1.1 there are several finite difference schemes, relying heavily on **Ito-Taylor Expansion**, an Ito-Calculus variant of classic Taylor Expansion. It is not in the scope of this work to discuss Ito-Taylor Expansion, but if needed one may refer to [KP11] for a formal discussion on this subject.

We present here the two discretization schemes used in this work:

Euler-Maruyama Scheme

Euler-Maruyama scheme[5], sometimes simply called **Euler scheme**, is a very simple discretization scheme. In this scheme, an SDE of the form

[4] One could easily question the need to discretize Equation 3.1.1 since it has an explicit solution given by 3.1.2 and, if payoff $h(.,.)$ only depends on current value of $S(t)$, only the value of $S(T)$ need to be simulated using the above solution and could be used directly to compute C^*. Hence no time step Δt smaller than T would be needed and therefore there would be no need to discretize Equation 3.1.1 at all.
Although this is correct, in a more general case one may have a more complex, non-lognormal dynamics for $S(t)$, for which its describing SDE has no explicit solution, and/or a path-dependent payoff. When this happens, we can no longer make use of a explicit expression for $S(t)$ and may need to use some of its intermediary path values in order to compute the payoff. Hence, in order to provide a method which accounts for this more general case, we include here explicitly a step to discretize the SDE.

[5] Its name comes from it being based on Euler's original method for ODE discretization

$$\begin{cases} dX(t) = & b(t, X(t))dt + \sigma(t, X(t))dW(t) \\ X(0) = & X_0 \end{cases} \tag{3.3.1}$$

is discretized as

$$X_{i+1} = X_i + b(t_i, X_i)(t_{i+1} - t_i) + \sigma(t_i, X_i)(W_{i+1} - W_i)$$

Figure 3.3: Example of 100 Monte Carlo paths generated for stock price evolution following Equation 3.1.1 using Euler-Maruyama scheme, with $S_0 = 100,\ T = 1,\ r = 0.1,\ \sigma = 0.3,\ N_t = 100$

Milstein Scheme

Euler-Maruyama relies on a very simple first-order approximation of Ito-Taylor Expansion of $X(t)$ process, which leads to truncation error. To provide a better approximation, Milstein developed a first-order approximation scheme that includes more terms from Ito-Taylor Expansion.

In **Milstein scheme**, SDE 3.3.1 is discretized as

$$X_{i+1} = X_i + b(t_i, X_i)(t_{i+1} - t_i) + \sigma(t_i, X_i)(W_{i+1} - W_i) + \\ + \frac{1}{2}\sigma(t_i, X_i)\sigma_x(t_i, X_i)((W_{i+1} - W_i)^2 - (t_{i+1} - t_i))$$

It has been shown ([KP11]) that, despite being computationally more costly, Milstein Scheme in general has better convergence properties and is numerically more stable than Euler-Maruyama.

For a discussion about other discretization schemes, more rigorous numerical analysis and implementation tips, refer to [KP11], [Iac09] and [Gla04].

One final remark must be made on Monte Carlo methods: in contrast to Tree methods, because of (pseudo)random number generation, Monte Carlo methods have a probabilistic nature, yielding not an exact value, but rather an estimated value and a confidence interval, given by Central Limit Theorem.

The asymptotic convergence of Monte Carlo methods guarantees that, for sufficiently large N_{MC} values, the true value we want to estimate will be at the given confidence interval.

To achieve more precise (i.e. narrower) confidence intervals, there are **variance reduction techniques** like **antithetic sampling**, **control variate** and **importance sampling** (see [Iac09] and [Gla04] for details). For the sake of simplicity and to focus solely on plain Monte Carlo algorithms, this work does not make use of any variance reduction technique. However, if desired, straightforward approaches like antithetic sampling should be fairly easy to incorporate.

3.3.2 Projection/Backward induction techniques

Now, consider an american security. Assuming a rational strategy, the holder will exercise it only when its current intrinsic value is greater than the expected value of holding the security for an additional Δt (**continuation value**). Thus, at a given time $t \in [0, T]$, the price of an american security may also be stated as ():

$$C(t) = \max(B(t), \mathbb{E}^{\mathbb{Q}}(C(t + \Delta t)|\mathcal{F}_t)) \tag{3.3.2}$$

To obtain the fair price of american securities at time $t = 0$, a naive approach would be to use Equation 3.3.2, rolling a Monte Carlo simulation to obtain $\mathbb{E}^{\mathbb{Q}}(C(t + \Delta t)|\mathcal{F}_t)$.

But as $C(t + \Delta t)$ may also be expressed by Equation 3.3.2 as a function of another continuation value, this incurs in a recursive problem, requiring many other Monte Carlo simulations, which makes this approach computationally unfeasible.

Thus, to correctly price american securities using Monte Carlo methods, one must resort to more efficient techniques.

One of the techniques to correctly price american securities in a more efficient manner is the so-called **projection** or **backward induction technique**. It consists in:

1. With a time discretization step of Δt, simulate the evolution of stochastic components through regular Monte Carlo;

2. Calculate the price of the security in a backward manner: starting from T,where the its price $C(T)$ is known, project the calculated price $C(t + \Delta t)$ into a selected basis of functions at time t to calculate $C(t)$

As we will see, both Least-squared Monte Carlo ([LS01]) and Hedged Monte Carlo ([PBS01]) methods are examples of backward induction techniques.

3.3.3 Least-squared Monte Carlo (LSMC)

In the article [LS01], the authors present a backward induction technique for pricing american securities (specifically, american options), named Least-squared Monte Carlo (LSMC).

The main idea behind LSMC algorithm is, while computing the fair price at time t in a backward manner, in order to calculate $C(t - \Delta t, S_{t-\Delta t})$, to project $C(t, S_t)$ at space L^2. More formally, we have:

$$\begin{aligned} C(t - \Delta t, S_{t-\Delta t}) &= \mathbb{E}^{\mathbb{Q}}(C(t, S_t)|\mathcal{F}_{t-\Delta t}) = \\ &= \underset{g(S_{t-\Delta t}) \in L^2(\Omega, \mathcal{F}_{t-\Delta t}, \mathbb{Q})}{\text{argmin}} \text{Var}^{\mathbb{Q}}\{g(S_{t-\Delta t}) - C(t, S_t)e^{-r\Delta t}\} \end{aligned}$$

But, it may be shown that $\mathbb{E}^{\mathbb{Q}}(g(S_{t-\Delta t}) - C(t, S_t)e^{-r\Delta t}) = 0$. So, the previous equation may be rewritten as:

$$\begin{aligned} C(t - \Delta t, S_{t-\Delta t}) &= \mathbb{E}^{\mathbb{Q}}(C(t, S_t)|\mathcal{F}_{t-\Delta t}) = \\ &= \underset{g(S_{t-\Delta t}) \in L^2(\Omega, \mathcal{F}_{t-\Delta t}, \mathbb{Q})}{\text{argmin}} \mathbb{E}^{\mathbb{Q}}((g(S_{t-\Delta t}) - C(t, S_t)e^{-r\Delta t})^2) \end{aligned}$$

L^2 is a convenient choice for a subspace onto which project $C(t, S_t)$ for 2 basic reasons:

1. To make use of common stochastic modelling techniques, it's common practice to require that functions be measurable and members of L^2. So, the above stated requirement is not so much of a strong requirement;

2. Since L^2 is a Hilbert space, it may be shown that there exists a countable orthonormal basis of functions $\{\chi_j(S)\}_j$ such that $g(S_{t-\Delta t})$ may be expressed as $g(S_{t-\Delta t}) = \sum_j \alpha_i \chi_j(S_{t-\Delta t})$. Thus, the stated optimization problem relies onto a least-squares problem (which explains the method's name).

After clarifying its main idea, the LSMC algorithm may then be fully stated as follows:

1. Simulate N_{MC} paths for the stochastic components with regular Monte Carlo

2. Choose a family of basis functions $\{\chi_j(S)\}_{j=1}^n$

3. For each Monte Carlo path, compute $C(t, S_t)$ initially as the intrinsic value of the security at t

4. For $t = T - \Delta t, T - 2\Delta t, \cdots, \Delta t, 0$ do:

   (a) Find $(\alpha_i)_i$ s.t.
   $\alpha = \underset{\alpha \in \mathbb{R}^n}{\text{argmin}}\ \mathbb{E}((C(t + \Delta t, S_{t+\Delta t})e^{-r\Delta t} - \alpha \cdot \chi(S_t))^2)$

   (b) For each Monte Carlo path, calculate $C'(t, S_t) = \alpha \cdot \chi(S_t)$

   (c) If optionality can be exercised at this moment, for each Monte Carlo path, update $C(t, S_t)$ as:
   $$C(t, S_t) = \begin{cases} C(t, S_t) & , if \quad C(t, S_t) \geq C'(t, S_t) \\ e^{-r\Delta t} C(t + \Delta t, S_{t+\Delta t}) & , \quad otherwise \end{cases}$$

5. Finally, compute the value of the security as $\mathbb{E}(C(0, S_0))$

The convergence of the LSMC algorithm was briefly discussed in the original article. Later on, it was proved and discussed in more detail in [CLP02] and [GY04].

In the original article, for demonstrating the algorithm, the authors use Laguerre Polynomials as basis functions, but highlight that other choices are also possible: Legendre, Chebyshev (1st, 2nd and 3rd type), Hermite, Gegenbauer and Jacobi Polynomials may be used, but also non polynomial-based basis, such as Fourier ones.

In a sense of reduced computational cost and more precise computation (i.e. narrower confidence interval), choosing the best-suited family of basis functions and its optimal dimension is usually a security-dependent concern[6]. Hence, to determine which basis functions and dimension to use pilot tests are usually taken.

[6] In the particular case of american options, in [GY04] the authors succeeded to prove that, given a specified number of Monte Carlo paths, there exists an optimal basis dimension that guarantees the convergence of LSMC algorithm.

3.3.4 Hedged Monte Carlo (HMC)

It's worth notice that LSMC method was designed to work only data given in with risk-neutral measure $\mathbb{Q}$, as values are treated as risk-neutral ones and discounted using it's associated numeráire $P_0(t) = e^{rt}$. To also cope with historical data, in the article [PBS01], the authors propose an extension to LSMC algorithm, named Hedged Monte Carlo (HMC).

The main idea behind HMC algorithm is to, while solving for the projection model in step 4a, to also take into account a basis of hedge functions $\phi_j(S)_j$:

$$(\alpha, \beta) = \underset{\alpha,\beta\in\mathbb{R}^n}{\text{argmin}}\ \mathbb{E}((C(t+\Delta t, S_{t+\Delta t})e^{-r\Delta t} - \alpha\cdot\chi(S_t) - \beta\cdot\phi(S_t)(S_{t+\Delta t}e^{-r\Delta t} - S_t))^2) \quad (3.3.3)$$

Essentially, what HMC method does is the same a trader would do: to price and hedge his option so as to minimize variations on its wealth due to random fluctuations on underlying asset's price. By doing this, the algorithm implicitly transforms data from historical measure into risk-neutral one ([BP04]).

To lower computation burden, authors suggest that a good simplification is to take

$$\begin{cases} \phi_j(S) & = \dfrac{d\chi_j(S)}{dS} \\ \beta_j & = \alpha_j \end{cases}$$

With this, Equation (3.3.3) becomes

$$\alpha = \underset{\alpha\in\mathbb{R}^n}{\text{argmin}}\ \mathbb{E}((C(t+\Delta t, S_{t+\Delta t})e^{-r\Delta t} - \alpha\cdot\chi(S_t) - \alpha\cdot\chi'(S_t)(S_{t+\Delta t}e^{-r\Delta t} - S_t))^2)$$

When not dealing with fat tail scenarios, this is usually a fairly good approximation for the hedge, since the difference between Black-Scholes Δ-hedge and optimal hedging strategy is often small.

Finally, the authors argue that, besides coping with data in historical measure, the proposed algorithm also has the following advantages over LSMC:

- Provides a more precise computation (i.e. a narrower confidence interval) for the true value of the security price
- Provides a value for the strategy to correctly hedge the security

Although not justified in the original article, this remarks are explored in more details in [BP04].

3.4 Comparison of Pricing Methods

After presenting each pricing method, we now summarize their similarities and differences.

Both methods, tree and Monte Carlo ones, are fairly used for pricing securities, quite easy to understand[7] and implement and can correctly cope with american-styled exercises. But tree methods do not correctly cope with path-dependent features (e.g. asian-style payoffs). On the other hand, Monte Carlo methods, because of its simulation-based nature, may easily be extended to cope with these features. Also, Monte Carlo methods are more easily adaptable to cope with exhogenous events, such as exhogeneous or firm-related events (e.g. credit risk defaults, mergers, stock dilution, etc), whilst this is very difficult to do in tree methods, as its capabilities are engraved in the very tree structure.

Computational cost in tree methods rises exponentially with smaller Δt (or equivalently, with more time discretization step), whilst for Monte Carlo methods, it rises in a linear way with greater N_t and gretare N_{MC}.

In general, tree methods are usually faster than Monte Carlo ones. But if its parameter Δt is sufficiently small, the number of tree nodes becomes very big and computation cost rise too much, and tree method may become slower than equivalent Monte Carlo ones.

A final important difference we saw is with respect to result values yielded by each method: tree methods yield only an estimated value for the price, whilst Monte Carlo ones yield an estimated value and a confidence interval.

Finally, summarizing our discussion, tree methods are faster and yield exact values, which makes them the reference pricing method for simple products, which do not have path-dependent features nor have exhogeneous or firm-related events. When these last features are present, despite its worse performance, Monte Carlo methods are usually preferred since they are flexible enough to be customized to cope with these particularities.

As was seen in Section 2.3, Convertible Bond contracts are very heterogeneous in practice, tailored for specific needs of the issuer, with unusual and specific clauses being fairly common. For all this, because of its flexibility and extensibility, we will focus primarily on Monte Carlo methods throughout this work, comparing with tree methods whenever possible.

[7] Because of its simulation nature, with each path independent from the others and reproducing a possible realization in the market, and because obtained results are more easily traced back into their path inputs, it may be argued that Monte Carlo methods are easier to understand than tree ones.

Chapter 4

Convertible Bond Modelling and Pricing

In this chapter, we will cover modelling and pricing of Convertible Bonds in a constructive manner.

In Section 4.1, we will review some of the pricing issues exposed in Section 2.3 and discuss models found in current literature, discussing the pros and cons of each approach. Finally, we will present the chosen approach for this work.

In Sections 4.2, 4.3, 4.4 and 4.5, beginning from a more simple product and simplified model towards more complex products and models, we discuss each setting, implementing and analyzing it.

4.1 Convertible Bond Modelling Review

In this section, we shall briefly review some Convert Bond modelling and pricing approaches found in the literature, contextualizing this work among these approaches.

From a modelling point of view, 2 main families of Convertible Bond models have appeared: firm value-based models and stock price-based models. **Firm value-based models** seek to map investor and issuer exercise decision structure using current firm value as its state variable. Although in theory these models are very rich and may account for several firm-related events (such as credit risk defaults, mergers, stock dilution, etc), in practice they are very difficult to implement. This is mainly due to the fact that firm-value is a variable not directly observable in the market, hard to estimate and model.

Stock price-based models, on the other hand, model exercise decision structure using the stock price as its state variable. Here stock price may be thought as an observable proxy for the firm value, since it is generally assumed that it reflects all information available in the

market and that any changes to firm value are immediately reflected in it[1]. Therefore, firm-related events should be, indirectly, reflected in stock price value and these models should theoretically encompass all cases firm-value based models excel in. In practice, many jump-like firm-related events, specially credit-risk default events, are not well captured by classical stock-based models, such as those in spirit of Black-Scholes. To cope with such events, these models are often augmented with jump processes, making pricing task much more complex and rendering closed-form formulas practically non-existent.

Historically, one of the first works to deal with Convertible Bond pricing was [IJ77]. Its author Ingersoll proposed a firm value-based model, using classical contingent claim's approach and arbitrage arguments to derive a closed-form formula for some special cases. Later, Lewis ([Lew91]) and Bühler-Koziol ([BK02]) extended this approach for more complex cases and firm's capital structures. However, presented formulas remained constrained to very special cases and ignored important and often found Convertible Bond features, such as early exercise and path-dependent ones.

In [BS77], Brennan-Schwartz presented a firm value-based model and a finite-difference method for pricing Convertible Bonds. Later, in their seminal paper [BS80], through a sensitivity analysis, these authors investigated which factors impacted the Convertible Bond's price the most. Contrary to one would naturally expect, they concluded that the stochasticity of interest rate and stock volatility do not heavily impact the final price and may as well be neglected without introducing large errors on price estimation. Rather, stochasticity of stock price and credit risk were the factors which influenced the most.

Stock price-based models were first introduced by McConnel-Schwartz in [MS86], where authors used a simplified model with a finite difference method for pricing the Liquid Yield Option Note (LYON) security, a then-innovative zero-coupon callable puttable Convertible Bond.

Later works focused on using numerical methods (lattice- and Monte Carlo-based) for coping with early-exercise features and including credit risk in used models. Noteworthy works along these lines include those published by Goldmann Sachs ([Sac94]), Ho-Pfeffer ([HP96]), Tsiveriotis-Fernandes ([TF98]), Ayache et al ([AFV03]), Takahashi et al ([TKN01]), Ammann et al ([AKW03]) and Milanov-Kounchev ([MK12]).

For an excelent, more in-depth and thorough review on modelling and pricing approaches found in literature, refer to [Zad10].

This work may be classified as a Black-Scholes stock price-based model work using Monte Carlo methods as its pricing method. It is mostly inspired by the work of Ammann et al in [AKW08]. Unlike that work, however, we do not use a two-stage simulation with parametric representation of early exercise decisions in the spirit of [Gar03]. Instead, we use backward induction methods such as those proposed by Longstaff-Schwartz ([LS01]) and Potters et al ([PBS01]). In this sense, it is much more akin to [WK05], except in current work bonds are considered to be riskless and interest rate is not stochastic, but rather constant and deterministic. Some improvements on backward induction's regression procedure proposed

[1] This is the **strong form** of the so-called **Efficient Market Hypothesis**. It is a bold simplifying hypothesis, heavily criticised but very commonly assumed in financial models.

by Bouchard-Warin ([WB12]) are also discussed and implemented.

4.2 Setting 1: Non-callable Non-puttable European Convertible Bond

In this first setting, we start with a non-callable non-puttable european Convertible Bond (i.e. whose conversion may only take place at maturity). Although this kind of product is too simple and not much realistic, in the sense that it is very rare to be found in the market, as we shall se later on, it is simple enough to obtain a closed-form solution and draw some preliminary conclusions about Convertible Bonds.

For modelling this kind of product, we choose a stock price-based model (see Section 4.1) - as we believe it to be more intuitive for the layman to grasp - using a plain Black-Scholes stochastic market model (see Section 3.1).

For simplification, we assume only the asset's stock price is stochastic in nature, fixing both interest rate and stock price's volatility as deterministic and constant. This simplification allows us to stick to a simpler implementation (which may be extended to treat the general case) and does not introduce large errors on computation of a Convertible Bond price, as confirmed by the sensitivity analysis provided in the seminal article [BS80].

Although credit risk is stated as a main influencing factor on Convertible Bond price in the sensitivity analysis provided in [BS80], throughout this work, also for implementation simplication, we consider the Convertible Bond to be credit-riskless, pointing out later in Section 4.6 how the provided methods may be extended to cope with this credit-risky.

As in Black-Scholes market model, let $S(t)$ be the asset's stock price, with σ being its constant volatility, and r the deterministic and constant risk-free interest rate. The stock is assumed to pay a continuous dividend yield q on its price. It's also assumed that the market is complete and the only source of uncertainty comes from stochasticity of the asset's stock price. Hence, as seen in Section 3.1, $S(t)$ follows the SDE 3.1.1

$$\begin{cases} dS(t) = & S(t)((r-q)dt + \sigma dW^{\mathbb{Q}}(t)) \\ S(0) = & S_0 \end{cases}$$

It should be noted that, throughout this work, all pricing computation and analysis is done assuming data is provided under risk-neutral measure $\mathbb{Q}$. Although, as stated in Section 3.3.4, LSMC method does not cope well with historical data, HMC method does so ([PBS01], [BP04]). Thus it can be used for pricing with historical data, remaining only calibration issues to be discussed, which will be done in Section 4.6.

Let $CB(t)$ be the Convertible Bond's fair price, $B(t)$ be the value of its corresponding straight bond, $n(t)$ be the conversion ratio at time t. Let κ be the redemption ratio at time $t = T$

(maturity) and N the nominal value.

Suppose initially that the underlying straight bond is a zero-coupon bond (i.e. it does not pay any coupons along its lifetime). As the Convertible Bond in current setting is an european one, only at time $t = T$ the investor may choose to convert it. Thus, its fair price is given at time $t = 0$ by:

$$\begin{aligned} CB^* = CB(0) &= \mathbb{E}^{\mathbb{Q}}(e^{-r(T-0)} \,\mathsf{max}(n(T)S(T), B(T))|\mathcal{F}_0) \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}(\underbrace{\mathsf{max}(n(T)S(T), B(T))}_{\text{(i)}}) \end{aligned}$$

But we may rearrange the term inside (i) as:

$$\begin{aligned} \mathsf{max}(n(T)S(T), B(T)) &= B(T) + \mathsf{max}(n(T)S(T) - B(T), 0) \\ &= B(T) + n(T)\,\mathsf{max}(S(T) - \frac{B(T)}{n(T)}, 0) \end{aligned}$$

Hence, CB^* then becomes:

$$CB^* = e^{-rT}\mathbb{E}^{\mathbb{Q}}(B(T) + n(T)\,\mathsf{max}(S(T) - \frac{B(T)}{n(T)}, 0))$$

As we are considering r to be deterministic and constant, $B(T)$ is also deterministic and hence $\mathcal{F}_t$-mensurable. Thus:

$$CB^* = e^{-rT}B(T) + n(T)\underbrace{e^{-rT}\mathbb{E}^{\mathbb{Q}}(\mathsf{max}(S(T) - \frac{B(T)}{n(T)}, 0))}_{\text{(ii)}}$$

But (ii) may also be regarded as the price of a call option with strike price $K = \dfrac{B(T)}{n(T)}$. Thus, applying Black-Scholes Formula for the call option price (3.1.4), we then arrive at the following Black formula for the Convertible Bond price[2]:

[2] In a setting where the straight bond value is considered to stochastic (e.g. because of a term structure model for the interest rate), one could still use Margrabe Formula, a more general version of Black-Scholes Formula which accounts for options on the exchange of two stochastic quantities. In this case, the two quantities to be exchanged would be $B(T)$ and $n(T)S(T)$. As we are considering deterministic and constant risk-free interest rates throughout this work, we shall not dwelve into Margrabe Formula's use. See [Zad10] and [AKW01] for a brief discusssion on this topic.

$$CB^* = e^{-rT}B(T) + n(T)\left(S_0 e^{-qT}\Phi(d_1) - \frac{B(T)}{N(T)}e^{-rT}\Phi(d_2)\right) \Rightarrow$$
$$\Rightarrow CB^* = e^{-rT}B(T) + n(T)S_0 e^{-qT}\Phi(d_1) - B(T)e^{-rT}\Phi(d_2) \quad (4.2.1)$$

with

$$d_1 = \frac{1}{\sigma\sqrt{T}}\left(log(\frac{N(T)S_0}{B(T)}) + ((r-q) + \frac{\sigma^2}{2})T\right)$$
$$d_2 = d_1 - \sigma\sqrt{T}$$
$$\Phi(x) = \int_{-\infty}^{x} e^{-\frac{1}{2}z^2}dz$$

Equation 4.2.1 gives us a closed-form formula for the Convertible Bond price in current setting, describing it as a sum of a straight bond component and embedded call option component[3].

What remains now is only to define how its corresponding straight bond value is evaluated, as well as the treatment of coupon payments. Throughout this work, the following valuation convention is followed:

1. At time $t = T$, the straight bond's value is given by:

$$B(T) = \kappa N$$

   where κN is the redemption value earned at time $t = T$.

2. At each time where a coupon payment is applicable (even at time $t = T$), its value $C(t)$ is added unconditionally to the value of the Convertible Bond at that point. By unconditionally, it is meant that apart from what exercise action takes place, the investor does receive the coupon available at that date. Or putting it another way, the investor does not need to give up current (and any other previous) coupon in order to do exercise the Convertible Bond.

In current setting, item 2 might not seem to impact much, because conversion, the only exercise action available at this setting, may only take place at time $t = T$, so that each other coupon is always earned and its discounted values may be summed asunder. But in more general settings, such as 4.3 and 4.4, we shall see that this coupon cash-flow influences directly the Convertible Bond price and exercise frontier.

Explicitly incorporating coupon payments at Equation 4.2.1, it then becomes:

[3]Because of this components idea, this model is sometimes called "Component Model" ([Zad10])

$$
\begin{aligned}
CB^* &= \sum_{\tau \in \Omega_{coupon}} e^{-r\tau} C(\tau) + e^{-rT} B(T) + n(T) S_0 e^{-qT} \Phi(d_1) - B(T) e^{-rT} \Phi(d_2) \Rightarrow \\
\Rightarrow CB^* &= \sum_{\tau \in \Omega_{coupon}} e^{-r\tau} C(\tau) + e^{-rT} \kappa N + n(T) S_0 e^{-qT} \Phi(d_1) - e^{-rT} \kappa N \Phi(d_2) \Rightarrow \\
\Rightarrow CB^* &= \sum_{\tau \in \Omega_{coupon}} e^{-r\tau} C(\tau) + n(T) S_0 e^{-qT} \Phi(d_1) + e^{-rT} \kappa N (1 - \Phi(d_2)) \qquad (4.2.2)
\end{aligned}
$$

where Ω_{coupon} is the set of coupon payment dates.

From Equation 4.2.2 we can draw some interesting conclusions on the asymptotic behavior of the Convertible Bond price CB^* with respect to the initial stock price S_0:

If $S_0 \to \infty$, then we have:

$$
\begin{aligned}
& S_0 \to \infty \Rightarrow log(\frac{N(T) S_0}{B(T)}) \to \infty \Rightarrow \\
& \begin{cases} d_1 & \to \infty \\ d_2 & \to \infty \end{cases} \Rightarrow \begin{cases} \Phi(d_1) & \approx 1 \\ \Phi(d_2) & \approx 1 \end{cases} \Rightarrow \\
& \Rightarrow CB^* \approx \sum_{\tau \in \Omega_{coupon}} e^{-r\tau} C(\tau) + n(T) S_0 e^{-qT}
\end{aligned}
$$

This means that, if initial stock price is sufficiently high, than the investor will almost surely convert the Convertible Bond and thus its value becomes just the parity value (possibly adjusted by continuous dividend yield) plus coupon values.

If $S_0 \to 0^+$, then we have:

$$
\begin{aligned}
& S_0 \to 0^+ \Rightarrow log(\frac{N(T) S_0}{B(T)}) \to -\infty \Rightarrow \\
& \begin{cases} d_1 & \to -\infty \\ d_2 & \to -\infty \end{cases} \Rightarrow \begin{cases} \Phi(d_1) & \approx 0 \\ \Phi(d_2) & \approx 0 \end{cases} \Rightarrow \\
& \Rightarrow CB^* \approx \sum_{\tau \in \Omega_{coupon}} e^{-r\tau} C(\tau) + e^{-rT} \kappa N
\end{aligned}
$$

This means that, if initial stock price is sufficiently low, than the investor will almost surely not convert the Convertible Bond and thus its value becomes just the discounted redemption value (possibly adjusted by continuous dividend yield) plus coupon values, i.e., what one would expect to receive from the corresponding straight bond.

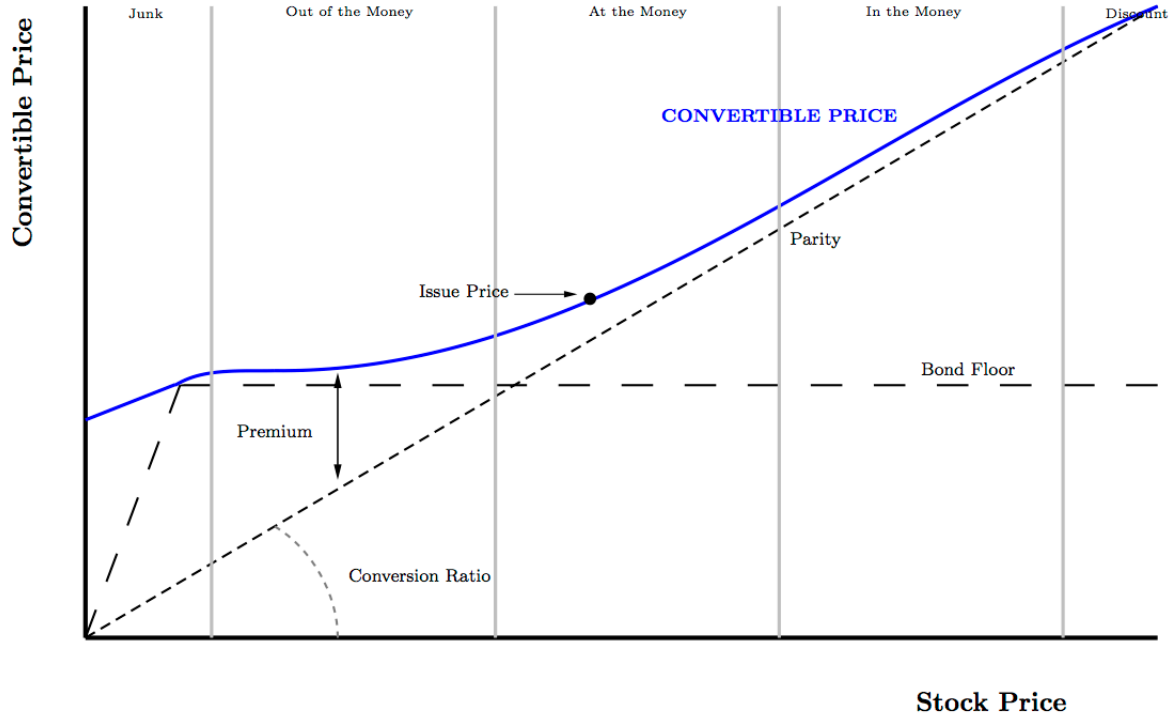

Figure 4.1: Non-callable non-puttable european Convertible Bond price with respect to stock price. Source: [Zad10]

These asymptotic values reflect what we would intuitively expect from investor exercise behavior in these situations.

Plotting Convertible Bond Price along the entire stock price spectrum in current setting leads us ([Zad10]) to Figure 4.1[4]:

Based on which region stock price falls, market classifies the Convertible Bond as **Junk**, **Out of the Money**, **At the Money**, **In the Money** and **Discount Convertible Bond** (see [Zad10] for more details).

Next, we move on to numerical methods. Although it would not strictly be necessary, as in this setting we have a closed-form formula for the Convertible Bond price, we implement Tree and Monte Carlo numerical methods in order to provide a foundation for implementation of forthcoming settings. We then cross-validate this initial implementation with the exact values from Black formula in Equation 4.2.2, adjusting the Tree and Monte Carlo parameters to achieve a reasonable precision level. Later on, whenwe include american and complex exercises and when no closed-form formula is available, we shall cross-validate Monte Carlo results with Tree ones, as Tree methods are largely popular and widely used in market industry for Convertible Bonds and hence could be take as a reasonable benchmark.

For the Monte Carlo method, each stock price path is evaluated according to Equation 3.1.1. We compute paths for both Euler-Maruyama and Milstein scheme. At time $t = T$, the Con-

[4] It is interesting to note that firm value-based models also lead to similar graphs when confronting Convertible Bond Price and stock price spectrum (see [BS77]).

vertible Bond's payoff is calculated as its intrinsic value according to Table 4.1 (a simplified version of the one from [AKW08]). Then, the price is calculated as a mean of discounted realized intrinsic values, according to Monte Carlo method (see 3.3.1).

Payoff	Condition	Time Restriction	Action
$n(T)S(T)$	if $n(T)S(T) \geq \kappa N$	if $t = T$	Conversion
κN	if $n(T)S(T) < \kappa N$	if $t = T$	Redemption

Table 4.1: Exercise actions and respective payoffs in Setting 1

For the Tree method, a regular CRR tree is implemented. As with the Monte Carlo method, at time $t = T$ the payoff is evaluated as the intrinsic value according to Table 4.1 and the Convertible Bond price is calculated in a recursive manner defined in the Tree method (see 3.2).

Example 4.2.1. Consider the following Convertible Bond base example (a simplified version of one proposed in [AKW08]):

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 2 years (european-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	- (non-callable)
Put provisions	- (non-puttable)

Table 4.2: Non-callable non-puttable european Convertible Bond

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

In Table 4.3 we summarize results found for this example, as well as references prices, for comparison.

Reference Prices			
Straight Bond	90.48374		
Black Formula	105.6615		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Black Formula)**	**Confidence Interval ($\alpha = 5\%$)**
Tree (100 time steps)	105.6129	0.04593137%	-
Monte Carlo / Euler (1000 paths / 100 time steps)	103.9161	1.651834%	[101.5143, 106.3179]
Monte Carlo / Milstein (1000 paths / 100 time steps)	103.7889	1.772279 %	[101.4055, 106.1722]
Monte Carlo / Euler (5000 paths / 100 time steps)	105.8127	0.1431244%	[104.7224, 106.9030]
Monte Carlo / Milstein (5000 paths / 100 time steps)	105.7839	0.1158365%	[104.6895, 106.8782]
Monte Carlo / Euler (10000 paths / 100 time steps)	105.7155	0.05112382%	[104.9414, 106.4896]
Monte Carlo / Milstein (10000 paths / 100 time steps)	105.7248	0.05990253%	[104.9479, 106.5017]
Monte Carlo / Euler (25000 paths / 100 time steps)	105.7023	0.03860627%	[105.2138, 106.1907]
Monte Carlo / Milstein (25000 paths / 100 time steps)	105.6968	0.03339507%	[105.2058, 106.1877]
Monte Carlo / Euler (50000 paths / 100 time steps)	105.575	0.08184681%	[105.2298, 105.9202]
Monte Carlo / Milstein (50000 paths / 100 time steps)	105.5594	0.09660883%	[105.2128, 105.9060]
Monte Carlo / Euler (100000 paths / 100 time steps)	105.6663	0.004568832%	[105.4214, 105.9112]
Monte Carlo / Milstein (100000 paths / 100 time steps)	105.6385	0.02170367%	[105.3930, 105.8841]

Table 4.3: Results for Example 4.2.1

First thing to notice from Table 4.3 is that the price given by the Black Formula is greater than the one from the equivalent straight bond. This agrees with what we would expect: a Convertible Bond gives more options to the investor than a straight bond. Moreover, its value is, at every time, almost surely greater than the one attained from the straight bond.

Next, analyzing the obtained results, we see that with just 100 time steps the Tree method gives us a pretty good price result, agreeing with Black Formula's value, with absolute relative error of $0.04593137\% < 1\%$. Thus, we shall stick ourselves to just 100 time steps, as it attains a reasonably good precision level without increasing too much the computational burden. For simplicity, we chose to maintain the same number of time steps for both Tree and Monte Carlo methods.

Monte Carlo methods with Euler-Maruyama and Milstein discretizations both also give pretty good price results, agreeing with Black Formula's value. With just 1000 paths, an absolute relative error of order 1% is attained in both Euler/Mistein schemes. Increasing the number of steps to 5000 paths, an error of order $0.1\% < 1\%$ is attained. Increasing the number of steps to 10000 and further, errors of order $0.01\% < 1\%$ are attained, pairing with the ones achieved with Tree method.

From both error and confidence intervals analysis of Example 4.2.1 we are led to believe

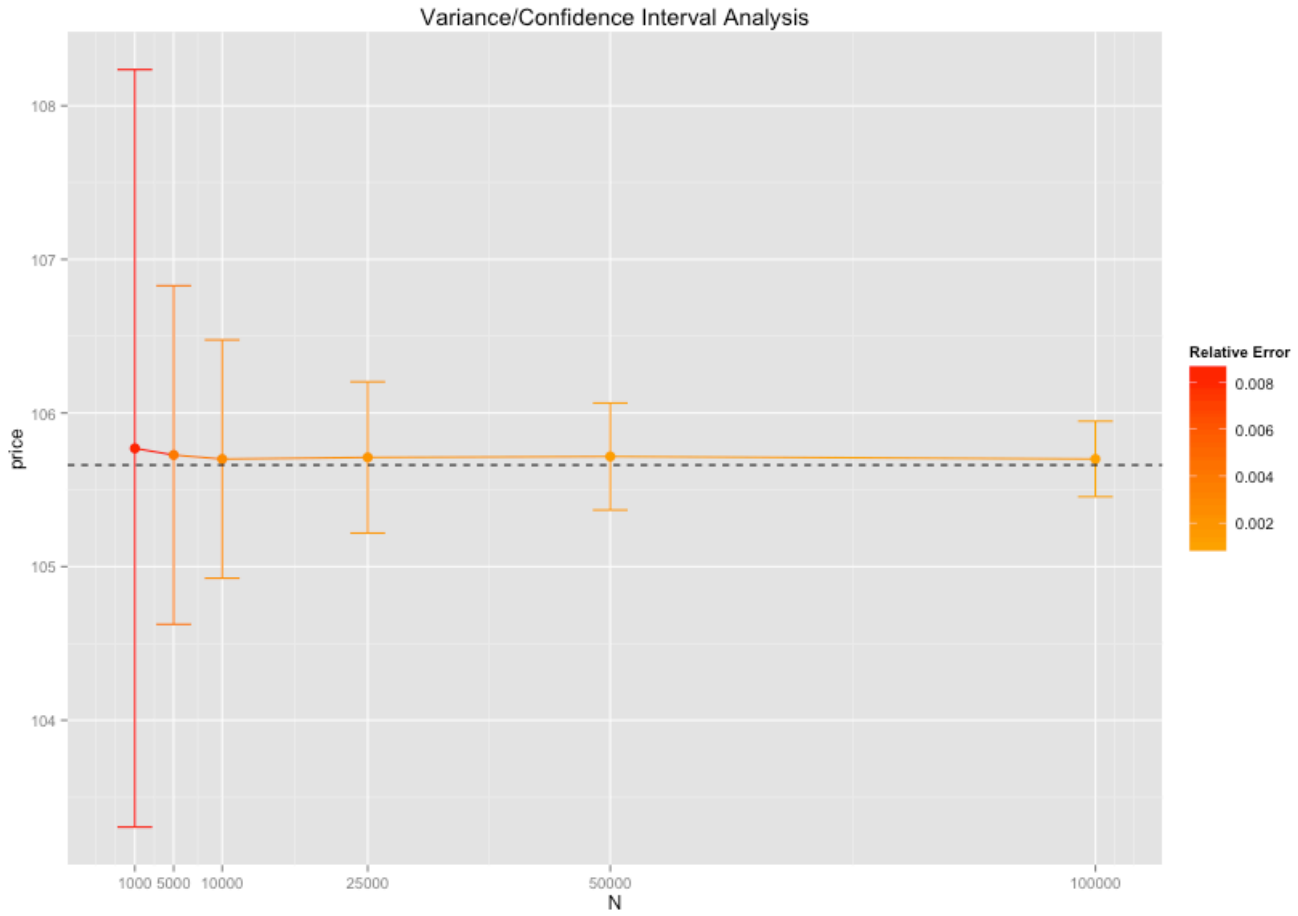

Figure 4.2: Variance/Confidence Interval analysis of the influence of number of Monte Carlo paths on Example 4.2.1 Convertible Bond price.

that we ought to use 10000 paths, as any greater number of paths increases too much the computational burden[5] without given a much better precision level.

To support this conclusions and exempt our analysis from sample bias, we conduct a more broad variance/confidence interval analysis on Example 4.2.1: for each number of sample paths to be analyzed (1000, 5000, 10000, 25000, 50000, 100000 paths), we execute 500 Monte Carlo simulations for Convertible Bond in Example 4.2.1 and average its resulting price, absolute relative error and confidence interval. For simplicity, we use only Euler discretization scheme, as empirical observations show that Milstein values do not diverge much from Euler's. The result is plotted in Figure 4.2. The range bars mark confidence intervals attained for each number of paths. Color gradient indicates relative error level. The Black Formula's reference price is denoted by the dashed line.

We see that, as we originally thought, 10000 paths gives us a good relative error level ($\sim 0.2\%$) and confidence interval with reasonably small computational burden, whilst greater numbers

[5] Although a really not much noticeable problem in current setting, in forthcoming settings, where backward induction techniques are used, greater numbers of paths drastically increase the processing time, from seconds to minutes or even hours in non-optimized codes. For real world usage scenarios, it is thus desireable to minimize the number of paths whenever possible.

of paths do not improve much on these but do overly increase the computational burden. So, from now on, we shall stick to 10000 paths in all our Monte Carlo simulations.

Finally, through empirical analysis, we take a look to what happens to the Convertible Bond price if we change its indenture, by increasing the redemption ratio, the conversion ratio or adding coupon payments.

Example 4.2.2. Consider the following Convertible Bond, modified from Example 4.2.1:

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	110%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 2 years (european-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	- (non-callable)
Put provisions	- (non-puttable)

Table 4.4: Example 4.2.1 Convertible Bond with increased redemption ratio

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

We get the results summarized in Table 4.5.

Reference Prices			
Straight Bond	99.53212		
Black Formula	112.0584		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Black Formula)**	**Confidence Interval ($\alpha = 5\%$)**
Tree	112.0598	0.00128591%	-
Monte Carlo / Euler	112.3238	0.2367987%	[111.5997, 113.0478]
Monte Carlo / Milstein	112.3062	0.2211207%	[111.5799, 113.0325]

Table 4.5: Results for Example 4.2.2. All methods used 100 time steps. For Monte Carlo simulations, 10000 paths were used.

Comparing obtained price with the one from Example 4.2.1, we observe a greater value ($112.0584 > 105.6615$). This agrees with what we would expect from Equation 4.2.2, i.e. the Convertible Bond price rises with increasing redemption ratio.

Example 4.2.3. Consider the following Convertible Bond, modified from Example 4.2.1:

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 2 years (european-styled) **Conversion restrictions**: - **Conversion ratio**: 1.5 **Conversion ratio adjustment clauses**: -
Call provisions	- (non-callable)
Put provisions	- (non-puttable)

Table 4.6: Example 4.2.1 Convertible Bond with increased conversion ratio

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

We get the results summarized in Table 4.7.

Reference Prices			
Straight Bond	90.48374		
Black Formula	133.6573		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Black Formula)**	**Confidence Interval ($\alpha = 5\%$)**
Tree	133.6891	0.02378404%	-
Monte Carlo / Euler	133.1709	0.3639102%	[131.7358, 134.6061]
Monte Carlo / Milstein	133.079	0.4326661%	[131.6423, 134.5158]

Table 4.7: Results for Example 4.2.3. All methods used 100 time steps. For Monte Carlo simulations, 10000 paths were used.

Comparing obtained price with the one from Example 4.2.1, we observe a greater value

($133.6573 > 105.6615$). This agrees with what we would expect from Equation 4.2.2, i.e. the Convertible Bond price rises with increasing conversion ratio.

Example 4.2.4. Consider the following Convertible Bond, modified from Example 4.2.1:

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	0.5 years
Coupon frequency	0.5 years
Coupon ratio	5%
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 2 years (european-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	- (non-callable)
Put provisions	- (non-puttable)

Table 4.8: Example 4.2.1 Convertible Bond with coupon payments

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

We get the results summarized in Table 4.9.

Reference Prices			
Straight Bond	109.2793		
Black Formula	124.4571		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Black Formula)**	**Confidence Interval ($\alpha = 5\%$)**
Tree	124.4085	0.03899478%	-
Monte Carlo / Euler	124.3811	0.06107139%	[123.6005, 125.1616]
Monte Carlo / Milstein	124.3776	0.06387747%	[123.5934, 125.1617]

Table 4.9: Results for Example 4.2.4. All methods used 100 time steps. For Monte Carlo simulations, 10000 paths were used.

Comparing obtained price with the one from Example 4.2.1, we observe a greater value ($124.4571 > 105.6615$). This agrees with what we would expect from Equation 4.2.2, i.e. the Convertible Bond price rises with introduction of coupon payments.

Although forthcoming settings diverge substantially from current one, many of its underlying assumptions do not hold and Equation 4.2.2 simply may not be used, in practice most of the above stated empirical conclusions still apply to them.

Summary of Results and Conclusions

- A Black-Scholes-like closed-form formula is presented for the non-callable non-puttable european Convertible Bond;
- For the given closed-form formula, we inspected the price's asymptotic behavior with respect to initial stock price. We have found that obtained asymptotic expressions for the price is in consonance with what one would intuitively expect as well as with well known graphs of price with respect to initial stock price, such as that in Figure 4.1;
- We have found that 100 time steps are sufficiently to give a very accurate estimated price with Tree method, resulting in less than 1% of relative error (with respect to the value given by the closed-form formula. To ease the comparison, we fixed the number of time steps as 100 for both Tree and Monte Carlo methods;
- Through a variance/confidence interval analysis, we have found that for 100 time steps 10000 paths were sufficient to give a pretty accurate estimated price, with the best compromise between smaller relative error, narrower confidence interval and not so great computational cost;
- Finally, starting with a base european Convertible Bond example and walking into some of its variants, we examined what would happen with the Convertible Bond price. Attained results were in consonance with the previously given closed-form formula.

4.3 Setting 2: Non-callable Non-puttable American Convertible Bond

Next, we move on to a more complex but much similar product: a non-callable non-puttable american Convertible Bond (i.e. whose conversion may take place at any time between issuance and maturity). This product is much more commonly found in market than the previous one from Setting 1.

For modelling this product, we follow the same approach used in Setting 1: we choose a stock price-based model, with only the asset's stock price as stochastic, and deterministic and constant interest rate, stock volatility and continuous dividend yield. Market is assumed to be complete and $S(t)$ is such that it follows SDE 3.1.1. For simplification, we also consider the Convertible Bond to be credit-riskless.

Since we now have an american-styled conversion, the closed-form formula 4.2.2 cannot

be used anymore to price this Convertible Bond. In fact, closed-form formulae for pricing instruments with american exercise are very hard to find and not even guaranteed to exist in the general case. Therefore, from this setting on, we shall resort to numerical pricing methods.

In this work, we are focusing in Monte Carlo methods, as we believe them to be superior in terms of flexibility and extensibility, important features to deal with rich and complex set of features a Convertible Bond may present. For benchmarking this methods though, we shall compare its results to the corresponding Tree method implementation.

The Tree method in current setting is implemented much like in Setting 1, with a CRR tree, except that at every node the corresponding intrinsic value is calculated as the maximum of discounted average value of succeeding nodes and the payoff that would be obtained at that time. Payoff is calculated according to Table 4.10 (another simplified version of the one in [AKW08]), where Ω_{conv} is the set of time where conversion is allowed (in current setting, $\Omega_{conv} = [0, T]$, since conversion is american-styled; in more general settings, conversion may follow any discrete structure) and $V'(t) = \mathbb{E}^{\mathbb{Q}}(CB(\tau^*)|\mathcal{F}_t)$ is the **continuation value** (i.e. the value the investor is expected to receive if instrument is not exercised immediately, but rather held for one more time period and later exercised at an optimal time $\tau^* > t$).

Payoff	**Condition**	**Time Restriction**	**Action**
$n(t)S(t)$	if $n(t)S(t) \geq V'(t)$	if $t \in \Omega_{conv}$	Conversion
κN	if $n(t)S(t) < \kappa N$	if $t = T$	Redemption
$V'(t)$	Otherwise		Continuation

Table 4.10: Exercise actions and respective payoffs in Setting 2

In terms of time steps, we follow the same reasoning exposed in Setting 1, i.e. $N_t = 100$ time steps, for both Tree and Monte Carlo methods. To solve *lack of resolution* issues in exercise boundaries evaluation (a topic we shall discuss further on), we expand the number of steps in Tree method to $N_t = 1000$ time steps, whilst constraining the exercise time steps to the same original 100 time steps, to keep Monte Carlo results comparable.

To cope with american exercises, we implement Monte Carlo methods with backward induction techniques. Both LSMC (Section 3.3.3) and HMC (Section 3.3.4) methods are implemented. For each one, both Euler-Maruyama and Milstein discretization schemes are used. Using the same rationale from Setting 1, we stick with $N_{MC} = 10000$ Monte Carlo paths. In both these Monte Carlo methods, payoff is also calculated according to Table 4.10.

With respect to which family of basis functions to use in LSMC and HMC methods, implementation was made such as to support (weighted and non-weighted) Laguerre, Legendre, Chebyshev (of 1st, 2nd, 3nd type), (physicists') Hermite, Gegenbauer and Jacobi polynomials, with dimensions $M \geq 2$. Interestingly, our empirical results indicate that, given a fixed scenario of Convertible Bond setting and number of Monte Carlo paths and time steps, each of the cited non-weighted polynomial basis produce exactly the same results, being completely equivalent. In contrast, their weighted counterparts seem to produce very poor

results. Hence, to standardize our simulation procedures and make results more comparable, we shall along this work stick only to (physicists') Hermite polynomials, giving then some insight to what happens with results when we vary the dimension of the basis.

To alleviate numerical errors in regression procedure in both LSMC and HMC methods, we also make use of the following modifications:

Use QR decomposition to solve least-squares problem in regression procedure

As seen in Sections 3.3.3 and 3.3.4, the core of both LSMC and HMC lies in regression procedure, which itself relies in an optimization problem that can be reduced to a least-squares problem of the form:

$$\alpha = \underset{\alpha \in \mathbb{R}^n}{\text{argmin}} \quad \|A\alpha - B\|^2$$

Since more efficient optimization techniques are available to solve quadratic optimization problems, there is no need to use more general convex optimization techniques. In [WB12], authors digress about pros and cons of some of the former techniques: linear system solving using Cholesky decomposition, QR decomposition and SVD decomposition. Since the QR decomposition technique provides a good balance between roundoff errors resilience, memory requirements and computational efficiency, we shall stick to it to implement regression in LSMC and HMC methods along this work.

Modify stock price path's generation procedure to use an array of initial stock price values instead of just one single value

Because in Monte Carlo methods all stock price paths are usually generated from a single starting point $S = S_0$, as the backward induction algorithm proceeds in its backward manner, farther from maturity $t = T$, near initial time $t = 0$, paths begin to collapse and little information becomes available, leading to response terms almost equal across all paths. Not only rendering poor regression results, this often leads to ill-conditioned or singular linear systems near initial time $t = 0$, causing the QR decompostion (and hence all regression procedure) to fail.

To tackle this problem, the idea is to generate stock price paths using an array of initial stock price values instead of just one single value. Let S_0^* be the original stock price initial value. We artificially create an array of stock price initial values in the neighborhood of S_0^*. Let S_0 now denote the set of stock price initial values. For each value in S_0, we sample N_{MC} Monte Carlo paths with N_t time steps, forming an array of multi-originated paths. Then we procede with usual LSMC/HMC algorithm, evaluating the regression at each time step with the full array, not only those paths originating in S_0^*. Finally, the final price is evaluated as the mean of found intrinsic values at time $t = 0$, but now conditioned to sigma-algebra generated by

$\{S_0 = S_0^*\}$ (i.e. the mean of found intrinsic values at time $t = 0$ for paths that started at $S_0 = S_0^*$).

For providing a balanced, symmetric array of stock price initial values, we choose parameters in a such way that the number of S_0 values that are greater than S_0^* (the upper band of S_0) is equal to the number of values that are smaller than S_0^* (the lower band of S_0). Let Δ_{S_0} be the distance value from S_0^* to both maximum value in the upper band and minimum value in the lower band. Let the number of values in upper band and lower band both be N_{S_0}. Thus, the whole array includes $2N_{S_0} + 1$ values. Then, we have:

$$
\begin{aligned}
S_{0_{max}} &= S_0^* + \Delta_{S_0} \\
S_{0_{min}} &= S_0^* - \Delta_{S_0} \\
\delta_{S_0} &= \frac{S_{0_{max}} - S_{0_{min}}}{2N_{S_0}} = \frac{(S_0^* + \Delta_{S_0}) - (S_0^* - \Delta_{S_0})}{2N_{S_0}} \Rightarrow \\
\delta_{S_0} &= \frac{2\Delta_{S_0}}{2N_{S_0}} = \frac{\Delta_{S_0}}{N_{S_0}}
\end{aligned}
$$

$$
\begin{aligned}
S_0 = \{&S_{0_{min}}, S_{0_{min}} + \delta_{S_0}, S_{0_{min}} + 2\delta_{S_0}, \cdots, \\
&S_0^* - \delta_{S_0}, S_0^*, S_0^* + \delta_{S_0}, \cdots, S_{0_{max}} - 2\delta_{S_0}, S_{0_{max}} - \delta_{S_0}, S_{0_{max}}\} \Rightarrow \\
\Rightarrow S_0 = \{&S_0^* - N_{S_0}\delta_{S_0}, S_0^* - (N_{S_0} - 1)\delta_{S_0}, \cdots, \\
&S_0^* - \delta_{S_0}, S_0^*, S_0^* + \delta_{S_0}, \cdots, S_0^* + (N_{S_0} - 1)\delta_{S_0}, S_0^* + N_{S_0}\delta_{S_0}\}
\end{aligned}
$$

In our empirical tests, we have found a good guess for $\Delta_{S_0} = 0.8S_0^*$.

Since the number of paths from other values in S_0 contribute to rise of regressions' computational cost but not to the accuracy of the Monte Carlo estimator (only those paths starting at $S_0 = S_0^*$ will be taken into account to form the Monte Carlo estimator), it is useful to break the initial simmetry and allow the number of Monte Carlo paths starting at S_0^* to be greater than those starting at other values in S_0. Let $N_{MC_{band}}$ denote the number of paths attributed to each initial value in upper and lower bands of S_0 and N_{MC} now denote the number of paths attributed to initial value S_0^*. The effective number of Monte Carlo paths now increase from N_{MC} to $N_{MC} + 2N_{S_0}N_{MC_{band}}$.

We have found, in our empirical tests for this setting, that parameters $N_{S_0} = 100$, $N_{MC} = 10000$, $N_{MC_{band}} = 10$ provide a good balance between computational cost, good regression results and precision level of Monte Carlo estimators.

Next, before stepping into an example of Convertible Bond in this setting, we shall investigate another natural question: what is the relationship between the price of the american Convertible Bond and its european counterpart?

A formal proof is out of the scope of this work, but we try to give here some intuitive reasoning to what happens. Let $CB_E(t)$ and $CB_A(t)$ be the price of the non-callable non-puttable european Convertible Bond and american Convertible Bond, respectively. Then, as the american one gives the investor the extra possibility to exercise it at earlier times and at each time its intrinsic value is the maximum of the corresponding european value and a, possibly greater, payoff, one would naturally expect that:

$$CB_A(t) \geq CB_E(t) \tag{4.3.1}$$

Therefore, Inequality 4.3.1 provides a lower bound estimator for the american Convertible Bond price and may be used as a simple sanity check for pricing method implementations in current setting. We shall see later on, that attained results in fact respect that inequality.

Example 4.3.1. Consider the following Convertible Bond example (another simplified version of one proposed in [AKW08]), listed in Table 4.11.

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 0-2 years (american-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	- (non-callable)
Put provisions	- (non-puttable)

Table 4.11: Non-callable non-puttable american Convertible Bond

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

For this example, we get the results summarized in Table 4.12. Tree method used 1000 time steps, with exercise time steps constrained to the same 100 time steps used by Monte Carlo methods. For Monte Carlo simulations, 201 values in stock price mesh were used, with 10000 paths for the original (central) stock price initial value and 10 for each of the others.

For both LSMC and HMC methods, a Hermite polynomial basis with dimension $M = 4$ is used.

Reference Prices			
Straight Bond	90.48374		
Black Formula (European Convertible Bond)	105.6615		
Tree (1000 time steps)	109.1298		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Tree)**	**Confidence Interval ($\alpha = 5\%$)**
LSMC / Euler / Hermite M=5	111.3244	2.011036%	[110.5088, 112.1400]
LSMC / Milstein / Hermite M=5	111.2122	1.908247%	[110.3960, 112.0284]
HMC / Euler / Hermite M=5	111.0183	1.730531%	[110.2009, 111.8356]
HMC / Milstein / Hermite M=5	110.908	1.629442%	[110.0970, 111.7189]

Table 4.12: Results for Example 4.3.1.

First thing to notice from results in Table 4.12 are in consonance with Inequality 4.3.1, indicating attained results to be quite reasonable.

Also, from results in Table 4.12 we see that HMC algorithm yielded slightly better results than LSMC, with smaller errors and also narrower confidence interval. This indicates that the hedge functions used in HMC, approximated as the derivative of the price basis functions (see 3.3.4), are correctly capturing the optimal hedging strategy for this Convertible Bond, yielding finally a better price than LSMC did. Simulations using Milstein discretization also gave slightly better results than their counterpart using Euler-Maruyama discretization, as expected.

Interestingly, we also notice from Table 4.12 that, although attained relative errors for LSMC and HMC methods are reasonably low (around), they are quite large when compared to those attained for regular Monte Carlo method in Setting 1 (around). Empirically investigating which parameters drive this trend, it was found that this is indeed a numerical issue and the high stock volatility (σ) used in Example 4.3.1 was its cause. Although in theory high volatility should not have a big impact on backward induction methods such as LSMC and HMC, high volatility scenarios cause big fluctuations on stock price from a time step to another, generating wildly-behaved (i.e. that change too quickly) stock price paths, leading to poorer regression results compared to those attained with more mildly-behaved stock price paths originated in low volatility scenarios. Since accuracy of backward induction methods relies critically on good estimation of continuation values to correctly capture exercise decision structure and these continuation values are themselves inferred from the regression procedure, poorer regression results at each time step leads to estimation errors that are cascaded as the algorithm walks backwardly, finally leading to a greater relative error with respect to Tree method than that attained regular Monte Carlo. Thus, it is observed that relative error in LSMC and HMC methods rises as volatility increases. These findings are in consonance to those made in vanilla american options settings ([Jia09]), indicating this to be not an issue related to a Convertible Bond setting, but rather a drawback of LSMC and HMC methods themselves.

Unfortunately, since in real world scenarios stock volatility is not a controlled parameter of the model but rather calibrated from market data, this *volatility effect* poses a serious issue for general use of LSMC and HMC methods to price Convertible Bonds. To alleviate this problem, one needs to improve regression. To try to achieve this goal, we then make use of the following technique proposed by Bouchard and Warin in article [WB12]: instead of, at each time step, doing just one regression encompassing all paths, stock price data is sorted and partitioned into N_{reg} intervals, with a regression executed separately for each of those stock price intervals. Using $N_{reg} = 1$ in Bouchard's regression is completely equivalent to use the regular regression procedure.

Although individual empirical results shows Bouchard's regression technique does lower the attained relative errors, relative errors results vary strongly with used number of regression intervals N_{reg} and polynomial basis' dimension M[6], as well as with volatility itself. To better visualize this volatility effect on relative errors and analyze if there is an optimal choice of the pair of parameters N_{reg} and M, the following study is conducted[7]: for each of 50 simulations (each with different random number samples) of $N_{MC} = 10000$ paths and $N_t = 100$ time steps, the evolution of relative error to Tree method with respect to several σ (0.05, 0.1, 0.2, 0.3, 0.4, 0.5) is tracked for each combination for several M (2, 3, 4, 5) and several numbers of regression intervals N_{reg} (1, 2, 5, 10, 50). Finally, results are then arranged horizontally by M and vertically by N_{reg}. This study is conducted for both LSMC and HMC methods, with results respectively summarized in Figures 4.3 and 4.4. Only results for Euler-Maruyama discretization are shown, since we have found Milstein ones to be quite similar.

Analyzing Figures 4.3 and 4.4, we reach the following conclusions:

- The aforementioned volatility effect on relative error is easily confirmed: the relative error consistently rises as volatility increases;

- With respect to the evolution of relative error with regression intervals, a consistent behaviour for both LSMC and HMC methods is displayed: increasing the number of regression intervals a little (e.g. $N_{reg} = 2, 5$) result, on average, in smaller relative errors when compared to regular regression procedure. Nevertheless, increasing too much the number of regression intervals (e.g. $N_{reg} = 10, 50$) result, on average, in relative errors even larger than regular regression procedure;

- With respect to the evolution of relative error with basis' dimension, a consistent behaviour is also displayed: increasing the basis' dimension results, on average, in larger relative errors;

- The optimal choice of parameters is $M = 3$ and $N_{reg} = 2$ for both LSMC and HMC methods. For this choice, in both methods relative errors are constrained, on average, to impressively low levels (less than 1%), similar to those attained with regular Monte

[6] As said previously, our empirical results indicate that all non-weighted polynomial basis yield the same estimated Convertible Bond price, so exactly which family of polynomial basis' functions is employed should be irrelevant here. Nevertheless, as said, throught this work we used Hermite polynomial basis.

[7] The study presented here is highly dependent and restricted to the provided example of Convertible Bond in current Setting. Yet, it was very computational intensive, demanding many hours, days of processing. Because of this, a more broad, generalized study on current Setting is outside the scope of this work.

Carlo, even in scenarios with high volatility ($\sigma = 0.4, 0.5$).

Therefore, when using LSMC and HMC methods, one must always take into account this volatility effect on the attained relative error. In general, use Bouchard's regression technique helps to reduce impact of high volatility on attained prices and errors. However, the best choice of parameters of (polynomial) basis' dimension and number of regression intervals is not obvious and highly depends upon the setting and its decision structure, demanding a study like the one presented here. Since this type study is very computational intensive and demands several hours, days of processing, it may not always be possible to execute it. For current Setting (non-callable non-puttable american Convertible Bond), however, we believe that the combination $M = 3$ and $N_{reg} = 2$ provides a reasonable starting point.

Next, we examine another natural question: what is the shape of the optimal exercise boundary. The **optimal exercise boundary** (or simply **exercise boundary**) is the set of stock price values that separates an exercise region from a non-exercise region. An important detail to notice is that, depending on the exercise action, the exercise region may be lower bounded or upper bounded by the exercise boundary. For the case of voluntary conversion, for greater stock price values the investor will choose to convert the Convertible Bond. Hence, exercise region for conversion must be lower bounded. Using the same analysis, we also conclude that exercise region for redemption must be upper bounded. These conclusions are summarized in Table 4.13.

Action	Type of Boundary Region
Conversion	Lower bounded
Redemption	Upper bounded

Table 4.13: Boundary regions in Setting 2

In Figure 4.5, we present the exercise boundary graph obtained for Tree method initially with 100 steps. Notice how at each level the boundary bounces up and down between successive values. This is due to a *lack of resolution* experienced with Tree method: because of the low number of time steps used, near the real exercise value we have two different tree nodes with a large gap between them, making the attained boundary bounce up and down in the graph. This effect, may be relieved using a technique presented in [AKW08]: whilst restricting the exercises to the same 100 time steps, we then rise the number of time steps in the tree to 1000 time steps. The resulting exercise boundary graph is presented in Figure 4.6. We see that the bounce effect on the boundary vanishes, being replaced by a smoother boundary.

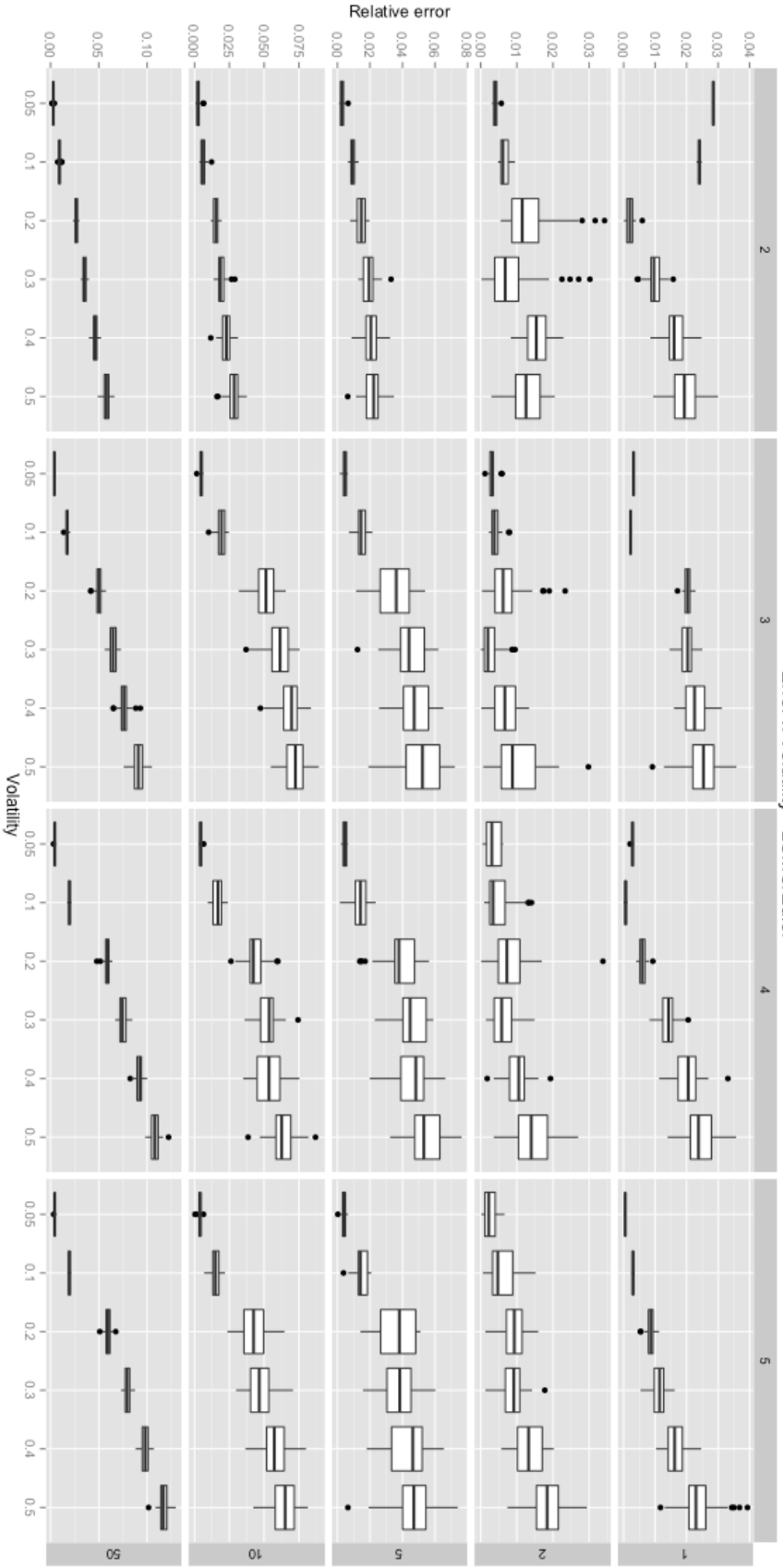

Figure 4.3: Influence of volatility on relative error for LSMC in Setting 2

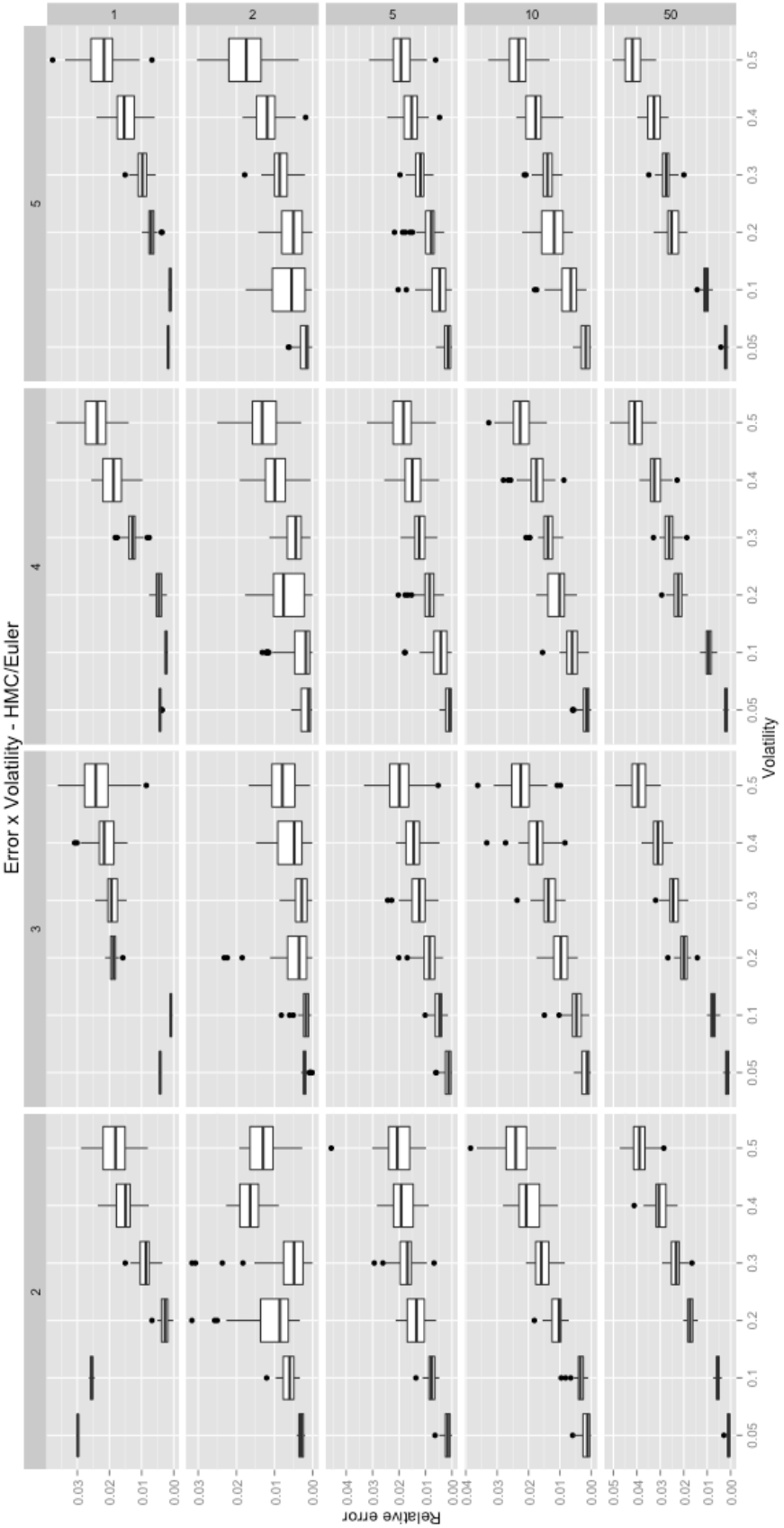

Figure 4.4: Influence of volatility on relative error for HMC in Setting 2

Next, we compare the exercise boundary obtained with Tree method with those obtained with LSMC/HMC algorithms. We have found boundaries to be quite chaotic and varying wildly with the combination of basis' dimension and number of regression intervals used. Figures 4.7 and 4.8 present results obtained for LSMC and HMC algorithms, arranged horizontally by basis' dimension and vertically by number of regression intervals. Only results for Euler-Maruyama discretization are shown, since we have found Milstein ones to be quite similar. Since all (non-weighted) polynomial basis[8] yielded the same exercise boundaries for each combination of basis'dimension and number of regression intervals used, all boundaries were estimated using a (physicists') Hermite basis.

As we may see from Figures 4.8 and 4.8, LSMC and HMC exercise boundary results are very poor in comparison to corresponding Tree results, similar to what was found in other works, such as [BHM13]. The reason behind this problem is pretty much the same of price estimation error: at each time step, due to numerical issues in regression, continuation values are not correctly estimated. Since these values are critical to correctly exercise decision structure, the exercise boundary is poorly captured. Since numerical errors are cascaded backwards because of the backward induction algorithm, as we walk farther from maturity boundary estimation gets even worse.

Unfortunately, we also see from Figures 4.7 and 4.8, that, unlike the price case, no combination of basis' dimension and number of regression intervals in any of the 2 methods, LSMC and HMC, yield a good exercise boundary, reasonably alike to the one given by the Tree method. This indicates another drawback of these backward induction methods: in general, LSMC and HMC methods may not, at least in the form presented and used in this work, be appropriate for finding exercise boundaries of Convertible Bonds. In their place, it is advised to use Tree methods (that as we shall see later on, give reliable results) and PDE methods, formulating the pricing problem analytically as a free boundary problem and possibly solving it numerically (see [BN04], [BNV06])[9].

[8] Since we have found that, as with prices, their weighted versions also produce worse results in terms of boundaries, we only present here boundary results for their non-weighted counterparts.

[9] This latter approach is classically regarded as the best choice for finding exercise boundaries. Unfortunately, it may not always be used with Convertible Bonds, because already cited highly heteregenous and sometimes path dependent features make it very difficult to formulate the corresponding free boundary problem analytically as a PDE.

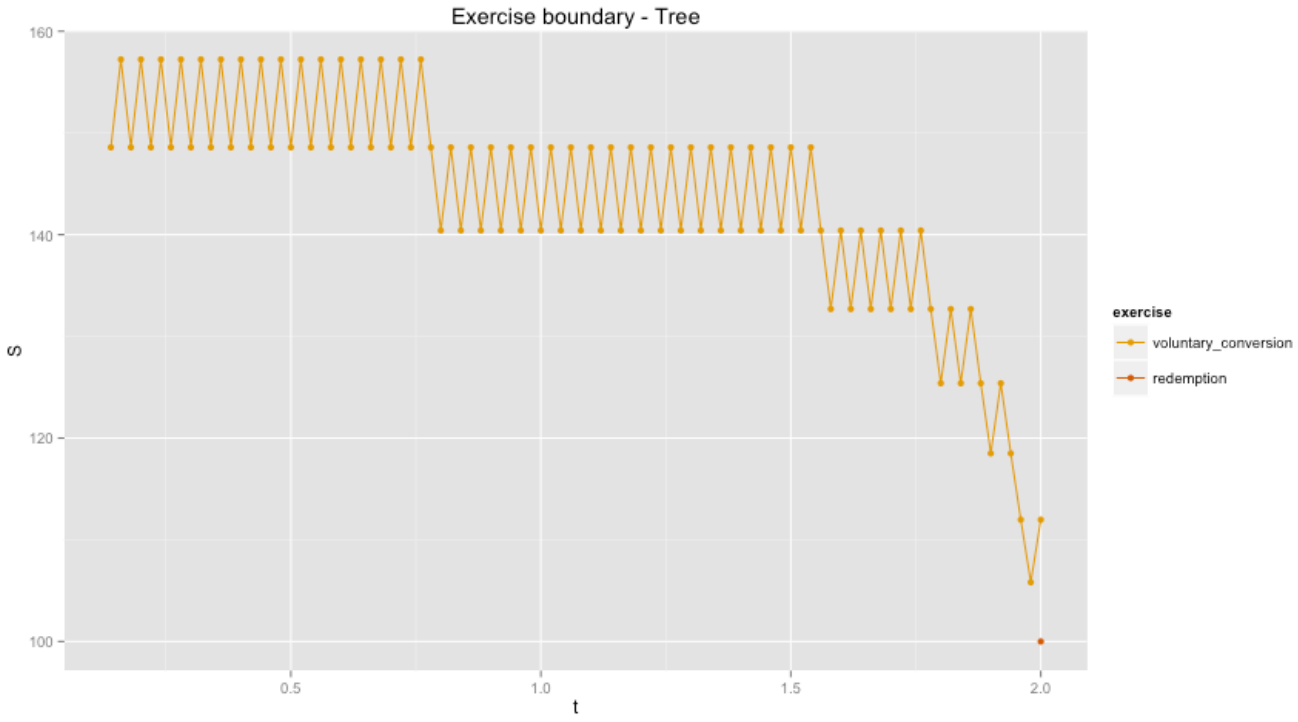

Figure 4.5: Exercise boundary obtained for Example 4.3.1 using Tree method and 100 time steps

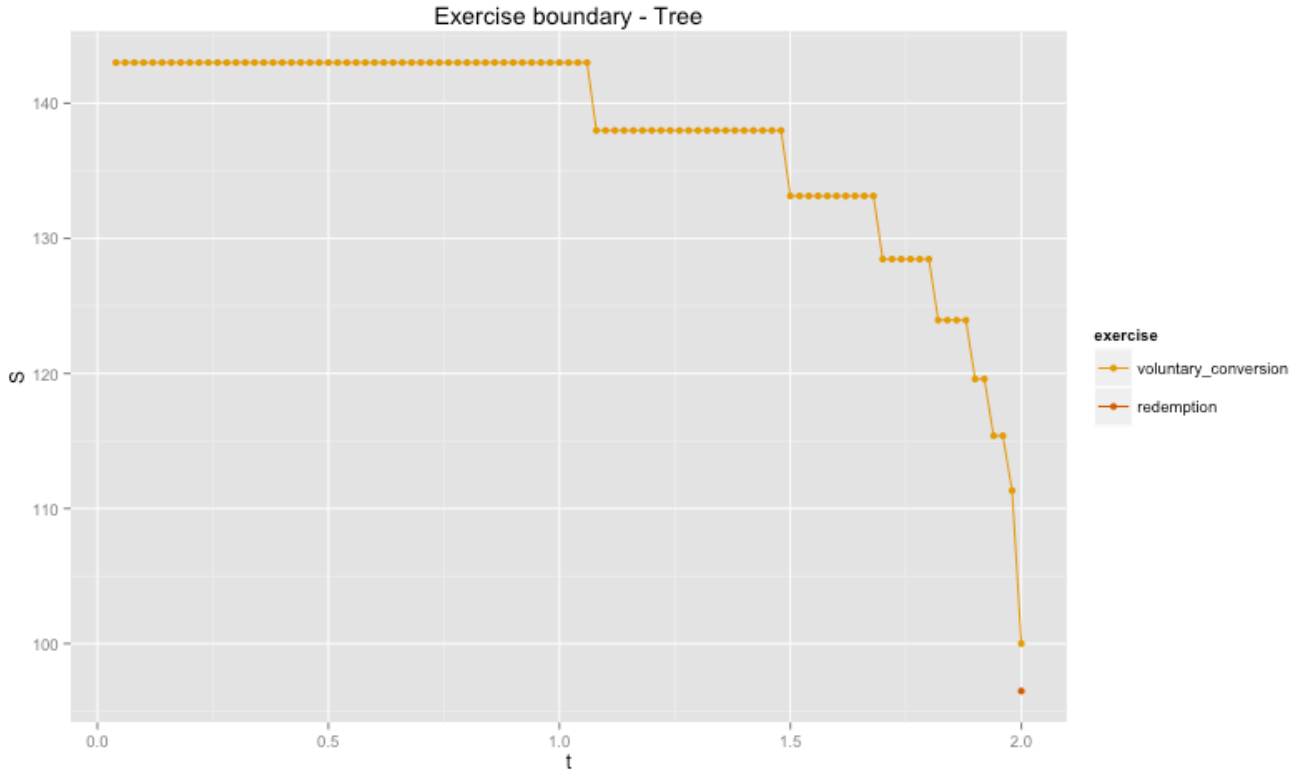

Figure 4.6: Exercise boundary obtained for Example 4.3.1 using Tree method and 1000 time steps

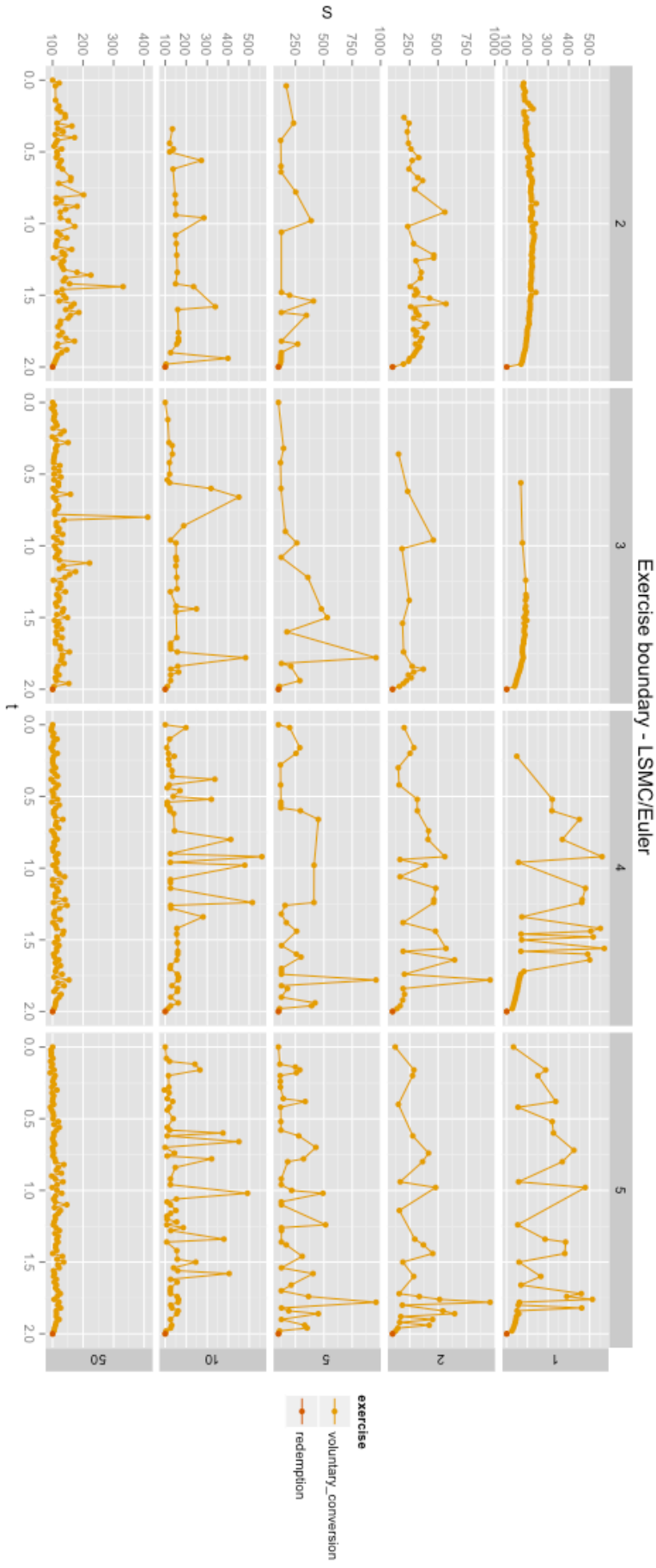

Figure 4.7: Boundaries obtained with LSMC for Example 4.3.1

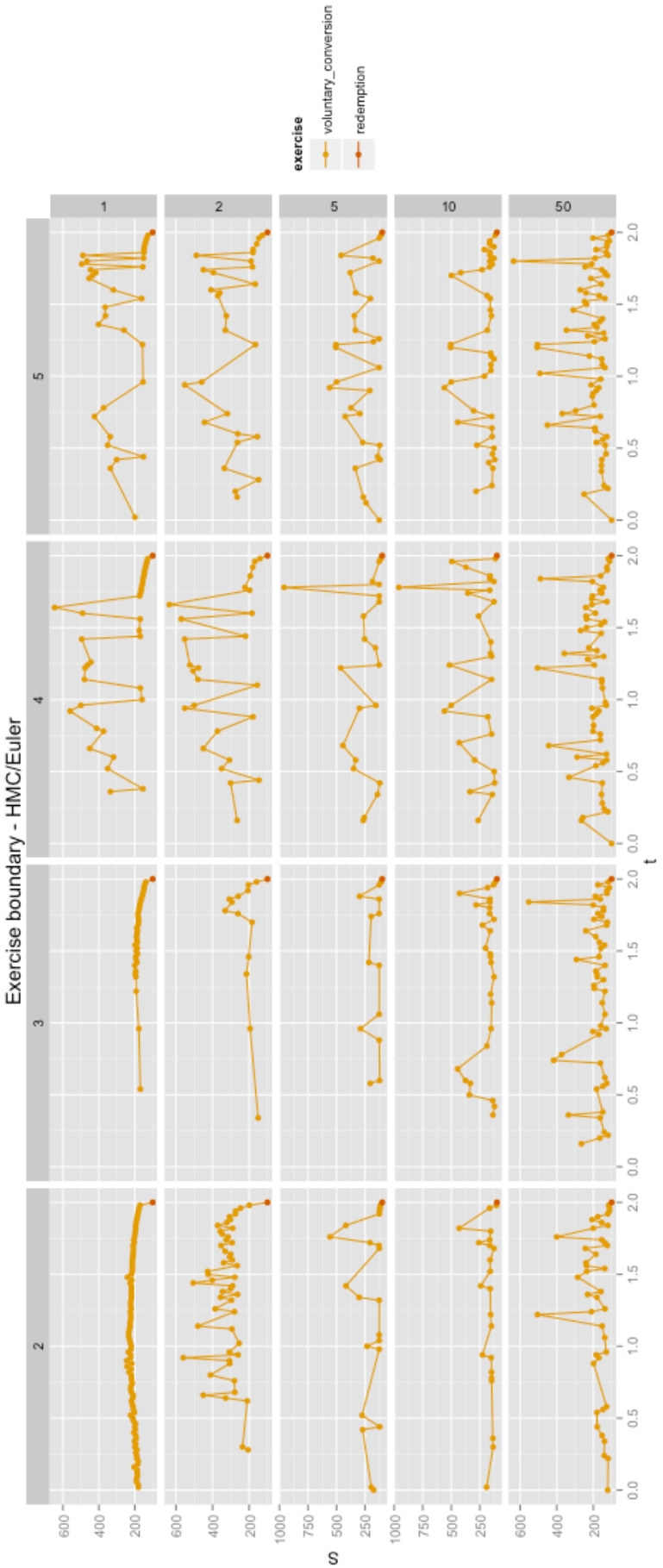

Figure 4.8: Boundaries obtained with HMC for Example 4.3.1

Summary of Results and Conclusions

- We have showed how to price a non-callable non-puttable american Convertible Bond using both LSMC and HMC methods. 2 modifications (use QR decomposition to solve the least-squares problem in regression procedure; use an array of initial stock prices instead of just one single value) were discussed and incorporated in these methods;

- We have stated and given an intuitive reasoning an an inequality relating prices of a non-callable non-puttable american Convertible Bond and its european counterpart (4.3.1). This inequality gives a lower bound for the price of the american Convertible Bond, as well as serving as a sanity check for testing numerical methods;

- Through an example of american Convertible Bond, we have showed that both LSMC and HMC yield fairly good price estimates, with HMC yielding slightly better results, but nothing remarkable. Also, simulations using Milstein discretization yielded slightly better results than those using Euler-Maruyama, but also nothing remarkable;

- Empirically investigating results attained in given example of american Convertible Bond, we have identified that its high volatility was impacting negatively on LSMC and HMC results, with errors around 1%. Through a volatility study on given example, we have sucessfully proven that relative error consistently rises as volatility increases. Based on empirical findings, we have reasoned about the cause of this *volatility effect*, relating it to numerical problems in regression procedure. Since this effect was also observed in contexts of vanilla american options, there is evidence to believe this to be a major drawback of LSMC and HMC methods themselves;

- To alleviate this *volatility effect*, we suggest to employ Bouchard's regression technique to, instead of executing just one single regression, partition stock price data into several intervals and execute a separate regression for each of these intervals. We have found that, for this Setting, increasing a little the number of regression intervals ($N_{reg} = 2, 5$) result, in average, in smaller relative errors than those attained with regular regression procedure. On the other hand, increasing too much the number of regression intervals ($N_{reg} = 10, 50$) result in much larger relative errors;

- For the given example in this Setting, through the executed volatility study we have successfully found the combination basis' dimension $M = 3$ and number of regression intervals $N_{reg} = 2$ to be the optimal choice o parameters in the sense of stabilizing and restricting relative error, in average, at lower levels ($< 1\%$), even in scenarios of high volatility. This is result is highly dependent on the specific example of Convertible Bond, but we believe this combination to be a reasonable starting point for other examples of american Convertible Bond of this Setting;

- Finally, through a exercise boundary study of given example, we have found that LSMC and HMC methods do yield very poor boundaries when compared to those yielded by the Tree method, sporting yet another major drawback of these methods. Based on empirical findings, we have reasoned about the cause of this issue, also relating it to numerical problems in regression procedure.

4.4 Setting 3: Callable Puttable American Convertible Bond

Next, we move on to a even more complex product: a callable and/or puttable american Convertible Bond, a more general class of instruments commonly found in the market. This setting is inherently more complex than the previous because, contrary to what happened in them, there are now 2 actors, the issuer and the investor, with conflicting interests (the investor seeks to maximize its own gains, the issuer seeks exactly the opposite), interacting to determine what payoff is attributed to the instrument.

For modelling this product, we follow the same approach used in both Settings 1 and 2: we choose a stock price-based model, with only the asset's stock price as stochastic, and deterministic and constant interest rate, stock volatility and continuous dividend yield. Market is assumed to be complete and $S(t)$ is such that it follows SDE 3.1.1. For simplification, we also consider the Convertible Bond to be credit-riskless.

Following what was done in Setting 2, we do the pricing using both LSMC and HMC methods, with both Euler-Maruyama and Milstein discretization. All modifications to backward induction methods presented in Setting 2 are also applied. Obtained results are compared with those obtained with Tree method, which as regarded as our benchmark. Tree method is implemented in the same manner as in setting 1 and 2, using a CRR tree.

The payoff at each time is calculated according to Table 4.14 (the same as listed in [AKW08]), where Ω_{conv} is the set of time where conversion is allowed (in current setting, $\Omega_{conv} = [0, T]$, since conversion is american-styled; in more general settings, conversion may follow any discrete structure), Ω_{call} and Ω_{put} are respectively the set of time where call and put exercises are allowed (both may follow any discrete structure), $C(t)$ and $P(t)$ are respectively the agreed strike prices for call and put exercise (normally constant, but could also be time-dependent)[10] and $V'(t) = \mathbb{E}^{\mathbb{Q}}(CB(\tau^*)|\mathcal{F}_t)$ is the continuation value. Since we now have two types of conversion (voluntary and forced), we explicitly distinguish them in the payoff table.

A natural question that arises, similar to that examined from Setting 2, is: what is the relationship between the price of the non-callable non-puttable american Convertible Bond and its pure-callable (i.e. callable but non-puttable) and pure-puttable (puttable but non-callable) counterparts?

Again, a formal proof is out of the scope of this work, and we try to give here only some intuitive reasoning to what happens. Let $CB_A(t)$, $CB_A^c(t)$, $CB_A^p(t)$ be the price of the non-callable non-puttable american Convertible Bond and its pure-callable and pure-puttable counterparts, respectively. As a call feature gives the issuer the possibility of ending the contract before maturity, effectively removing possibility of later conversion or redemption by the in-

[10] As is usual in stochastic models in Finance, we also require all stochastic processes $\{n(t)\}_{t\in[0,T]}$, $\{C(t)\}_{t\in[0,T]}$, $\{P(t)\}_{t\in[0,T]}$ to be adapted processes w.r.t the filter $\{\mathcal{F}_t\}_{t\in[0,T]} = \{\sigma(W^{\mathbb{Q}}(t)\}_{t\in[0,T]}$. In other words, we require all Convertible Bond's indenture terms to depend only on past stock information, not on future ones.

Payoff	Condition	Time Restriction	Action
$n(t)S(t)$	if $n(t)S(t) > V'(t)$ and $P(t) \leq n(t)S(t)$	if $t \in \Omega_{conv}$ if $t \in \Omega_{put} \cap \Omega_{conv}$	Voluntary Conversion
$P(t)$	if $P(t) > V'(t)$ and $n(t)S(t) < P(t)$	if $t \in \Omega_{put}$ if $t \in \Omega_{put} \cap \Omega_{conv}$	Put
$C(t)$	if $V'(t) > C(t)$ and $C(t) \geq n(t)S(t)$	if $t \in \Omega_{call}$ if $t \in \Omega_{call} \cap \Omega_{conv}$	Call
$n(t)S(t)$	if $V'(t) > C(t)$ and $C(t) < n(t)S(t)$	if $t \in \Omega_{call} \cap \Omega_{conv}$	Forced Conversion
κN	if $n(t)S(t) < \kappa N$	if $t = T$	Redemption
$V'(t)$	Otherwise		Continuation

Table 4.14: Exercise actions and respective payoffs in Setting 3

vestor, one would naturally expect that:

$$CB_A^c(t) \leq CB_A(t) \tag{4.4.1}$$

On the other hand, as a put feature gives the investor the possibility of ending the contract before maturity, possibly yielding a greater value than that achieved with conversion or holding the bond until redemption, one would also naturally expect that:

$$CB_A^p(t) \geq CB_A(t) \tag{4.4.2}$$

Combining Inequalities 4.4.1 and 4.4.2 yields

$$CB_A^c(t) \leq CB_A(t) \leq CB_A^p(t) \tag{4.4.3}$$

Therefore, Inequality 4.4.3 may be used as a simple sanity check for pricing method implementations in current setting. We shall see later on, that attained results in fact respect that inequality.

With respect to the callable and puttable american Convertible Bond price, besides Inequality 4.4.3 assuring that its price is bounded by its non-callable and non-puttable counterparts' prices, not much can be really said: its exact price depends on what are the strike prices of call and put features, whether these features are in fact exercised[11] and the interaction between investor and issuer, resulting in a feature being dominant over the other, pulling price upwards or downwards with respect to the price of its non-callable and non-puttable counterpart.

[11] If call strike price is too high, this feature will simply not be exercised. Conversely, if put strike price is too low, it will also not be exercised. These effects can be empirically observed both in obtained price and exercise boundaries.

Consider the following Convertible Bond examples:

Example 4.4.1. Pure-puttable american Convertible Bond example (a pure-puttable version from the one proposed in [AKW08]), listed in Table 4.15.

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 0-2 years (american-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	- (non-callable)
Put provisions	**Strike price**: 98 **Put dates**: 0-2 years (american styled)

Table 4.15: Pure-puttable american Convertible Bond

Example 4.4.2. Pure-callable american Convertible Bond example (a pure-callable version from the one proposed in [AKW08]), listed in Table 4.16.

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 0-2 years (american-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	**Strike price**: 110 **Call dates**: 0-2 years (american-styled) **Hard call clause**: - **Soft call clause**: - **Call notice period**: -
Put provisions	- (non-puttable)

Table 4.16: Pure-callable american Convertible Bond

Example 4.4.3. Callable puttable american Convertible Bond example (the same proposed in [AKW08]), listed in Table 4.17.

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 0-2 years (american-styled) **Conversion restrictions**: - **Conversion ratio**: 1 **Conversion ratio adjustment clauses**: -
Call provisions	**Strike price**: 110 **Call dates**: 0-2 years (american-styled) **Hard call clause**: - **Soft call clause**: - **Call notice period**: -
Put provisions	**Strike price**: 98 **Put dates**: 0-2 years (american styled)

Table 4.17: Callable puttable american Convertible Bond

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

As with Setting 2, empirical results indicate that the aforementioned *volatility effect* also affects the price of Convertible Bond in this Setting: relative error consistently rises as volatility increases. We notice that the pure-puttable case (Example 4.4.1) behaves much like the non-callable non-puttable one (Example 4.3.1. This is fairly reasonable, since in both non-callable non-puttable and pure-puttable cases, the exercise decision structures are strikingly similar: the only actor involved is the investor, seeking to maximize its own gains. Therefore, the volatility effect impacts both cases in a very similar way, with the choice of parameters basis' dimension $M = 3$ and $N_{reg} = 2$, optimal in Setting 2, also yielding excellent results for both LSMC and HMC methods (relative error, in average, $< 1\%$).

Callable cases (pure-callable and callable puttable - Examples 4.4.2 and 4.4.3, respectively) present however a much distinct exercise decision structures: as previously said, 2 actors with conflict interests are now involved, the investor and the issuer, interacting to determine the attributed payoff. In high volatility scenarios ($\sigma = 0.4, 0.5$), both LSMC and HMC methods are not able to correctly capture call features in this complex decision structure. By not capturing call features, the estimated price is driven towards to their non-callable counterpart's

price, leading to a large relative error. This results in a much more intense volatility effect occurring in these cases. In Figures 4.9 and 4.10, are summarized the results for a volatility study, in spirit of the one done in Setting 2, executed for the callable puttable case. Results for the pure-callable case are very similar and are thus supressed.

From Figures 4.9 and 4.10, for both LSMC and HMC methods no combination of parameters is able to stabilize and restrict relative errors to levels below 1%. However, for LSMC method combination of parameters $M = 5$ and $N_{reg} = 50$ constitutes an optimal choice, stabilizing errors between 1 and 2%. On the other hand, for HMC method no combination of parameters is able to stabilize errors. Rather, for all combinations, relative error keeps consistently rising as volatility increases. This indicates HMC method to be much more sensible to volatility effect than LSMC.

In Tables 4.18, 4.19 and 4.20, illustrate results attained with LSMC and HMC methods for Examples 4.4.1, 4.4.2 and 4.4.3 respectively. In all cases, the following parameters were used: Tree method used 1000 time steps, with exercise time steps constrained to the same 100 time steps used by Monte Carlo methods. For Monte Carlo simulations, 201 values in stock price mesh were used, with 10000 paths for the original (central) stock price initial value and 10 for each of the others. For Example 4.4.1, the optimal combination of parameters $M = 3$ and $N_{reg} = 2$ was used. For Examples 4.4.2 and 4.4.3, a combination of parameters $M = 5$ and $N_{reg} = 50$ was used.

First thing to notice from results in Tables 4.18, 4.19 and 4.20 are in consonance with Inequality 4.4.3, indicating attained results to be quite reasonable.

From results in Table 4.18 we see that in the pure-puttable case HMC algorithm yielded slightly better results than LSMC, with smaller errors and narrower confidence intervals. From Tables 4.19 and 4.20, however, we see that in both callable cases (pure-callable and callable puttable) HMC algorithm yielded remarkably worse results than LSMC, with larger errors and wider confidence intervals. This indicates that although hedge functions used in HMC are correctly capturing the optimal hedging strategy for the pure-puttable Convertible Bond, yielding a better price than LSMC did, this was not the case with callable cases. In these latter cases, HMC did not correctly capture optimal hedging strategy, yielding a worse price. For these cases, LSMC yielded not only better results, but also quite good ones, around 1%. This in consonance to our findings in the volatility study, confirming HMC to be more sensible to volatility effect than LSMC is. In all cases, simulations with Milstein discretization gave slightly better results than their Euler-Maruyama counterparts, as expected.

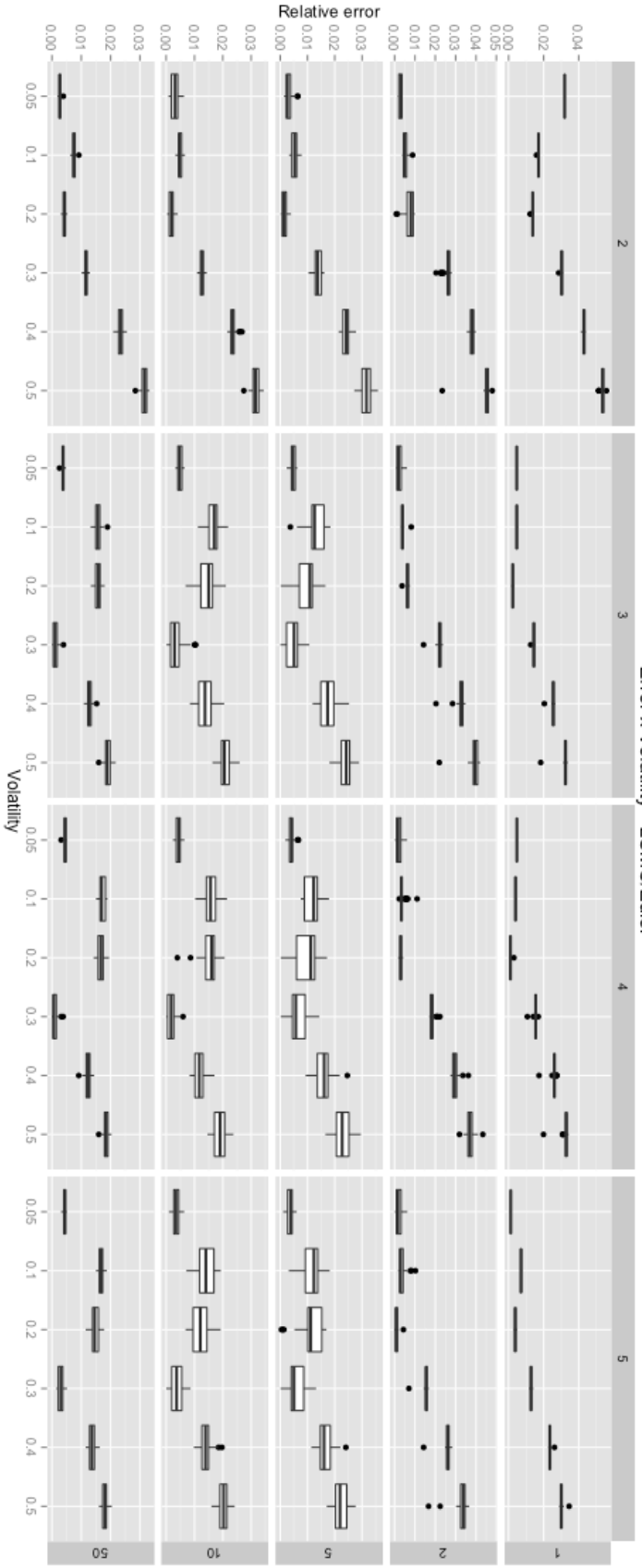

Figure 4.9: Influence of volatility on relative error for LSMC for Example 4.4.3 in Setting 3

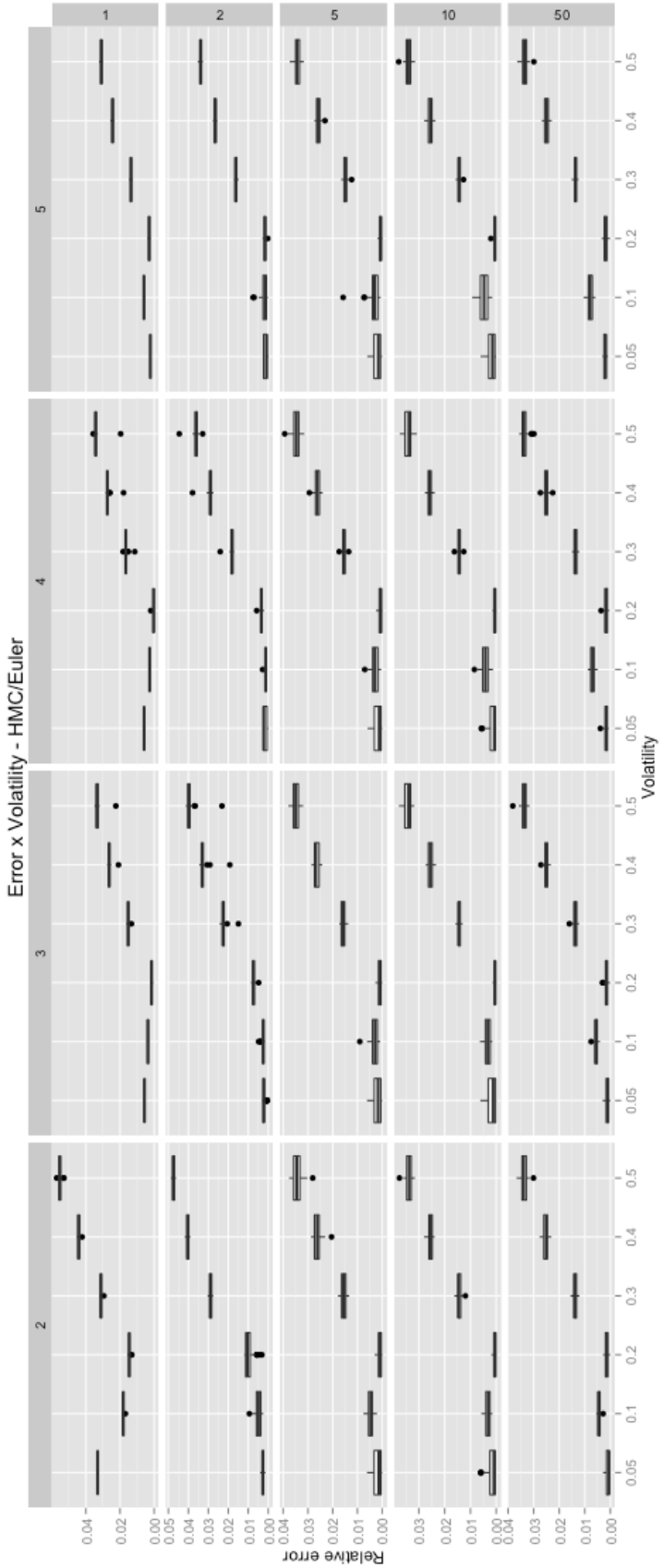

Figure 4.10: Influence of volatility on relative error for HMC for Example 4.4.3 in Setting 3

Reference Prices			
Straight Bond	90.48374		
Black Formula (European Convertible Bond)	105.6615		
Tree (1000 time steps)	110.0798		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Tree)**	**Confidence Interval ($\alpha = 5\%$)**
LSMC / Euler / Hermite M=3	111.1231	0.9477812%	[110.3050, 111.9412]
LSMC / Milstein / Hermite M=3	110.833	0.6841812%	[110.0104, 111.6555]
HMC / Euler / Hermite M=3	110.7936	0.6484659%	[109.9778, 111.6095]
HMC / Milstein / Hermite M=3	110.6945	0.5584301%	[109.8706, 111.5184]

Table 4.18: Results for Example 4.4.1.

Reference Prices			
Straight Bond	90.48374		
Black Formula (European Convertible Bond)	105.6615		
Tree (1000 time steps)	105.8801		
Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Tree)**	**Confidence Interval ($\alpha = 5\%$)**
LSMC / Euler / Hermite M=5	104.6371	1.173985%	[104.5022, 104.7720]
LSMC / Milstein / Hermite M=5	104.6798	1.133631%	[104.5415, 104.8181]
HMC / Euler / Hermite M=5	103.125	2.60207%	[102.9657, 103.2844]
HMC / Milstein / Hermite M=5	103.1265	2.600639%	[102.9664, 103.2867]

Table 4.19: Results for Example 4.4.2.

Next, we examine the shape of the optimal exercise boundary for the callable puttable case (Example 4.4.3). In Table 4.21, we list the criteria used to determine which values compose the exercise boundary for each exercise action, i.e. if a boundary of a given exercise action is an upper bounded region or lower bounded region. .

In Figure 4.11, we present the exercise boundary graph obtained for Tree method with 1000 steps.

Regarding exercise boundary in a callable puttable american Convertible Bond setting, one would naturally expect the following to hold:

- The put boundary should be found below the conversion boundary, since, when the

Reference Prices	
Straight Bond	90.48374
Black Formula (European Convertible Bond)	105.6615
Tree (1000 time steps)	106.5198

Numerical Results			
Method	**Price**	**Absolute Relative Error (w.r.t Tree)**	**Confidence Interval ($\alpha = 5\%$)**
LSMC / Euler / Hermite M=5	104.7858	1.627845%	[104.6685, 104.9031]
LSMC / Milstein / Hermite M=5	105.0619	1.368667%	[104.9537, 105.1701]
HMC / Euler / Hermite M=5	103.6802	2.665783%	[103.5499, 103.8105]
HMC / Milstein / Hermite M=5	103.7686	2.582799%	[103.6378, 103.8994]

Table 4.20: Results for Example 4.4.3.

Action	Type of Boundary Region
Voluntary Conversion	Lower bounded
Put	Upper bounded
Call	Upper bounded
Forced Conversion	Lower bounded
Redemption	Upper bounded

Table 4.21: Boundary regions in Setting 3

stock price is sufficiently low, the investor would exercise the put rather than convert the bond;

- The call boundary should be found below the conversion boundary, since, when the stock price is sufficiently high, the issuer would rather exercise the call rather than let the investor hold the bond for one more period;

- The forced conversion should be found below the voluntary conversion, since, when the stock price is really high, the investor would convert the bond first rather than being forced into conversion by a call exercised by the issuer.

It is easy to notice that the found boundary indeed is consistent with all of above boundary rationale. It is also practically identical (except for separate representation of forced and voluntary conversion) to the one found in [AKW08], the original article that presented Example 4.4.3, indicating it to be seemingly correct.

As with Setting 2, both LSMC and HMC methods produce very poor boundary estimates, regardless of the chosen combination of parameters M and N_{reg}.

Summary of Results and Conclusions

- We have showed how to price a callable puttable american Convertible Bond using both LSMC and HMC methods, including its pure-puttable and pure-callable variants. Both modifications introduced in Setting 2 were also used in this Setting;

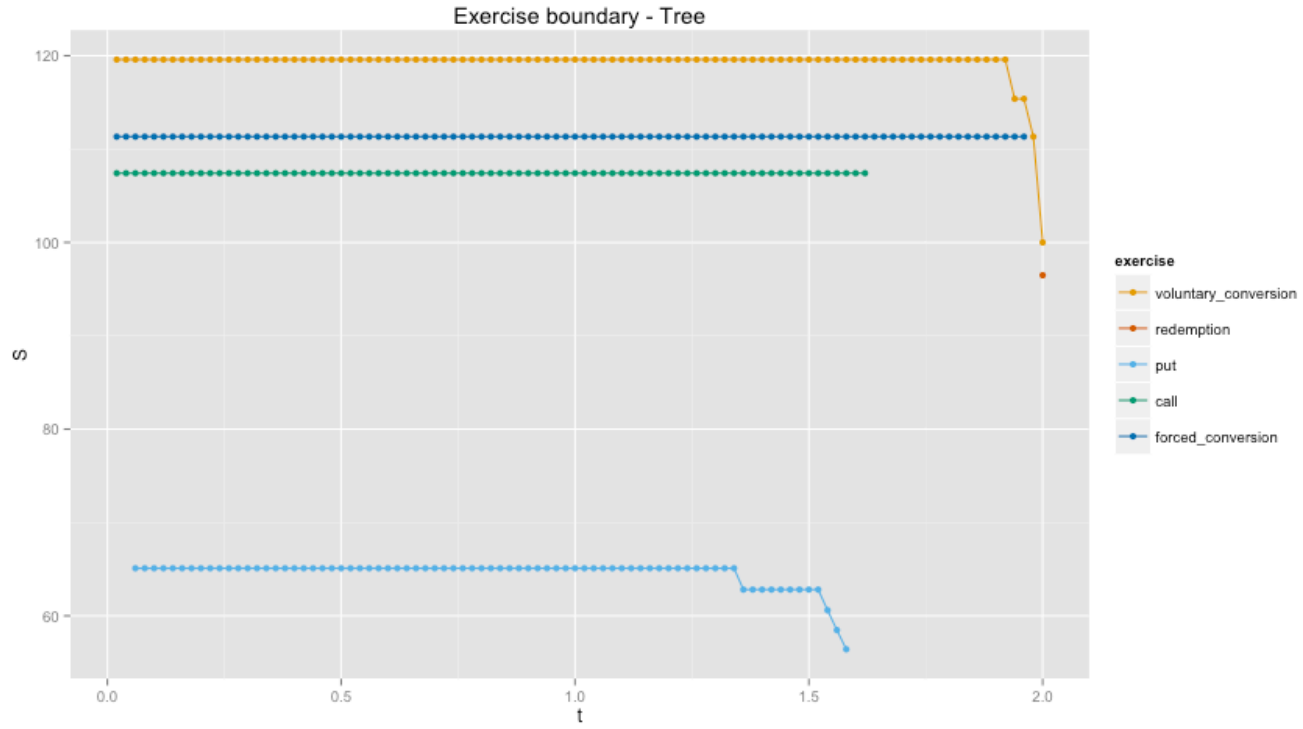

Figure 4.11: Exercise boundary obtained for Example 4.4.3 using Tree method and 1000 time steps

- We have stated and given an intuitive reasoning an an inequality relating prices of a pure-puttable, pure-callable and callable puttable american Convertible Bonds and its non-callable non-puttable counterpart (4.4.3). This inequality gives a bounds for the price of the these Convertible Bond, as well as serving as a sanity check for testing numerical methods;

- Through empirical investigation, we have found prices in this Setting to be also affected by the *volatility effect* mentioned in Setting 2. The pure-puttable case behaves much like the non-callable non-puttable one, from Setting 2. This is attributed to the their exercise decision structures being very similar, since only the investor is involved in determining the instrument's payoff. We have also found that combination of parameters $M = 3$ and $N_{reg} = 2$ also yields excelent results for this bond for both LSMC and HMC methods;

- We have found pure-callable and callable puttable cases to suffer from a much more intense volatility effect. This is attributed to their exercise decision structure being much distinct than pure-callable and non-callable non-puttable ones, since now both investor and issuer, each with conflicting interests, interact to determine the instrument's payoff. A volatily study indicate that HMC method is much more sensible to the volatility effect than LSMC. No combination of parameters M and N_{reg} was deemed optimal for HMC, in the sense of stabilizing and restricting the relative error at low levels. For LSMC, however, combination $M = 5$ and $N_{reg} = 50$ was able to stabilize relative error at levels between 1 and 2%;

- Through examples of callable puttable american Convertible Bond and its pure-puttable and pure-callable variants, we have showed that simulations using Milstein discretization yielded slightly better results than those using Euler-Maruyama, but nothing remarkable. For the pure-puttable case, LSMC and HMC both yielded good results, with HMC yielding slightly better ones. For pure-callable and callable puttable cases, however, HMC yielded remarkably worse results than LSMC, confirming its greater sensibility to volatility effect;

- Finally, we have found that, as in Setting 2, LSMC and HMC methods do yield very poor boundaries when compared to those yielded by the Tree method.

4.5 Setting 4: Path-dependent Callable Puttable American Convertible Bond

Next, we move on to the most general product we shall deal with in this work: a callable and/or puttable american Convertible Bond with path-dependent features.

For modelling this product, we follow the same approach used in both Settings 1, 2 and 3: we choose a stock price-based model, with only the asset's stock price as stochastic, and deterministic and constant interest rate, stock volatility and continuous dividend yield. Market is assumed to be complete and $S(t)$ is such that it follows SDE 3.1.1. For simplification, we also consider the Convertible Bond to be credit-riskless.

Following what was done in Settings 2 and 3, we do the pricing using both LSMC and HMC methods, with both Euler-Maruyama and Milstein discretization. All modifications to backward induction methods presented in Setting 2 are also applied. Since we now have path-dependent features and these would be difficult to evaluate using Tree method, we only evaluate prices with LSMC and HMC methods, not comparing to those attained with Tree method.

Although many types of path-dependent features are possible to be found in a Convertible Bond's indenture, these are usually of 2 types: path-dependent strike price/ratio and path-dependent exercise restriction. Although implementation was made flexible and generic to allow modelling and pricing of any example of these 2 types in all of available exercise types (conversion, call and put), we shall present and discuss along this Setting only with the 2 most commonly found features (see Section 2.2): a reset clause (a type of conversion ratio adjustment clause) and soft call clause (a restriction clause on call exercise).

Payoff at each time is calculated according to Table 4.22. This table is much similar to Table 4.14, presented in Setting 3, except all ratios/strike prices $n(t)$, $C(t)$ and $P(t)$ may now also be path-dependent and sets Ω_{conv}, Ω_{call} and Ω_{put} now embody all restrictions for conversion, call and put, respectively, including those dependent on stock price path. As in Setting 3, all, we require all stochastic processes $\{n(t)\}_{t\in[0,T]}$, $\{C(t)\}_{t\in[0,T]}$, $\{P(t)\}_{t\in[0,T]}$ to be adapted processes w.r.t the filter $\{\mathcal{F}_t\}_{t\in[0,T]} = \{\sigma(W^{\mathbb{Q}}(t)\}_{t\in[0,T]}$.

Payoff	Condition	Restriction	Action
$n(t)S(t)$	if $n(t)S(t) > V'(t)$ and $P(t) \leq n(t)S(t)$	if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{conv}$ if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{put} \cap \Omega_{conv}$	Voluntary Conversion
$P(t)$	if $P(t) > V'(t)$ and $n(t)S(t) < P(t)$	if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{put}$ if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{put} \cap \Omega_{conv}$	Put
$C(t)$	if $V'(t) > C(t)$ and $C(t) \geq n(t)S(t)$	if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{call}$ if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{call} \cap \Omega_{conv}$	Call
$n(t)S(t)$	if $V'(t) > C(t)$ and $C(t) < n(t)S(t)$	if $(t, \{S(u)\}_{u\in[0,t]}) \in \Omega_{call} \cap \Omega_{conv}$	Forced Conversion
κN	if $n(t)S(t) < \kappa N$	if $t = T$	Redemption
$V'(t)$	Otherwise		Continuation

Table 4.22: Exercise actions and respective payoffs in Setting 4

Example 4.5.1. Consider the following path-dependent callable puttable american Convertible Bond example (based on the one proposed in [AKW08]), listed in Table 4.23.

Regular bond features/terms	
Maturity	2 years
Face value	100
First coupon date	-
Coupon frequency	-
Coupon ratio	-
Redemption ratio	100%
Convertible Bond features/terms	
Conversion provisions	**Conversion dates**: 0-2 years (american-styled) **Conversion restrictions**: - **Conversion ratio**: 1 (initially) **Conversion ratio adjustment clauses**: **(Reset clause)** Whenever the mean of the last 20 days' stock prices is greater than 130% of initial stock price, conversion ratio is set to 0.8. Otherwise, it is set to 1
Call provisions	**Strike price**: 110 **Call dates**: 0-2 years (american-styled) **Hard call clause**: - **Soft call clause**: Call may only take place at a given time if mean of the last 20 days' stock price is greater than 110% of initial stock price **Call notice period**: -
Put provisions	**Strike price**: 98 **Put dates**: 0-2 years (american styled)

Table 4.23: Path-dependent callable puttable american Convertible Bond

Consider also the following market scenario for the stock price and interest rate:

$$\begin{cases} r & = 0.05 \\ S_0 & = 100 \\ \sigma & = 0.4 \\ q & = 0.1 \end{cases}$$

The above specified Convertible Bond present both a reset clause and a soft call clause. Both are triggered if, at a given time step, the mean of the last 20 days' stock price is greater than a certain level (130% for reset clause, 110% for soft call clause). For pricing this Convertible Bond using Monte Carlo methods, stock price paths are simulated as usual. Then, at each time step, for each path we evaluate if the above stated clauses are triggered or not, altering related parameters and/or enabling respective parameters. The rest of backward induction algorithms proceed as usual, taking into account the (possibly modified) exercise parameters when computing effective payoff.

In Table 4.24 are shown attained price results with LSMC and HMC methods for above example. In all Monte Carlo simulations, 201 values in stock price mesh were used, with 10000 paths for the original (central) stock price initial value and 10 for each of the others. For both LSMC and HMC methods, following what was found for the callable puttable case in Setting 3, the combination of parameters $M = 5$ and $N_{reg} = 50$ was used. Since we have path-dependent features defined on a daily basis, to avoid approximation errors due to a coarser discretization, $252T = 252x2 = 504$ time steps are used, so that each time step corresponds to a day.

Reference Prices		
Straight Bond	90.48374	
Black Formula (European Convertible Bond)	105.6615	
Numerical Results		
Method	**Price**	**Confidence Interval ($\alpha = 5$%)**
LSMC / Euler / Hermite M=5	111.8555	[111.6100, 112.1011]
LSMC / Milstein / Hermite M=5	111.9981	[111.7534, 112.2429]
HMC / Euler / Hermite M=5	109.6866	[109.4096, 109.9637]
HMC / Milstein / Hermite M=5	109.6723	[109.3951, 109.9494]

Table 4.24: Results for Example 4.5.1.

From Table 4.24, We see that both LSMC and HMC methods were able to estimate a price for the path-dependent callable puttable american Convertible Bond from Example 4.5.1. However, it can as well be seen that prices attained with HMC differ reasonably from those attained with LSMC (around 2% of discrepancy). This may probably be credited to aforementioned *volatility effect*, since from our empirical analysis in Setting 3 we know that HMC is much more sensible to it. As with previous settings, prices attained using Milstein discretization were not much different from Euler ones (< 1% of discrepancy).

Finally, following the empirical analysis developed along previous settings, we are inclined

to accept the price attained with LSMC/Euler method (111.8555) as a fairly good approximate on the true instrument's price.

Summary of Results and Conclusions

- Through an example, we have showed how to price a path-dependent callable puttable american Convertible Bond using both LSMC and HMC methods. We have seen, however, that HMC methods tend to produce fairly different prices, probably due to it being much more sensible to aforementioned *volatility effect*. As with previous settings, prices attained using Milstein discretization were not much different from Euler ones;

- Following the empirical analysis developed along previous settings, we are inclined accept the price attained with LSMC/Euler method as a fairly good approximate on the true instrument's price.

4.6 Calibration Issues and Possible Extensions

In this section, we examine calibration issues related to the proposed model for pricing the Convertible Bond. Also, we explore some extensions/additions that are possible to be made on top of provided pricing framework.

Calibration Issues

Convertible Bonds are non-standardized, very heterogenous and often unique instruments (Section 2.3). Due to these properties, Convertible Bond are generally not sold in exchanges, making them non-liquid instruments, sold in over-the-counter (OTC) trades. Hence, there is generally a lack of price information about these instruments, even for Convertible Bonds from the same firm, making it very difficult for an investor to price it by similarity with other convertibles.

For pricing OTC instruments, what is typically done is to use some model, feeding it with parameters such that the corresponding prices given by the model for more liquid instruments, such as vanilla options, match their practiced prices in the market. The model is said to be calibrated with market data, in the sense that it is in consonance with price values found in the real market. From this point on, one proceeds with using closed-form formulas (if available) or numerical methods for calculating the price of the OTC instrument.

In the context of this work, the Convertible Bond is modeled after a Black-Scholes model (see Section 3.1), with r, σ, S_0, q deterministic and constant. Whilst S_0 and q are observable and well known features from the stock price, r and σ may in reality not be constant or

even deterministic. Next, we propose some straightforward methods for calibrating these parameters.

The seminal article [BS80] states that the interest rate does not heavily influence the Convertible Bond final price. Therefore, it is sufficient to estimate a constant and deterministic r. This can be done by selecting a zero-coupon (government) bond with same maturity as the Convertible Bond, observe its practiced value in the market and, considering a model with deterministic and constant interest rate, determining which r value would give the zero-coupon bond the same value as the one practiced in the market.

Once r is determined, we only need to define σ. If vanilla call and/or put options from the same firm as the issuer of the Convertible Bond are available, one may calculate, using classical Black-Scholes closed-form formula, which constant and deterministic σ gives the same price as those practiced in the market. If vanilla options are not available, one may still find a reasonable value for σ by fitting heteroskedasticity econometric models, such as GARCH, to observed stock price's time series.

Possible Extensions

Stochastic Volatility Models

Throughout this work, in the proposed model we have assumed a constant and deterministic stock volatility. Although the seminal article [BS80] states that stock volatility does not heavily influence the Convertible Bond final price, a possible extension to the presented Monte Carlo pricing framework could be to consider a model where the volatility is not constant and deterministic but stochastic. More precisely, instead of having the stock price dynamics follow SDE 3.1.1, it would follow an SDE system of the form

$$\begin{cases} dS(t) &= S(t)(rdt + \sqrt{\nu(t)}dW_1^{\mathbb{Q}}(t)) \\ d\nu(t) &= \alpha(S,t)dt + \beta(S,t)dW_2^{\mathbb{Q}}(t) \\ S(0) &= S_0 \\ \nu(0) &= \nu_0 \end{cases} \tag{4.6.1}$$

where $W_1^{\mathbb{Q}}(t)$ and $W_2^{\mathbb{Q}}(t)$ are correlated Brownian Motions such that $\langle dW_1^{\mathbb{Q}}(t), dW_2\mathbb{Q}(t)\rangle = \rho$, with $\rho > 0$ a constant, and $\alpha(S,t)$ and $\beta(S,t)$ are such that $\nu(t) > 0 \quad a.s.$.

A simple and very popular stochastic volatility model that could be used is the mean reverting model proposed by Heston et al [Hes93]. In **Heston model**, the SDE System 4.6.1 takes the form

$$\begin{cases} dS(t) & = S(t)(rdt + \sqrt{\nu(t)}dW_1^{\mathbb{Q}}(t)) \\ d\nu(t) & = \upsilon(\theta - \nu(t))dt + \xi\sqrt{\nu(t)}dW_2^{\mathbb{Q}}(t) \\ S(0) & = S_0 \\ \nu(0) & = \nu_0 \end{cases} \tag{4.6.2}$$

where υ, θ and ξ are constants. If $2\upsilon\theta > \xi^2$, then we have $\nu(t) > 0 \quad a.s.$. θ is the **long-term variance**; υ is the **mean revertion rate**, i.e., the rate at which $\nu(t)$ reverts to the long-term variance; and ξ is the **volatility of the volatility (vol of the vol)**.

Incorporating SDE System 4.6.2 into the presented Monte Carlo pricing framework should be fairly straightforward: one would only need to alter the procedure that generates the stock price paths in such a way that they now obey this SDE system. This can be easily done by discretizing both SDEs in SDE System 4.6.2 using the same discretization scheme (e.g. Euler-Maruyama or Milstein schemes), and evaluating at each time t the stock volatility $\nu(t)$ and then feeding it in the other equation to calculate $S(t)$.

Stochastic Interest Rate Models

As with stochastic volatility, seminal article [BS80] also states that interest rates do not heavily influence the Convertible Bond final price. Thus, we have also assumed in our proposed model a constant and deterministic interest rate. However, one may be interested in extending the provided pricing framework to a setting where the interest rate follows a given term structure. Although non-constant but deterministic interest rates are also possible to be used, it is more common to consider interest rates that follow an stochastic model.

In general, we would have the stock price dynamics follow a SDE System of the form

$$\begin{cases} dS(t) & = S(t)(r(t)dt + \sigma dW_1^{\mathbb{Q}}(t)) \\ dr(t) & = \lambda(r,t)dt + \zeta(r,t)dW_2^{\mathbb{Q}}(t) \\ S(0) & = S_0 \\ r(0) & = r_0 \end{cases} \tag{4.6.3}$$

where $W_1^{\mathbb{Q}}(t)$ and $W_2^{\mathbb{Q}}(t)$ are independent Brownian Motions and $\lambda(S,t)$ and $\zeta(S,t)$ are such that $r(t) > 0 \quad a.s.$.

A simple and popular stochastic interest rate model that could be used is the mean reverting model proposed by Cox, Ross and Ingersoll [CJR85]. The **Cox-Ross-Ingersoll (CIR) model** is essentially analogous to Heston stochastic volatility model. In CIR model, the SDE system 4.6.3 takes the form

$$\begin{cases} dS(t) & = S(t)(r(t)dt + \sigma dW_1^{\mathbb{Q}}(t)) \\ dr(t) & = a(b-r)dt + \zeta\sqrt{r(t)}dW_2^{\mathbb{Q}}(t) \\ S(0) & = S_0 \\ r(0) & = r_0 \end{cases} \tag{4.6.4}$$

where a, b and ζ are constants. If $2ab > \zeta^2$, then we have $r(t) > 0 \quad a.s.$. b is the **long-term interest rate mean level**; a is the **mean revertion rate**, i.e., the rate at which $r(t)$ reverts to the long-term interest rate mean level; and ζ is the **volatility of the interest rate**.

As with Heston stochastic volatility model, incorporating SDE System 4.6.4 into the presented Monte Carlo pricing framework should also be fairly straightforward: one should adapt the procedure that generates the stock price paths in such a way that they now obey this SDE system. This must also be done by discretizing each of its SDEs using a discretization scheme and evaluating at each time t the interest rate $r(t)$ and feeding it in the other equation to calculate $S(t)$.

It should also be noted that, unlike with stochastic volatility, in this case the interest rate paths must also be retained because they are used along backward induction process to discount each value at each time t using its current interest rate $r(t)$. This, however, should also be pretty straightforward.

Credit-Risky Models

Throughout this work, we have assume a credit-riskless model for our Convertible Bond. We know from the seminal article [BS80] that credit risk does heavily influence the Convertible Bond price. Thus, a very natural extension one shall consider is to include credit risk in our proposed model.

Modelling credit risk involves investigating **default events**, i.e. events where the issuer does not pay some coupon or principal in full or in due time. Although clearly not the only cause for defaults, the most common one and which has been the object of many research of last few years is that of **bankruptcy**, when the issuer cannot meet its debt obligations anymore[12]. Thus, many credit risk modelling choose to abstract away other causes and focus only in modelling default events due to bankruptcy.

When a default event takes place, both parts of the Convertible Bond are affected: the bond part, because the firm will not honor future coupons and principal[13], and also the stock part, because the stock value of the firm also falls, possibly becoming worthless (i.e. $S = 0$) . Thus, one need to model how frequently this default events occur and what happens to both bond and stock parts of the Convertible Bond.

[12] Bankruptcy does not necessarily imply **insolvency**. The latter one is the financial lack of liquidity to repay its debts, whereas the former is a legal status of not being able to honor its debts, imposed by some court. For sake of simplicity, in this work when using the word bankruptcy we assume a insolvency state.

[13] Although the firm will not honor any future coupons and principal, the identure however may specify that

There are 2 main types of approaches when it comes to default event modelling: exhogenous default and endogenous default.

An **exhogenous default model** sees a default event as an exhogeneous event that happens outside the pricing model, trying to model its probability and effects. It then seeks to incorporate this default event treatment into the model by means of including a spread s into the discounting factor of the bond payments. If a firm has higher risk of default, the spread increases and with it also the discounting factor, rendering a less valued Convertible Bond. If a firm has no risk of default at all, the model reduces to the one presented in this work. The credit spread itself may be calibrated externally with historical data of defaults of that firm (if available) or by similarity with other firms of similar size and within the same industry sector.

This exhogenous model is very simplistic and, although the credit spread may be used to discount also the payoff part, it does not account for the possibility of the stock price falling in value.

In contrast, an **endogenous default model** seek to incorporate the probability and effects of default explicitly in the pricing model. There have been presented many approaches to do this (for instance, see [TF98], [TKN01], [AFV03], [BW03] and [MK12]). One that fits well a Monte Carlo pricing method is to incorporate defaults explicitly in the SDE modelling, through jump-diffusion processes. Jump-diffusion processes are a more complex topic and are outside the scope of this work, to be subject of study and implementation in a future work. For an example of jump-diffusion approach in credit-risky models, refer to [MK12].

For a good review on credit-risky models for Convertible Bonds, refer to [Zad10].

in case of default the investor is assured to recover a certain fraction of the underlying bond value. This is specified in terms of a **recovery ratio** (sometimes called **recovery factor**) and is generally operationalized by means of insurances or guarantees contracted by the issuer and presented at the time of issuance.

Chapter 5

Conclusion

In this work, we have presented the concept of Convertible Bonds and its related features and terms, discussing its pricing problem and related issues. Using a stock value-based model and Monte Carlo methods, we have successfully demonstrated, in a constructive manner, how to model and price Convertible Bonds, starting from a simple, non-callable non-puttable european setting, towards a more complex, callable puttable american setting.

We believe have succesfully demonstrated how to use both LSMC and HMC methods to cope with american exercise features, highlighting their high flexibility, which allows them to also cope with more rich features, such as path-dependent ones. Whenever possible, to benchmark these methods we have compared their estimated price with equivalent ones from (binomial) Tree method, exhamining attained relative errors. From given Convertible Bond examples, we have founded HMC method to give marginally better results than LSMC method, not justifying its use. Likewise, we have found Milstein discretization to yield slightly better results than Euler-Maruyama discretization ones, not justitying the use of the former.

Another important contribution of this work is that, whilst demonstrating the use of LSMC and HMC methods, through empirical analysis we have incidentally also uncovered 2 important drawbacks of these methods: high sensibility of estimated prices to stock volatility scenarios and poor estimation of exercise boundaries. We have found that use of Bouchard's technique of executing several regressions instead of just one, each over a partition of stock price data available at that time step, has helped alleviate this volatility effect problem. For given examples, we have executed a volatility study, indicating optimal choice of parameters basis' dimension and number of regression intervals in the sense of minimizing relative errors. Although found optimal choices are highly dependent on given example, we believe they provide a reasonable starting point for other Convertible Bond instances from the same setting.

From a more practical perspective, another important contribution of this work is a concrete, clear and ready-to-use code implementation of LSMC and HMC pricing methods. It was built from the ground up in a parametrizable way, in such a way it can be easily adapted to cope with more rich Convertible Bond settings, such as those including time and path-dependent features. The provided pricing framework may also be easily adapted to price a range of

other instruments, from simple, vanilla instruments to more exotic derivatives.

We believe to have accomplished all its proposed goals, demonstrating, in a constructive way, the applicability, strengths and drawbacks of LSMC and HMC backward induction methods for pricing Convertible Bonds. As many questions risen along this work fell outside its scope, we leave the following suggestions for future works:

- Provide a formal proof of bounds for Convertible Bond prices presented in Sections 4.3 and 4.4;
- Investigate further and provide a formal mathematical description of presented *volatility effect* on prices and relative errors found for LSMC and HMC methods;
- Execute broader and systematic studies on best choices of simulation parameters across general examples of Convertible Bonds for proposed settings.
- Investigate further reasons behind the presented phenomenon of poor estimation of boundary exercises by LSMC and HMC methods, providing modifications of their algorithms such as to enable its use for boundary estimation;
- Provide modifications of LSMC and HMC algorithms to make them faster to compute and more precise, so as to enable execution of broader and more general simulation parameter studies.

Appendix A

Implementation

In this chapter, we present the implementation approach, discussing some of the implementation issues found and related implementation decisions.

In Section A.1, the choice of platform/programming language for this work implementation is discussed.

In Section A.2, the implementation approach itself is presented, with its issues presented and discussed as they arise.

Finally, in Section A.3, we make a final remark about the provided implementation.

A.1 Choice of Platform/Programming Language

In Finance field, there are some platforms/programming laguanges that have been more popular for implementing mathematical finance code: Microsoft Excel/VBA [Exc], MATLAB [MAT] and R [R C13]. Of course more general programming languages, like C, C++, Python, Ruby can be used, but as one of our main goals is to provide a clear and more ready-to-use implementation code (see Section 1.2), we restricted our platform analysis to the former.

For the implementation of this work, the chosen platform/programming language was R. The rationale behind this decision is presented below.

1. For this work, we needed good vector/matrix manipulation syntax, as well as numerical computation and pseudo-random number generation routines. Both 3 have numerical computation and pseudo-random number generation routines available, but Excel/VBA's are somewhat very limited, when compared to MATLAB and R. Also, Excel/VBA, because of its Visual Basic inheritance, presents a *clumsy* vector syntax and lacks full matrix operation support, whilst both MATLAB and R have both good vector/-

matrix support [1]. Hence, Excel/VBA was discarded;

2. Both MATLAB and R have a well stablished user base, with plenty example code, tutorials and libraries available. MATLAB would have been a more natural choice, as it has been more heavily used in industry. But whilst a great platform, MATLAB is a proprietary product with many separately sold toolboxes, requiring licenses for commercial use. On the other hand, R is completely free and open source. As we did not want to impose any cost burden to use this work code right away, MATLAB was discarded;

3. Also, being an open source platform/programming language, R has more bindings to other programming languages, specially C, C++, Ruby, Python, Java, which makes it easier to integrate with other systems and products.

It must be kept in mind that the following arguments reflect the vision of the author at the time of this work writing and thus may sound kind of opinionated. Every platform has its own strengths and weaknesses. The reader is thus encouraged to first look well into the problem and available facilities they have, to only them choose the more suited platform.

The R version used in this work was 3.0.2, the up-to-date version as of the time of this work writing. The R libraries used will be presented as needed in the next section, when the code is thoroughly discussed.

Before stepping into implementation itself, it must be noted that it is not in the scope of this work to explain basic programming concepts nor R syntax. If one needs to learn programming concepts and R programming language, the excellent books [Cot13], [Tee11] and [Iac09] are advised.

A.2 Implementation Approach

Although in Chapter 4, we discuss 4 settings and for each one the implementation was done in separate files, as latter settings extend the former ones and implementation structure was pretty much the same in terms of file, for the sake of conciseness, we present and discuss here only the implementation of the Setting 4 (4.5). Implementations related to Tree method and evaluation of exercise boundaries are also ommited, since we do not use them on Setting 4. A reader, however, should find no difficulty to implement these things.

The implementation approach was to have a main script file (`CB_price.r`), which include auxiliary R's files to deal with specific portions of the implementation, such as bond-related functions (`bond.r`), payoff-related functions (`payoff.r`), SDE simulation (`sde.r`) and backward induction Monte Carlo methods (`backward_induction.r`).

[1] In fact, R has also a rather *awkward* syntax when it comes to matrix/vector operations. It is not uncommon for a specific command line, such as multiplying a matrix by a vector column-wise, to not throw an error, but to work in an unexpected way. In general, one should always be careful with this type of operation and backtest them.

A note about performance

Before stepping into the implementation itself, a disclaimer about performance: the goal of this work was to provide a concise, clear, flexible, extensible and ready-to-use implementation. The work focused on LSMC and HMC algorithms. To make algorithms perfectly intelligible, clear code was preferred over obscure one. Thus, use of cryptic constructs, R's arcane features and C code for heavy lifting work were firmly avoided, at expense of performance.

At some point, however, performance was so poor that the provided implementation was unusable in real world scenarios, taking around 40 to 60 mins of processing to fully compute LSMC/Euler, LSMC/Milstein, HMC/Euler and HMC/Milstein prices in Settings 2 and 3, with 100 time steps and 14000 Monte Carlo paths. Optimizations then had to be made to ensure a better performance, but these were handpicked as to not compromise code clearness. Examples of such optimizations included functions' simplication, substitution of calls of heavy functions for equivalent but lighter ones, caching of heavily used data structures and values and pre-allocating matrix and vectors whenever possible. Ultimately, the most CPU intensive functions were precompiled into byte code and its compiled version was used along the main script. All these optimizations helped to attain a dramatically better performance, with implementation taking now around 5 minutes in Setting 3 and 7 minutes in Setting 4 to complete all price computation for the same number of time steps and Monte Carlo paths[2].

A.2.1 Main code: CB_price.r

```
# Author: Leandro Loriato (llrt at impa.br)
#
# Calculates the price of a Convertible Bond in setting 3, in which we have:
# - Complex product (callable, puttable, american-style conversion)
# - Simplified model (only asset is stochastic, do not consider credit risk issues)
#

library(compiler)

# import auxiliary functions
source('bond.r')
source('payoff.r')
source('sde.r')
source('backward_induction.r')
source('boundary.r')


# general parameters
t0 = 0 # initial time
T = 2 # final time
N.t = 100 #252*2 # number of discretization steps in time
dt = (T-t0)/N.t # calculates time discretization step
```

[2] All above stated processing times were measured using a 2011 Early Macbook Pro hardware with a 2.3 GHz Intel Core i5 processor and 8 GB 1333 MHz DDR3 RAM.

```
23
24
25 # bond model
26 r = 0.05 # assuming deterministic risk-free interest rate
27 bond.principal = 100 # bond's principal
28 bond.redemption.ratio = 1 # ratio at which principal is redempted at maturity
29
30 bond.coupon.frequency = 0.5 # frequency on which coupons are paid (assuming equally spaced coupons)
31 bond.coupon.dates = seq(t0 + bond.coupon.frequency, T, by=bond.coupon.frequency) # coupon dates,
       starting from payment period
32                                                    # immediately after t0 to, and including, T
33 bond.coupon.rate = 0 # coupon rate to be paid on bond's principal (assuming equally valued coupons and
       pre-fixed rate)
34 # combine coupon parameters into one data structure
35 bond.coupon = list(frequency=bond.coupon.frequency, dates=bond.coupon.dates, rate=bond.coupon.rate)
36
37 # number of stocks a convertible bond's may be exchanged for
38 bond.conversion.ratio = function(t, S.vector){
39   # if mean of stock prices of the last 20 last days (including current one) is greater
40   # than 130% of initial stock price, then conversion ratio is reset to 0.8
41   n = length(S.vector)
42   mean.S = mean(S.vector[max(1, n-19):n])
43
44   if(mean.S > 1.3*S0){
45     conversion.ratio = 0.8 # sets conversion ratio to 0.8
46   } else{
47     conversion.ratio = 1 # sets conversion ratio to 1
48   }
49
50   invisible(conversion.ratio)
51 }
52 bond.conversion.dates = seq(t0, T, by=dt) # assuming american-style (conversion possible at each time)
53 bond.conversion.restriction = function(t, S.vector){ # embodies restrictions, beside those time-related,
54                                     # that must be met for conversion action
55                                     # to be allowed (can be modified to encompass other types of
56                                     # restrictions, e.g. of Contingent Convertible Bonds)
57   invisible(TRUE) # no conversion restriction
58 }
59
60 # combine coupon parameters into one data structure
61 bond.conversion = list(ratio=bond.conversion.ratio, dates=bond.conversion.dates, restriction=bond.
       conversion.restriction)
62
63 # call optionality:
64 # issuer may choose to end contract before maturity, forcing the investor the option to exchange the
       convertible
65 # bond for given strike price or to convert it (if current time is one of conversion dates)
66 bond.call.present = TRUE # whether a call optionality is present or not
67 bond.call.strike = function(t, S.vector){ # call's strike price
68   call.strike = 110 # assuming constant over time
69
70   invisible(call.strike)
71 }
72 bond.call.dates = seq(t0, T, by=dt) # dates where call optionality may take place
73 bond.call.restriction = function(t, S.vector){ # embodies restrictions, beside those time-related,
74                                   # that must be met for call action to be allowed
75                                   # (can be modified to encompass other types of restrictions)
76
```

```
  # if mean of stock prices of the last 20 days (including current one) is greater
  # than 110% of initial stock price, then call exercise is allowed
  n = length(S.vector)
  mean.S = mean(S.vector[max(1, n-19):n])

  if(mean.S > 1.1*S0){
    ret = TRUE # call exercise is allowed
  } else{
    ret = FALSE # call exercise is not allowed
  }

  invisible(ret)
}
# combine call parameters into one data structure
bond.call = list(present=bond.call.present, strike=bond.call.strike, dates=bond.call.dates, restriction=bond.call
    .restriction)


# put optionality:
# investor may choose to end contract before maturity, forcing the issuer to buy the convertible bond for
    given
# strike price
bond.put.present = TRUE # whether a put optionality is present or not
bond.put.strike = function(t, S.vector){ # put's strike price
  put.strike = 98 # assuming constant over time
  invisible(put.strike)
}
bond.put.dates = seq(t0, T, by=dt) # dates where put optionality may take place
bond.put.restriction = function(t, S.vector){ # embodies restrictions, beside those time-related,
                              # that must be met for put action to be allowed
                              # (can be modified to encompass other types of restrictions, e.g. of
                              # Contingent Convertible Bonds)
  invisible(TRUE)
}
# combine put parameters into one data structure
bond.put = list(present=bond.put.present, strike=bond.put.strike, dates=bond.put.dates, restriction=bond.put.
    restriction)


# creates a data structure to contain all relevant bond parameters
bond.params = list(principal=bond.principal, redemption.ratio=bond.redemption.ratio,
              coupon=bond.coupon, conversion=bond.conversion,
               call=bond.call, put=bond.put)


# config about which prices, given by each method, to calculate
calculate.LSMC.euler = TRUE # whether to calculate price given by LSMC/Euler or not
calculate.LSMC.milstein = TRUE # whether to calculate price given by LSMC/Milstein or not
calculate.LSMC = list(euler=calculate.LSMC.euler,
                      milstein=calculate.LSMC.milstein)
calculate.HMC.euler = TRUE # whether to calculate price given by HMC/Euler or not
calculate.HMC.milstein = TRUE # whether to calculate price given by HMC/Milstein or not
calculate.HMC = list(euler=calculate.HMC.euler,
                      milstein=calculate.HMC.milstein)
# creates a data structure to contain all relevant graph config about which prices to calculate
calculate = list(LSMC=calculate.LSMC,
              HMC=calculate.HMC)

```

```
# stock model
S0 = 100 # initial value for S
q = 0.1 # continuous dividend yield
# SDE of form dX(t) = drift(t, X) dt + sigma(t, X) dW(t)
.drift = (r-q) # stock drift equals the risk-free interest rate in risk-neutral measure
drift = expression(.drift*x) # expression for the drift
.sigma = 0.4 # stock volatility
sigma = expression(.sigma*x) # expression for the sigma


# Monte Carlo parameters
S0.delta = 0.8*S0 # distance of original S0 to maximum/minimum value in the array
S0.max = S0 + S0.delta  # maximum S0 value in initial stock price's array
S0.min = S0 - S0.delta # minimum S0 value in initial stock price's array
N.S0.band = 100 # number of values in each band of the array
N.S0 = 2*N.S0.band # number of values in the whole array (excluding the central value, i.e. original one)
d.S0 = (S0.delta)/N.S0.band # stock price step in the array
S0.array = seq(S0.min, S0.max, by=d.S0) # initial stock price's array
S0.array.without.central = S0.array[-(N.S0.band+1)] # initial stock price's array without central value

N.MC.central = 10000 # number of Monte Carlo simulations for original initial stock price value
N.MC.band = 10 # number of Monte Carlo simulations for rest of initial stock price's array

price.fun.LSMC = basis(type="hermite", M=5) # LSMC's basis functions for instrument price
price.fun.HMC = basis(type="hermite", M=5) # HMC's basis functions for instrument price


# generating paths
Z = matrix(rnorm(N.t*(N.MC.central + N.MC.band*N.S0)), ncol=(N.MC.central + N.MC.band*N.S0), nrow=N.t)
      # generates N(0,1) samples

if(any(calculate$LSMC$euler, calculate$HMC$euler)){ # simulate Euler-Maruyama paths only if its
  # corresponding prices are to be calculated
  # simulates the solution for the given SDE using Euler-Maruyama method

  S.euler = matrix(ncol=(N.MC.central + N.MC.band*N.S0), nrow=(N.t+1)) # pre-allocates matrix for S

  for(.S0 in S0.array.without.central){ # for each value in the initial stock price's array, except the original one
    # generates N.MC.band stock price paths
    j = which(S0.array.without.central==.S0) # obtain index of current stock price in array without central
    indexes = ((j-1)*N.MC.band+1):(min(j*N.MC.band, N.MC.band*N.S0))
    S.euler[,indexes] = SDE.solve(t0=t0, T=T, X0=.S0, N.t=N.t, drift=drift, sigma=sigma, method="euler", Z=Z[,
      indexes])
  }

  # finally, for the original initial stock price, generates N.MC.central stock price paths
  indexes = (N.MC.band*N.S0+1):(N.MC.central + N.MC.band*N.S0)
  S.euler[,indexes] = SDE.solve(t0=t0, T=T, X0=S0, N.t=N.t, drift=drift, sigma=sigma, method="euler", Z=Z[,
      indexes])
}

if(any(calculate$LSMC$milstein, calculate$HMC$milstein)){ # simulate Milstein paths only if its
  # corresponding prices are to be calculated
  # simulates the solution for the given SDE using Milstein method

  S.milstein = matrix(ncol=(N.MC.central + N.MC.band*N.S0), nrow=(N.t+1)) # pre-allocates matrix for S

  for(.S0 in S0.array.without.central){ # for each value in the initial stock price's array, except the original one
```

```
187     # generates N.MC.band stock price paths
188     j = which(S0.array.without.central==.S0) # obtain index of current stock price in array without central
189     indexes = ((j-1)*N.MC.band+1):(min(j*N.MC.band, N.MC.band*N.S0))
190     S.milstein[,indexes] = SDE.solve(t0=t0, T=T, X0=.S0, N.t=N.t, drift=drift, sigma=sigma, method="milstein",
          Z=Z[,indexes])
191   }
192
193   # finally, for the original initial stock price, generates N.MC.central stock price paths
194   indexes = (N.MC.band*N.S0+1):(N.MC.central + N.MC.band*N.S0)
195   S.milstein[,indexes] = SDE.solve(t0=t0, T=T, X0=S0, N.t=N.t, drift=drift, sigma=sigma, method="milstein", Z
        =Z[,indexes])
196 }
197
198 # bond price at time T
199 bond.T = bond.value.T(bond.params, r)
200
201
202 # pricing
203 # pricing with LSMC algorithm
204 if(calculate$LSMC$euler){ # calculate LSMC/Euler price only if it is to be calculated
205   ret.LSMC.euler = price.MC(S=S.euler, S0=S0, r=r, q=q, t0=t0, T=T, N.t=N.t, N.MC=(N.MC.central + N.MC.
        band*N.S0),
206                             backward.induction.method="LSMC", price.fun=price.fun.LSMC, bond.params=bond.
        params,
207                             regression.method="bouchard", regression.intervals=50)
208   price.LSMC.euler = ret.LSMC.euler$MC
209   exercise.LSMC.euler = ret.LSMC.euler$exercise
210 }
211
212 if(calculate$LSMC$milstein){ # calculate LSMC/Milstein price only if it is to be calculated
213   ret.LSMC.milstein = price.MC(S=S.milstein, S0=S0, r=r, q=q, t0=t0, T=T, N.t=N.t, N.MC=(N.MC.central + N.
        MC.band*N.S0),
214                              backward.induction.method="LSMC", price.fun=price.fun.LSMC, bond.params=bond.
        params,
215                              regression.method="bouchard", regression.intervals=50)
216   price.LSMC.milstein = ret.LSMC.milstein$MC
217   exercise.LSMC.milstein = ret.LSMC.milstein$exercise
218 }
219
220 # pricing with HMC algorithm
221 if(calculate$HMC$euler){ # calculate HMC/Euler price only if it is to be calculated
222   ret.HMC.euler = price.MC(S=S.euler, S0=S0, r=r, q=q, t0=t0, T=T, N.t=N.t, N.MC=(N.MC.central + N.MC.
        band*N.S0),
223                            backward.induction.method="HMC", price.fun=price.fun.HMC, bond.params=bond.params,
224                            regression.method="bouchard", regression.intervals=50)
225   price.HMC.euler = ret.HMC.euler$MC
226   exercise.HMC.euler = ret.HMC.euler$exercise
227 }
228 if(calculate$HMC$milstein){ # calculate HMC/Milstein price only if it is to be calculated
229   ret.HMC.milstein = price.MC(S=S.milstein, S0=S0, r=r, q=q, t0=t0, T=T, N.t=N.t, N.MC=(N.MC.central + N.
        MC.band*N.S0),
230                             backward.induction.method="HMC", price.fun=price.fun.HMC, bond.params=bond.params
        ,
231                             regression.method="bouchard", regression.intervals=50)
232   price.HMC.milstein = ret.HMC.milstein$MC
233   exercise.HMC.milstein = ret.HMC.milstein$exercise
234 }
235
```

```
236
237 # Black's formula pricing
238 price.black_ = price.black(r=r, q=q, t=0, T=T, S=S0, sigma=.sigma, bond.params=bond.params)
239
240
241 # combine obtained prices into one data structure
242 price.LSMC_ = list()
243 if(calculate$LSMC$euler){ # adds LSMC/Euler price to structure if it was to be computed
244   price.LSMC_$euler = price.LSMC.euler$value
245 }
246 if(calculate$LSMC$milstein){ # adds LSMC/Milstein price to structure if it was to be computed
247   price.LSMC_$milstein = price.LSMC.milstein$value
248 }
249
250 price.HMC_ = list()
251 if(calculate$HMC$euler){ # adds HMC/Euler price to structure if it was to be computed
252   price.HMC_$euler = price.HMC.euler$value
253 }
254 if(calculate$HMC$milstein){ # adds HMC/Milstein price to structure if it was to be computed
255   price.HMC_$milstein = price.HMC.milstein$value
256 }
257
258 price = list(black=price.black_, LSMC=price.LSMC_, HMC=price.HMC_)
```

Listing A.1: CB_price.r

This is the main script file. In lines $8 - 16$, the relevant auxiliary functions, declared in other files, are imported. The `compiler` library is also imported to provide functions for precompilation of CPU intensive functions.

In lines $18 - 23$, some general, time-related parameters such as maturity and number of time steps are configured.

In lines $25 - 111$, the bond model is configured. Special attention was taken to make the model as parametrizable as possible: interest rate, bond's principal, redemption ratio, coupons' rate, frequency and payment dates, conversion ratio and dates, call and put strike price and dates can be configured. Dates are configured as vectors to provide high flexibility and allow for discrete exercise time structure. Also, conversion, call and put ratio/strike price values and restrictions were implemented as functions so as to allow for time- and path-dependent variants of these features. Finally, all bond parameters are combined into one data structure for convenience.

In lines $119 - 131$, some parameters are created to allow the user to inform for which methods, LSMC/Euler, LSMC/Milstein, HMC/Euler and HMC/Milstein, prices must be calculated.

In lines $132 - 140$, the stock model is configured. Initial stock price, stock's continuous dividend yield and volatility can be configured.

In lines $142 - 156$, Monte Carlo parameters, such as the number of paths, may be configured. Implementation was made so as to allow the use of an array of several initial stock prices, with possibly different numbers of paths to be generated for original initial stock price and the others. Finally, different basis functions may be configured for LSMC and HMC

methods.

In lines $159 - 197$, the standard normal samples are generated and Monte Carlo paths for both Euler and Milstein discretizations are then simulated.

Finally, in lines $198-258$, pricing itself is done for LSMC/Euler, LSMC/Milstein, HMC/Euler and HMC/Milstein. Actual price calculation is done only for methods for which user configured to be done. Implementation was made to allow regression use of regular (monolithic) regression or Bouchard's regression, with parametrizable number of regression intervals. Finally, a reference price for the European Convertible Bond is calculated using Black formula.

A.2.2 Bond-related Functions: bond.r

```
1  # Author: Leandro Loriato (llrt at impa.br)
2  #
3  # Auxiliary functions to calculate the value of a bond at a given time
4
5
6  discounted.coupon = function(bond.params, t, r){
7    payments = c() # initialize vector of made payments as empty
8
9    for (t_ in bond.params$coupon$dates){
10     # adds discounted value of accrued coupons to payments vector
11     payments = c(payments, exp(-r*(t_-t)) * bond.params$coupon$rate * bond.params$principal)
12   }
13
14   value = sum(payments)
15   invisible(value)
16 }
17
18 # calculates the redemption value of a bond, given relevant parameters
19 redemption = function(bond.params, r){
20   # redemeed value is the principal times the redemption ratio
21   value = bond.params$principal * bond.params$redemption.ratio
22   invisible(value)
23 }
24 # for performance, pre-compile CPU intensive function
25 redemption = cmpfun(redemption)
26
27 # calculates the value of a bond at time T, given relevant parameters
28 bond.value.T = function(bond.params, r){
29   # value of the bond at time T is the redemption value
30   value =  redemption(bond.params, r)
31
32   invisible(value)
33 }
```

Listing A.2: bond.r

In this file, bond-related functions such as bond's discounted coupons and redemption value are presented.

A.2.3 Payoff-related Functions: payoff.r

```
# Author: Leandro Loriato (llrt at impa.br)
#
# Auxiliary functions to calculate the payoff of a Convertible Bond given the relevant parameters


# legend of exercise possibilities
exercise.legend = c("continuation"=0,
                    "voluntary_conversion"=1,
                    "put"=2,
                    "call"=3,
                    "forced_conversion"=4,
                    "redemption"=5)

# given a time step, computes its exercise time restriction bitmap,
# evaluating which exercise actions' time restrictions are met
exercise.time.bitmap = function(t, bond.params){
  ret = list()
  ret$conversion = FALSE
  ret$put = FALSE
  ret$call = FALSE
  ret$redemption = FALSE

  # conversion
  if(t %in% bond.params$conversion$dates){
    ret$conversion = TRUE
  }
  # put
  if(bond.params$put$present && t %in% bond.params$put$dates){
    ret$put = TRUE
  }
  # call
  if(bond.params$call$present && t %in% bond.params$call$dates){
    ret$call = TRUE
  }
  # redemption
  if(t==T){
    ret$redemption = TRUE
  }

  invisible(ret)
}
# for performance, pre-compile CPU intensive function
exercise.time.bitmap = cmpfun(exercise.time.bitmap)


# calculates a payoff value for the convertible bond at a time, given a stock price,
# a reference value and other relevant parameters
payoff = function(t, S.vector, r, continuation.value, exercise.time.bitmap, bond.params){

  # initialize auxiliary variables
  S = tail(S.vector, n=1)
  action = 'continuation'
  conversion.value = bond.params$conversion$ratio(t, S.vector) * S
  call.value = bond.params$call$strike(t, S.vector)
  put.value = bond.params$put$strike(t, S.vector)
  redemption.value = redemption(bond.params, r)
```

```
57
58  if(bond.params$conversion$restriction(t, S.vector) && exercise.time.bitmap$conversion && conversion.
        value > continuation.value){
59      if(bond.params$put$restriction(t, S.vector) && exercise.time.bitmap$put && put.value > conversion.value)
         {
60          action = 'put'
61      } else{
62          action = 'voluntary_conversion'
63      }
64  } else if(bond.params$put$restriction(t, S.vector) && exercise.time.bitmap$put && put.value > continuation.
        value){
65      action = 'put'
66  } else if(bond.params$call$restriction(t, S.vector) && exercise.time.bitmap$call && continuation.value > call.
        value){
67      if(exercise.time.bitmap$conversion && conversion.value > call.value){
68          action = 'forced_conversion'
69      } else{
70          action = 'call'
71      }
72  } else if(exercise.time.bitmap$redemption && redemption.value > conversion.value){
73      action = 'redemption'
74  }
75
76
77  # based on which action takes place, evaluates corresponding payoff
78  switch(action,
79         'voluntary_conversion' = { # voluntary conversion takes place
80           value = conversion.value
81         },
82         'put'={ # put exercise takes place
83           value = put.value
84         },
85         'call'={ # call exercise takes place
86           value = call.value
87         },
88         'forced_conversion'={ # forced conversion takes place
89           value = conversion.value
90         },
91         'redemption'={ # redemption takes place
92           value = redemption.value
93         },
94         'continuation'={ # none of other actions take place, investor holds convertible bond for one more period
95           value = continuation.value
96         }
97  )
98
99  action = exercise.legend[[action]]
100
101 ret = list(value=value, action=action)
102 invisible(ret)
103 }
104 # for performance, pre-compile CPU intensive function
105 payoff = cmpfun(payoff)
```

Listing A.3: payoff.r

In this file, payoff and exercise related functions are presented. In lines $46 - 105$ is presented

the payoff function. This is the core of the Convertible Bond's logic and would be the place to alter when implementing new and creative exercise features.

In lines $7-13$, exercise legend is presented. For performance, internally each exercise type is represented as a number. For convenience, though, this legend can be used to associate intelligible exercise name with its numeric code.

In lines $14-43$, the `exercise.time.bitmap` function is presented. This function was created solely for performance optimization reasons: by profiling payoff function, it was verified that a lot of time was spent verifying if the given date (time step) was a member of each exercise date set, i.e. if a given exercise type could take place at the given date. Due to this being implemented using the relatively slow $\%in\%$ operator and it being checked at each payoff function call (e.g., for each combination time step/Monte Carlo path), overall performance was very poor. To amend this, all membership checks of a given time step was done only once for each time step, at the very begining of backward induction iteration, yielding a consolidated data structure - the exercise bitmap (implemented as a list, for clearness) - containing flags that inform for which exercise type if, from an exercise dates' perspective, it could take place at the given time step.

A.2.4 SDE Simulation: sde.r

```
1  # Author: Leandro Loriato (llrt at impa.br)
2  #
3  # Auxiliary functions to simulate the path of the solution of a given SDE
4  #
5  # SDE.solve function based on sde.sim function from Stefano Iacus's Simulation and Inference for Stochastic
6  # Differential Equations book (section 2.4, page 69)
7
8
9  # auxiliary function that simulates the solution using Euler-Maruyama discretization method
10 SDE.solve.euler = function(t0, T, N.t, X0, drift_, sigma_,  Z){
11   t = seq(t0, T, length=(N.t+1)) # spans the time range
12   dt = (T-t0)/N.t # calculates time discretization step
13   sqrt.dt = sqrt(dt) # for saving computation, pre-computes sqrt(dt)
14
15   X = matrix(ncol=ncol(Z), nrow=(N.t+1)) # pre-allocates a matrix with N.t+1 rows for X
16   X[1,] = X0 # first X value is X0
17
18   for(i in 2:(N.t+1)){
19     drift__ = drift_(t[i-1], X[i-1,])
20     sigma__ = sigma_(t[i-1], X[i-1,])
21     X[i,] = X[i-1,] + drift__*dt + sigma__*sqrt.dt*Z[i-1,]
22   }
23
24   invisible(X)
25 }
26 # for performance, pre-compile CPU intensive function
27 SDE.solve.euler = cmpfun(SDE.solve.euler)
28
29 # auxiliary function that simulates the solution using Milstein discretization method
30 SDE.solve.milstein = function(t0, T, N.t, X0, drift_, sigma_, sigma.x_, Z){
31   t = seq(t0, T, length=(N.t+1)) # spans the time range
```

```
32  dt = (T−t0)/N.t # calculates time discretization step
33  sqrt.dt = sqrt(dt) # for saving computation, pre−computes sqrt(dt)
34
35  X = matrix(ncol=ncol(Z), nrow=(N.t+1)) # pre−allocates a matrix with N.t+1 rows for X
36  X[1,] = X0 # first X value is X0
37
38  for(i in 2:(N.t+1)){
39    drift__ = drift_(t[i−1], X[i−1,])
40    sigma__ = sigma_(t[i−1], X[i−1,])
41    sigma.x__ = sigma.x_(t[i−1], X[i−1,])
42
43    X[i,] = X[i−1,] + drift__*dt +
44      sigma__*sqrt.dt*Z[i−1,] +
45      (1/2)*sigma__*sigma.x__*(dt*Z[i−1,]^2 − dt)
46  }
47
48  invisible(X)
49 }
50 # for performance, pre−compile CPU intensive function
51 SDE.solve.milstein = cmpfun(SDE.solve.milstein)
52
53 # main function that, for an SDE of form dX(t) = drift(t, X) dt + sigma(t, X) dW(t), simulates its solution
54 SDE.solve = function(t0=0, T=1, X0=1, N.t=100, drift, sigma, method=c("euler", "milstein"), Z){
55
56  if(missing(method)){ # if not provided, use Euler−Maruyama method
57    method = "euler"
58  } else{ # if provided, use chosen method
59    method = match.arg(method)
60  }
61
62  # evaluates given drift, sigma and sigma.x expressions into functions
63  drift_ = function(t, x){eval(drift)} # drift function
64  sigma_ = function(t, x){eval(sigma)} # sigma function
65
66  if(method=="milstein"){ # if Milstein method is chosen, sigma.x should have been provided
67    sigma.x = D(sigma, "x") # derivative of sigma with respect to X
68    sigma.x_ = function(t, x){eval(sigma.x)} # sigma.x function
69  }
70
71  # generates a sample path from SDE solution's X with chosen method
72  if(method == "euler"){
73    X = SDE.solve.euler(t0, T, N.t, X0, drift_, sigma_,  Z)
74  } else if(method == "milstein"){
75    X = SDE.solve.milstein(t0, T, N.t, X0, drift_, sigma_, sigma.x_, Z)
76  }
77
78  invisible(X)
79 }
80 # for performance, pre−compile CPU intensive function
81 SDE.solve = cmpfun(SDE.solve)
```

Listing A.4: sde.r

In this file, SDE simulation functions are presented. These functions were based on the ones provided in Section 2.4, Page 69 of [Iac09]. The main function is `SDE.solve`, presented in lines $53 - 81$. It delegates discretization procedure to specific, lower level functions, which implement different discretization methods. Only Euler and Milstein discretizations are im-

plemented, but others may also be implemented by providing a specific function and altering `SDE.solve` function to use it and expose this option to caller.

A.2.5 Backward Induction Monte Carlo Methods: backward_induction.r

```
# Author: Leandro Loriato (llrt at impa.br)
#
# Auxiliary functions to calculate a convertible bond's price using Monte Carlo method with
# backward induction techniques: Least-Squared Monte Carlo (LSMC) e Hedged Monte Carlo (HMC)

library(orthopolynom)

# given relevant parameters, calculates the convertible bond price via Black formula
price.black = function(r, q, t, T, S, sigma, bond.params){
  bond = bond.value.T(bond.params, r)

  d1 = function(t, s, bond, sigma){1/(sigma * sqrt(T-t)) * (log((bond.params$conversion$ratio((T-t), s) * s)/
      bond) + ((r - q) + (sigma^2)/2) * (T-t))}
  d2 = function(t, s, bond, sigma){d1(t, s, bond, sigma) - sigma*sqrt(T-t)}

  price =  discounted.coupon(bond.params, t, r) +
    exp(-r*(T-t))*bond +
    bond.params$conversion$ratio((T-t), S) * S * exp(-q*(T-t)) * pnorm(d1(t, S, bond, sigma)) -
    bond * exp(-r*(T-t)) * pnorm(d2(t, S, bond, sigma))
  invisible(price)
}

# given relevant parameters, calculates the delta hedge from Black formula
delta.hedge.black = function(r, q, t, T, S, sigma, bond.params){
  bond = bond.value.T(bond.params, r)

  d1 = function(t, s, bond, sigma){1/(sigma * sqrt(T-t)) * (log((bond.params$conversion$ratio((T-t), s) * s)/
      bond) + ((r - q) + (sigma^2)/2) * (T-t))}

  price =  bond.params$conversion$ratio((T-t), S) * exp(-q*(T-t)) * pnorm(d1(t, S, bond, sigma))
  invisible(price)
}

# function that evaluates desired basis functions and its derivatives
basis = function(
  type=c("laguerre", "chebyshev_1", "chebyshev_2", "chebyshev_3", "hermite", "legendre",
       "w_laguerre", "w_chebyshev_1", "w_chebyshev_2", "w_chebyshev_3", "w_hermite", "w_legendre",
       "black"),
  M=4, ...){

  if(missing(type)){ # if not provided, use Laguerre Polynomials
    type = "laguerre"
  } else{ # if provided, use chosen type
    type = match.arg(type)
  }

  # auxiliary function that, given a recurrences list for desired polynom,
  # evaluates the polynom functions and its derivative functions
  polynom.fun = function(recurrences){
    polynom = orthogonal.polynomials(recurrences)
```

```
  fun = polynomial.functions(polynom)

  polynom.x = polynomial.derivatives(polynom)
  fun.x = polynomial.functions(polynom.x)

  invisible(list(fun=fun, fun.x=fun.x))
}

# auxiliary function that, given a recurrences list for desired polynom and its
# respective weight expression, evaluates the weighted polynom functions and its
# derivative functions
weighted.polynom.fun = function(recurrences, weight){
  weight.fun = function(x){eval(weight)}

  weight.x = D(weight, "x")
  weight.x.fun = function(x){eval(weight.x)}

  gen.fun = function(f){
    force(f)
    function(x){weight.fun(x)*f(x)}
  }

  gen.fun.x = function(f, f.x){
    force(f)
    force(f.x)
    function(x){weight.fun(x)*f.x(x) + weight.x.fun(x)*f(x)}
  }

  polynom = orthogonal.polynomials(recurrences)
  polynom.fun = polynomial.functions(polynom)
  fun = list()
  for(i in 1:length(polynom.fun)){
    fun[[i]] = gen.fun(f=polynom.fun[[i]])
  }

  polynom.x = polynomial.derivatives(polynom)
  polynom.x.fun = polynomial.functions(polynom.x)
  fun.x = list()
  for(i in 1:length(polynom.x.fun)){
    fun.x[[i]] = gen.fun.x(f=polynom.fun[[i]], f.x=polynom.x.fun[[i]])
  }

  invisible(list(fun=fun, fun.x=fun.x))
}

# for specified basis, evaluate its functions and derivative functions list
switch(type,
       "laguerre"={ # Laguerre Polynomials
         recurrences = laguerre.recurrences(n=(M-1), normalized=TRUE)

         ret.fun = polynom.fun(recurrences)
         fun = ret.fun$fun
         fun.x = ret.fun$fun.x
       },
       "w_laguerre"={ # Weighted Laguerre Polynomials
         recurrences = laguerre.recurrences(n=(M-1), normalized=TRUE)
         weight = expression(exp(-x))

```

```
    ret.weighted = weighted.polynom.fun(recurrences, weight)
    fun = ret.weighted$fun
    fun.x = ret.weighted$fun.x
  },
  "chebyshev_1"={ # 1st Kind Chebyshev Polynomials
    recurrences = chebyshev.t.recurrences(n=(M-1), normalized=TRUE)

    ret.fun = polynom.fun(recurrences)
    fun = ret.fun$fun
    fun.x = ret.fun$fun.x
  },
  "w_chebyshev_1"={ # Weighted 1st Kind Chebyshev Polynomials
    recurrences = chebyshev.t.recurrences(n=(M-1), normalized=TRUE)
    weight = expression(1/sqrt(1-(x^2)/4))

    ret.weighted = weighted.polynom.fun(recurrences, weight)
    fun = ret.weighted$fun
    fun.x = ret.weighted$fun.x
  },
  "chebyshev_2"={ # 2nd Kind Chebyshev Polynomials
    recurrences = chebyshev.s.recurrences(n=(M-1), normalized=TRUE)

    ret.fun = polynom.fun(recurrences)
    fun = ret.fun$fun
    fun.x = ret.fun$fun.x
  },
  "w_chebyshev_2"={ # Weighted 2nd Kind Chebyshev Polynomials
    recurrences = chebyshev.s.recurrences(n=(M-1), normalized=TRUE)
    weight = expression(1/sqrt(1-(x^2)/4))

    ret.weighted = weighted.polynom.fun(recurrences, weight)
    fun = ret.weighted$fun
    fun.x = ret.weighted$fun.x
  },
  "chebyshev_3"={ # 3rd Kind Chebyshev Polynomials
    recurrences = chebyshev.c.recurrences(n=(M-1), normalized=TRUE)

    ret.fun = polynom.fun(recurrences)
    fun = ret.fun$fun
    fun.x = ret.fun$fun.x
  },
  "w_chebyshev_3"={ # Weighted 3rd Kind Chebyshev Polynomials
    recurrences = chebyshev.c.recurrences(n=(M-1), normalized=TRUE)
    weight = expression(1/sqrt(1-x^2))

    ret.weighted = weighted.polynom.fun(recurrences, weight)
    fun = ret.weighted$fun
    fun.x = ret.weighted$fun.x
  },
  "hermite"={ # Hermite Polynomials
    recurrences = hermite.h.recurrences(n=(M-1), normalized=TRUE)

    ret.fun = polynom.fun(recurrences)
    fun = ret.fun$fun
    fun.x = ret.fun$fun.x
  },
  "w_hermite"={ # Weighted Hermite Polynomials
    recurrences = hermite.h.recurrences(n=(M-1), normalized=TRUE)
```

```
        weight = expression(exp(-x^2))

        ret.weighted = weighted.polynom.fun(recurrences, weight)
        fun = ret.weighted$fun
        fun.x = ret.weighted$fun.x
      },
      "legendre"={ # Legendre Polynomials
        recurrences = legendre.recurrences(n=(M-1), normalized=TRUE)

        ret.fun = polynom.fun(recurrences)
        fun = ret.fun$fun
        fun.x = ret.fun$fun.x
      },
      "w_legendre"={ # Weighted Legendre Polynomials
        recurrences = legendre.recurrences(n=(M-1), normalized=TRUE)
        weight = expression(1)

        ret.weighted = weighted.polynom.fun(recurrences, weight)
        fun = ret.weighted$fun
        fun.x = ret.weighted$fun.x
      },
      "black"={ # Black Basis
        bond.T = bond.value.T(bond.params, r)

        fun = list()
        fun[[1]] = function(x){1 + x * 0}
        fun[[2]] = function(x){x - exp(-r*T)*bond.T/bond.params$conversion$ratio(T, x)}
        fun[[3]] = function(x){price.black(x, r=r, q=q, t=0, T=T, sigma=.sigma, bond.params=bond.params)}

        fun.x = list()
        fun.x[[1]] = function(x){0 + x * 0}
        fun.x[[2]] = function(x){1 + x * 0}
        fun.x[[3]] = function(x){delta.hedge.black(x, r=r, q=q, t=0, T=T, sigma=.sigma, bond.params)}
      }
  )

  ret = list(type=type, M=M, fun=fun, fun.x=fun.x)
  invisible(ret)
}


# generate a function that, given an evaluted basis function, calculates its value given an input vector X
funval = function(X){
  invisible(function(fun.eval){fun.eval(X)})
}


# sanitize a exercise matrix, leaving for each path only the entry that effected the
# exercise
clean.exercise = function(exercise, N.t){
  cleaned.exercise = matrix(exercise, nrow=nrow(exercise), ncol=ncol(exercise))
  stopping.time = rep(1, ncol(exercise))

  for(i in 1:ncol(exercise)){
    for(j in 1:N.t){
      if(cleaned.exercise[[j,i]] > 0){ # some exercise action was taken at current time
        stopping.time[[i]] = j
        cleaned.exercise[(j+1):(N.t+1),i] = 0 # as instrument already suffered an exercise action,
```

```
223         # every other future exercise action does not take place
224         break
225       }
226     }
227   }
228
229   invisible(list(cleaned.exercise=cleaned.exercise, stopping.time=stopping.time))
230 }
231 # for performance, pre-compile CPU intensive function
232 clean.exercise = cmpfun(clean.exercise)
233
234
235 # utilitary function for packing results from a Monte Carlo simulation into an appropriate object
236 as.montecarlo = function(C, S, S0, method, alpha=0.05){
237   .S = S[1,] # initial stock price values
238   indexes = which(.S==S0) # which column indexes correspond to stock price starting at S0 at time t=t0
239   filtered.C = C[,indexes] # matrix of paths of price starting at S0 at time t=t0
240
241   value = mean(filtered.C[1,]) # fair price at time t=t0
242   sigma = sd(filtered.C[1,]) # standard deviation of price at time t=t0
243
244   M = length(filtered.C[1,])
245   Z.alpha = qnorm(alpha/2, lower.tail=F)
246   confidence.interval = c(value-Z.alpha*sigma/sqrt(M), value+Z.alpha*sigma/sqrt(M))
247
248   MC = (list(C=filtered.C, method=method, value=value, sigma=sigma, alpha=alpha, confidence.interval=
        confidence.interval))
249   class(MC) = "montecarlo"
250   return(MC)
251 }
252 # for performance, pre-compile CPU intensive function
253 as.montecarlo = cmpfun(as.montecarlo)
254
255
256 # utilitary functions for summarizing relevant data from a montecarlo object
257 summary.montecarlo = function(MC){
258   s = list(value=MC$value, sigma=MC$sigma, confidence.interval=MC$confidence.interval)
259   class(s) = "summary.montecarlo"
260   return(s)
261 }
262 setGeneric("summary.montecarlo")
263
264 print.summary.montecarlo = function(s){
265   print(paste("value: ", s$value))
266   print(paste("sigma: ", s$sigma))
267   print("confidence interval: ")
268   print(s$confidence.interval)
269 }
270 setGeneric("print.summary.montecarlo")
271
272
273
274 # auxiliary function that evaluates regression args for LSMC method
275 regression.args.LSMC = function(S, C, j, rho, Ca.val){
276   # evaluate regression args for LSMC method
277   y = rho * C[j+1,]
278   x = Ca.val
279
```

```
  ret = list(x=x, y=y)

  invisible(ret)
}
# for performance, pre-compile CPU intensive function
regression.args.LSMC = cmpfun(regression.args.LSMC)


# auxiliary function that evaluates regression args for HMC method
regression.args.HMC = function(S, C, j, rho, Ca.val, Fa.val){
  # evaluate regression args for HMC method
  y = rho * C[j+1,]
  x = Ca.val + Fa.val * (rho*S[j+1,] - S[j,])

  ret = list(x=x, y=y)

  invisible(ret)
}
# for performance, pre-compile CPU intensive function
regression.args.HMC = cmpfun(regression.args.HMC)

# auxiliary function that solves regression using regular method,
# then calculating projected expected value according to regression
regression.regular = function(x, y, Ca.val){
  # in regular method, least squares fit is solved using QR on
  # whole sample
  alpha = solve(qr(x, LAPACK=TRUE), y)

  # calculate expected value
  exp.val = tcrossprod(Ca.val, t(alpha)) # Ca.val %*% alpha
  invisible(exp.val)
}
# for performance, pre-compile CPU intensive function
regression.regular = cmpfun(regression.regular)

# auxiliary function that solves regression using Bouchard's method,
# then calculating projected expected value according to regression
regression.bouchard = function(x, y, S, regression.intervals, Ca.val){
  # in Bouchard's method, least squares fit is solved using several QR,
  # each on a partition of the sample

  exp.val = vector(mode="numeric", length=length(y))

  .df.regression = data.frame(S=S, indexes=1:length(y))
  .df.regression = .df.regression[order(.df.regression$S),]

  interval.length = length(y)/regression.intervals

  for(k in 1:regression.intervals){ # for each regression interval
    # evaluate indexes corresponding to current interval
    interval.indexes = ((k-1)*interval.length+1):(min(k*interval.length, interval.length*regression.intervals))
    # evaluates corresponding original indexes
    original.indexes = .df.regression$indexes[interval.indexes]

    # extract partitioned x and y based on interval indexes
    .y = y[original.indexes]
    .x = x[original.indexes,]

```

```
    # solve regression for partitioned x and y
    alpha = solve(qr(.x, LAPACK=TRUE), .y)

    # calculate expected value
    exp.val[original.indexes] = tcrossprod(Ca.val[original.indexes,], t(alpha)) # Ca.val %*% alpha
  }

  invisible(exp.val)
}
# for performance, pre-compile CPU intensive function
regression.bouchard = cmpfun(regression.bouchard)


# main function that, given relevant parameters, calculates the american contingent claim price via Monte Carlo
price.MC = function(S, S0, r, q, t0, T, N.t, N.MC,
                    backward.induction.method=c("LSMC", "HMC"), price.fun, bond.params,
                    regression.method=c("regular", "bouchard"), regression.intervals){

  # sets backward induction method
  if(missing(backward.induction.method)){ # if not provided, use LSMC method
    backward.induction.method = "LSMC"
  } else{ # if provided, use chosen method
    backward.induction.method = match.arg(backward.induction.method)
  }

  # sets regression method
  if(missing(regression.method)){ # if not provided, use regular method
    regression.method = "regular"
  } else{ # if provided, use chosen method
    regression.method = match.arg(regression.method)

    # if Bouchard's regression method is chosen and number of regression intervals is not informed,
    # sets it in such a way that each interval contains 200 values
    if(regression.method=="bouchard" && missing(regression.intervals)){
      regression.intervals = N.MC/200
    }
  }

  dt = (T-t0)/N.t # calculates time discretization step
  t = seq(t0, T, by=dt) # spans the time range

  # pre-allocates a N.t+1 x N.MC matrix for C (contingent claim's price at each time) and exercise
  C = matrix(numeric(0), nrow=(N.t+1), ncol=N.MC)
  exercise = matrix(numeric(0), nrow=(N.t+1), ncol=N.MC)

  # bond value at time T
  bond.T = bond.value.T(bond.params, r)

  exercise.time.bitmap.ret = exercise.time.bitmap(T, bond.params) # computes exercise time restriction bitmap

  for(i in 1:N.MC){ # for each path, updates C and exercise values at time with index j
    S.vector = S[1:(N.t+1), i]
    payoff.ret = payoff(T, S.vector, r, bond.T, exercise.time.bitmap.ret, bond.params) # intrinsic value info at time T
    C[[N.t+1,i]] = payoff.ret$value # initializes C values at time T, as payoff at time T is known
    exercise[[1+N.t,i]] = payoff.ret$action # evaluates exercise decision
```

```
393  }
394
395  # sets basis functions
396  Ca.fun = price.fun$fun # basis functions for contingent claim's price
397  if(backward.induction.method=="HMC"){
398    Fa.fun = price.fun$fun.x # basis functions for hedge function for contingent claim's price
399  }
400
401  rho = exp(-r*dt) # pre-computing exp(-r*dt) to save computation
402
403  # apply the chosen backward induction algorithm
404  for(j in N.t:1){
405    cat("*") # print "*" character to give sense of computation progress
406
407    exercise.time.bitmap.ret = exercise.time.bitmap(t[[j]], bond.params) # computes exercise time restriction
         bitmap
408
409    for(i in 1:N.MC){ # for each path, updates C and exercise values at time with index j
410      S.vector = S[1:j, i]
411      payoff.ret = payoff(t[[j]], S.vector, r, rho*C[[j+1,i]], exercise.time.bitmap.ret, bond.params)
412      C[[j,i]] = payoff.ret$value # initialize C value with intrinsic value
413      exercise[[j,i]] = payoff.ret$action # evaluates exercise decision
414    }
415
416    # evaluate regression args according to chosen backward induction method
417    if(backward.induction.method=="LSMC"){
418      # apply Ca functions over S values at time with index j
419      Ca.val = sapply(Ca.fun, funval(S[j,]))
420      # evaluate regression args
421      ret.regression.args = regression.args.LSMC(S, C, j, rho, Ca.val)
422    } else if(backward.induction.method=="HMC"){
423      # apply Ca and Fa functions over S values at time with index j
424      Ca.val = sapply(Ca.fun, funval(S[j,]))
425      Fa.val = sapply(Fa.fun, funval(S[j,]))
426      # evaluate regression args
427      ret.regression.args = regression.args.HMC(S, C, j, rho, Ca.val, Fa.val)
428    }
429    y = ret.regression.args$y
430    x = ret.regression.args$x
431
432    # solve least-squares fit using chosen regression method and
433    # calculate projected expected value at time with index j
434    if(regression.method=="regular"){ # regular regression method
435      exp.val = regression.regular(x, y, Ca.val)
436    } else if (regression.method=="bouchard"){ # Bouchard's regression method
437      exp.val = regression.bouchard(x, y, S[j,], regression.intervals, Ca.val)
438    }
439
440
441    # for each component over time with index j:
442    # -> C if the value of C is greater than or equal the expected value calculated
443    # -> expected value if the value of C is less than the expected value calculated
444    for(i in 1:N.MC){
445      if(C[[j,i]] >= exp.val[[i]]){ # exercise takes place
446        #C[[j,i]] = C[[j,i]] # update value to current intrinsic value
447      } else{ # return next time's discounted value
448        C[[j,i]] = rho*C[[j+1,i]] # update value to discounted value of time j+1
449        exercise[[j,i]] = exercise.legend[["continuation"]] # update current exercise action to continuation
```

```
450       }
451
452       # if current time is a coupon payment date, also add coupon to current value
453       if(t[[j]] %in% bond.params$coupon$dates){
454         C[[j,i]] = C[[j,i]] + bond.params$coupon$rate * bond.params$principal
455       }
456     }
457   }
458
459   # clean exercise matrix to retain only the first exercise action at each path, calculating its stopping time
460   ret.clean = clean.exercise(exercise, N.t)
461   stopping.time = ret.clean$stopping.time
462   exercise = ret.clean$cleaned.exercise
463
464   ret = list(MC=as.montecarlo(C, S, S0, method=backward.induction.method, alpha=0.05), exercise=exercise,
          stopping.time=stopping.time)
465
466   invisible(ret)
467 }
468 price.MC = cmpfun(price.MC)
```

Listing A.5: backward_induction.r

In this file, backward induction related functions are presented. In lines $8 - 30$ a function implemented Black closed form formula for European Convertible Bond is presented, as well as its delta-hedge related formula.

Next, in lines $32 - 203$ is presented a function for calculating a list of basis functions given a basis type and its dimension (`M`). Weighted and non-weighted Laguerre, 1st, 2nd and 3rd order Chebyshev, Hermite and Legendre orthogonal polynomial basis are implemented. To allow for use of arbitrary basis' dimension, the `orthopolynom` library is used (imported in line 6). By doing so, we also benefit from the fact that, for generating polynomial functions, the `orthopolynom` library makes use of Horner's method, a more numerically stable method for calculating the value of a polynomial at a given point.

Weighted versions are not explicitly generated by `orthopolynom` library (although it does provides each basis' weight), so they are generated by hand, iterating through the non-weighted counterparts' list and generating functions that multiply the weight with the non-weighted polynomial. These functions were not used along the examples in this work, due to they yielding poorer results, but were kept as possible options, should they be used in other contexts.

Although not explicitly discussed along this work (mainly due to it providing poorer results, like the weighted versions), a Black basis function is also available for use. This is a 3-dimensional basis of the form:

$$\begin{cases} \phi_1(x) & = 1 \\ \phi_2(x) & = x - K \\ \phi_3(x) & = BS(x) \end{cases}$$

where $BS(x)$ is the Black closed-form formula for the european equivalent of the instrument and K a strike price, in the sense of a call strike price. In this work, $BS(x)$ is given by Equation 4.2.2 and implemented in the function in the beginning of the file; K is implemented as $exp(-rT)B(T)/n(T)$, where $B(T)$ and $n(T)$ are, respectively, the straight bond value and conversion ratio at time $t = T$.

To abstract away which method was used for creating basis functions - using `orthopolynom` or crafting by hand - a uniform data structure is created and return. For use with HMC algorithm, for each basis function its derivative counterpart is also evaluated and returned. Finally, basis function computation was implemented in an extensible way so as to make easy to add other types of basis functions', such as Fourier ones.

In lines $206 - 209$ is presented a convenience function that, given a vector of points, return another function which takes a function as argument and returns a vector of values yielded by application of argument function to the given vector of points. This construct may seem cryptic at first, but is very useful for applying the basis' function over a vector of values using R's `sapply` vectorized function.

In lines $212 - 253$ is presented a utilitary function for cleaning exercise data yielded by pricing algorithms from exercise actions that did not take place at a given time step because other exercise action took place earlier. Although not used in Setting 4, this function is useful for sanitizing exercise data and prepare them for evaluating attained exercise boundary.

In lines $235 - 270$ are utilitary functions for building from returned regression data a Monte Carlo data model, so as to make it more easy to inspect Monte Carlo methods results.

Finally, in lines $274 - 468$ backward induction pricing methods themselves are implemented. Main pricing procedure is implemented via function `price.MC` in lines $351 - 468$. All common backward induction steps are factored into these functions with specific regression procedures delegated to more low level functions. Both LSMC and HMC algorithms are implemented. Since they mainly differ in which regression arguments are effectively used, this logic is factored into 2 functions, `regression.args.LSMC` and `regression.args.HMC`. Regression itself is carried on by other 2 functions, one implementing regular, monolithic regression procedure (`regression.regular`) and other implementing Bouchard's regression procedure (`regression.bouchard`). For Bouchard's regression procedure, care was taken to allow an arbitrary number of regression intervals. Care should be taken, however, to ensure each regression interval contains enough information for regression procedure to converge.

A.3 Final Remark

To make provided implementation more ready-to-use, wherever possible we tried to factor away common functionality and parameters, making it more concise and parametrizable. However, since Convertible Bonds are very heterogenous and may present several creative and path-dependent features (see Chapters 2 and 4), to correctly price them, one would

probably need to adapt the given code. In this sense, we preferred not to overparametrize the code, to maintain it highly flexible and extensible, easy to modify. We expect the user to be able to clearly understand the premises behind the code, what every part does and how they interact, and to bend the provided implementation to satisfy its own needs, maybe even using it as a base framework for pricing other types of financial instruments.

Bibliography

[AFV03] Elie Ayache, Peter A. Forsyth, and Kenneth R. Vetzal. Valuation of convertible bonds with credit risk. *The Journal of Derivatives*, 11(1):9–29, 2003.

[AG12] Credit Suisse AG. Convertible bonds: Fundamentals, asset allocation, solvency. Technical report, 2012. https://www.credit-suisse.com/ch/fixed_income/doc/white_paper_en.pdf accessed on 11/09/2013.

[AKW01] Manuel Ammann, Axel H. Kind, and Christian Wilde. The pricing of convertible bonds. Technical report, Discussion Paper, University of St. Gallen, 2001.

[AKW03] Manuel Ammann, Axel Kind, and Christian Wilde. Are convertible bonds underpriced? an analysis of the french market. *Journal of Banking & Finance*, 27(4):635–653, 2003.

[AKW08] Manuel Ammann, Axel Kind, and Christian Wilde. Simulation-based pricing of convertible bonds. *Journal of Empirical Finance*, 15(2):310–331, 2008.

[BHM13] Carole Bernard, Mary Hardy, and Anne MacKay. State-dependent fees for variable annuity guarantees. *Available at SSRN 2258199*, 2013.

[BK02] Wolfgang Bühler and Christian Koziol. Valuation of convertible bonds with sequential conversion. *Schmalenbach Business Review (sbr)*, 54(4), 2002.

[BN04] Ana Bermúdez and Maria R. Nogueiras. Numerical solution of two-factor models for valuation of financial derivatives. *Mathematical Models and Methods in Applied Sciences*, 14(02):295–327, 2004.

[BNV06] Alfredo Bermúdez, Maria R. Nogueiras, and Carlos Vázquez. Numerical solution of variational inequalities for pricing asian options by higher order lagrange–galerkin methods. *Applied Numerical Mathematics*, 56(10):1256–1270, 2006.

[BP04] Jean-Philippe Bouchaud and Marc Potters. *Theory of Financial Risks: From Statistical Physics to Risk Management*. Cambridge University Press, 2 edition, 2004.

[BS73] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The journal of political economy*, pages 637–654, 1973.

[BS77] Michael J. Brennan and Eduardo S. Schwartz. Convertible bonds: valuation and optimal strategies for call and conversion. *Journal of Finance*, 32(5):1699–1715, 1977.

[BS80] Michael J. Brennan and Eduardo S. Schwartz. Analyzing convertible bonds. *Journal of Financial and Quantitative Analysis*, 15(04):907–929, 1980.

[BW03] Ana Bermudez and Nick Webber. An asset based model of defaultable convertible bonds with endogenised recovery. Technical report, Working Paper, City University, Cass Business School, London, 2003.

[CJR85] John C. Cox, Jonathan E. Ingersoll Jr, and Stephen A. Ross. A theory of the term structure of interest rates. *Econometrica: Journal of the Econometric Society*, pages 385–407, 1985.

[CLP02] Emmanuelle Clément, Damien Lamberton, and Philip Protter. An analysis of a least squares regression method for american option pricing. *Finance and Stochastics*, 6(4):449–471, 2002.

[Com11] McKinsey & Company. Mapping global capital markets 2011. Technical report, 2011. http://www.mckinsey.com/insights/global_capital_markets/mapping_global_capital_markets_2011 accessed on 11/09/2013.

[Cot13] Richard Cotton. *Learning R*. O'Reilly, 2013.

[CRR79] John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of financial Economics*, 7(3):229–263, 1979.

[Exc] Microsoft excel. http://office.microsoft.com/pt-br/excel/. Accessed: 2013-12-02.

[Gar03] Diego García. Convergence and biases of monte carlo estimates of american option prices using a parametric exercise rule. *Journal of Economic Dynamics and Control*, 27(10):1855–1879, 2003.

[Gla04] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer, 2004.

[GY04] Paul Glasserman and Bin Yu. Number of paths versus number of basis functions in american option pricing. *The Annals of Applied Probability*, 14(4):2090–2119, 2004.

[Hes93] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of financial studies*, 6(2):327–343, 1993.

[HP96] Thomas S. Y. Ho and David M. Pfeffer. Convertible bonds: model, value attribution, and analytics. *Financial Analysts Journal*, pages 35–44, 1996.

[Hul09] John Hull. *Options, Futures and Other Derivatives*. Prentice Hall finance series.

Pearson/Prentice Hall, 2009.

[Iac09] Stefano Maria Iacus. *Simulation and Inference for Stochastic Differential Equations*. Springer, 2009.

[IFR] International financial reporting standards. http://www.ifrs.org/. Accessed: 2013-11-09.

[IJ77] Jonathan E Ingersoll Jr. A contingent claim valuation of convertible securities. *Journal of Financial Economics*, 4(3):289–321, 1977.

[Jia09] Quiyi Jia. *Pricing American Options using Monte Carlo Methods*. PhD thesis, Department of Mathematics, Uppsala University, 2009.

[KK01] Ralph Korn and Elke Korn. *Option Pricing and Portfolio Optimization: Modern Methods of Financial Mathematics*. Crm Proceedings & Lecture Notes. American Mathematical Society, 2001.

[KP11] Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Stochastic Modelling and Applied Probability. Springer, 2011.

[Lew91] Craig M. Lewis. Convertible debt: Valuation and conversion in complex capital structures. *Journal of Banking & Finance*, 15(3):665–682, 1991.

[LS01] Francis A. Longstaff and Eduardo S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *Review of Financial Studies*, pages 113–147, 2001.

[MAT] MATLAB - the language of technical computing. http://www.mathworks.com/products/matlab/. Accessed: 2013-12-02.

[Mer73] Robert C. Merton. Theory of rational option pricing. *The Bell Journal of Economics and Management Science*, pages 141–183, 1973.

[MK12] Krasimir Milanov and Ognyan Kounchev. Binomial tree model for convertible bond pricing within equity to credit risk framework. *arXiv preprint arXiv:1206.1400*, 2012.

[MR06] Marek Musiela and Marek Rutkowski. *Martingale Methods in Financial Modelling*. Stochastic Modelling and Applied Probability. Springer, 2006.

[MS86] John McConnell and Eduardo S. Schwartz. Lyon taming. *Journal of Finance*, 41(3):561–576, 1986.

[PBS01] Marc Potters, Jean-Philippe Bouchaud, and Dragan Sestovic. Hedged Monte-Carlo: Low variance derivative pricing with objective probabilities. *Physica A: Statistical Mechanics and its Applications*, 289(3-4):517–525, 2001.

[R C13] R Core Team. R: A language and environment for statistical computing. http://www.

R-project.org/, 2013. Accessed: 2013-12-02.

[Sac94] Goldman Sachs. Valuing convertible bonds as derivatives. *Quantitative strategies research notes*, 11:1–30, 1994.

[Shr04] Steven Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models*. Springer Finance. Springer, 2004.

[Tee11] Paul Teetor. *R Cookbook*. O'Reilly, 2011.

[TF98] Kostas Tsiveriotis and Chris Fernandes. Valuing convertible bonds with credit risk. *The Journal of Fixed Income*, 8(2):95–102, 1998.

[TKN01] Akihiko Takahashi, Takao Kobayashi, and Naruhisa Nakagawa. Pricing convertible bonds with default risk: a duffie-singleton approach. 2001.

[WB12] Xavier Warin and Bruno Bouchard. Monte-carlo valuation of american options: facts and new algorithms to improve existing methods. Technical report, Paris Dauphine University, 2012.

[WK05] Christian Wilde and Axel H. Kind. Pricing convertible bonds with monte carlo simulation. *Available at SSRN 676507*, 2005.

[Zad10] Ariel Zadikov. *Methods of Pricing Convertible Bonds*. PhD thesis, University of Cape Town, 2010.