Numerical methods for option pricing: polynomial approximation and high dimensionality

Thèse N° 7386

Présentée le 1er novembre 2019

à la Faculté des sciences de base
Algorithmes numériques et calcul haute performance - Chaire CADMOS
Programme doctoral en mathématiques

pour l'obtention du grade de Docteur ès Sciences

par

Francesco STATTI

Acceptée sur proposition du jury

Prof. T. Mountford, président du jury
Prof. D. Kressner, Prof. D. Filipovic, directeurs de thèse
Prof. Y. Nakatsukasa, rapporteur
Prof. J. Kallsen, rapporteur
Prof. S. Deparis, rapporteur

2019

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Acknowledgements

The PhD has been a long and challenging journey and it would not have been possible to successfully reach the end of it without the encouragement of many wonderful people I have met in this period. I am grateful for the support I have received in these four years.

First of all, I would like to thank my thesis advisors, Prof. Damir Filipović and Prof. Daniel Kressner, for giving me the opportunity to jointly work in their research groups, and for guiding me through my PhD studies with helpful suggestions, constructive comments, and inspirational ideas.

Furthermore, I would like to thank the members of my PhD committee, Prof. S. Deparis, Prof. J. Kallsen, and Prof. Y. Nakatsukasa for their work and their suggestions. Also, I thank Prof. T. Mountford for agreeing to preside my PhD defense.

Part of my research conducted in these years has been supported by my co-authors Kathrin, Yuji, and Robert. It was a great pleasure for me to work together and to learn from them.

ANCHP has been like a family for me and I would like to express my gratitude to its wonderful members and former members I have met: Alice, Axel, Christoph, Jonas, Kathryn, Lana, Michael P., Michael S., Petar, Robert, Thalia, and Zvonimir. An extremely special mention goes to Ana and Stefano, who have helped me a lot and have motivated me during the hardest times throughout the years. They have been like a sister and a brother to me. Also, a big thank you goes to my SFI colleagues Damien, Erik, Lotfi, Sander, and Sebastian.

Besides ANCHP, I am very grateful to all the members of the Mathicse group who have made my time at EPFL enjoyable and less stressful. I will never forget the long coffee breaks and the infinitely many pizzas we ate at Luigia.

Even if I spent a lot of time on the EPFL campus, I have also had the chance to enjoy the company of many friends and of my flatmates. Therefore, I would like to thank them all for enriching my time in Lausanne and the weekends in Ticino with very nice non-work related moments.

I am deeply thankful to Lucia whose love, support, and patience have completely revolutionized my life. Ever since I met her, she has made every single day of my life more colorful and joyful. Thank you, Lucy.

Finally, I would like express my deep gratitude to my parents for supporting the choices I have made in my life, for their constant unconditional love, and for their continuous encouragement. Thanks to them, I have never felt alone.

Lausanne, September 2019 Francesco S.

Abstract

Options are some of the most traded financial instruments and computing their price is a central task in financial mathematics and in practice. Consequently, the development of numerical algorithms for pricing options is an active field of research. In general, evaluating the price of a specific option relies on the properties of the stochastic model used for the underlying asset price. In this thesis we develop efficient and accurate numerical methods for option pricing in a specific class of models: polynomial models. They are a versatile tool for financial modeling and have useful properties that can be exploited for option pricing.

Significant challenges arise when developing option pricing techniques. For instance, the underlying model might have a high-dimensional parameter space. Furthermore, treating multi-asset options yields high-dimensional pricing problems. Therefore, the pricing method should be able to handle high dimensionality. Another important aspect is the efficiency of the algorithm: in real-world applications, option prices need to be delivered within short periods of time, making the algorithmic complexity a potential bottleneck. In this thesis, we address these challenges by developing option pricing techniques that are able to handle low and high-dimensional problems, and we propose complexity reduction techniques.

The thesis consists of four parts:

First, we present a methodology for European and American option pricing. The method uses the moments of the underlying price process to produce monotone sequences of lower and upper bounds of the option price. The bounds are obtained by solving a sequence of polynomial optimization problems. As the order of the moments increases, the bounds become sharper and eventually converge to the exact price under appropriate assumptions.

Second, we develop a fast algorithm for the incremental computation of nested block triangular matrix exponentials. This algorithm allows for an efficient incremental computation of the moment sequence of polynomial jump-diffusions. In other words, moments of order $0, 1, 2, 3 \ldots$ are computed sequentially until a dynamically evaluated criterion tells us to stop. The algorithm is based on the scaling and squaring technique and reduces the complexity of the pricing algorithms that require such an incremental moment computation.

Third, we develop a complexity reduction technique for high-dimensional option pricing. To this end, we first consider the option price as a function of model and payoff parameters. Then, the tensorized Chebyshev interpolation is used on the parameter space to increase the efficiency in computing option prices, while maintaining the required accuracy. The high dimensionality of the problem is treated by expressing the tensorized interpolation in the tensor train format and by deriving an efficient way, which is based on tensor completion, to approximate the interpolation coefficients.

Lastly, we propose a methodology for pricing single and multi-asset European options. The approach is a combination of Monte Carlo simulation and function approximation. We address the memory limitations that arise when treating very high-dimensional applications by combining the method with optimal sampling strategies and using a randomized algorithm to reduce the storage complexity of the approach.

The obtained numerical results show the effectiveness of the algorithms developed in this thesis.

Keywords. Option pricing, polynomial models, matrix exponential, polynomial bounds, scaling and squaring method, complexity reduction, tensorized Chebyshev interpolation, high-dimensional problems, low-rank tensor approximation, Monte Carlo methods.

Estratto

Le opzioni sono uno dei principali strumenti finanziari e calcolarne il valore è un problema importante in matematica finanziaria e nella pratica. Lo sviluppo di algoritmi numerici per la prezzatura delle opzioni (option pricing) è perciò un campo di ricerca attivo. In generale, un metodo per calcolare il prezzo di un'opzione dipende dalle proprietà del modello stocastico usato per il prezzo del sottostante. In questa tesi sviluppiamo metodi numerici efficienti e accurati per un determinato tipo di modelli: i modelli polinomiali. Essi sono uno strumento versatile per la modellizzazione finanziaria e offrono proprietà interessanti che si possono usare per prezzare le opzioni.

Quando si sviluppano tecniche di option pricing si presentano delle sfide sostanziali. Per esempio, il modello scelto potrebbe avere uno spazio parametrico di dimensione alta. Inoltre, problemi multidimensionali si presentano nel prezzare opzioni basate su più sottostanti, ossia opzioni multi-asset. Perciò, il metodo sviluppato deve essere in grado di gestire l'alta dimensionalità. Un altro aspetto importante è l'efficienza dell'algoritmo: in applicazioni reali i prezzi devono essere calcolati in un breve lasso di tempo, per questo motivo gli algoritmi devono essere il più efficienti possibile. In questa tesi trattiamo questi aspetti sviluppando tecniche di option pricing per opzioni single e multi-asset. Inoltre, proponiamo delle tecniche per ridurre la complessità algoritmica.

La tesi è composta da quattro parti.

Nella prima parte proponiamo un metodo per prezzare opzioni Europee e Americane. L'approccio utilizza i momenti del sottostante per calcolare, attraverso dei problemi di ottimizzazione polinomiale, sequenze monotone di limiti superiori e inferiori del prezzo. Sotto ipotesi appropriate, aumentando l'ordine dei momenti usati i limiti convergono monotonamente verso il prezzo esatto.

Nella seconda parte sviluppiamo un algoritmo efficiente per calcolare sequenzialmente esponenziali di matrici triangolari a blocchi. Questo ci permette di calcolare efficientemente la sequenza dei momenti di diffusioni polinomiali con salti. In altre parole, i momenti di ordine 0,1,2,3... vengono calcolati uno dopo l'altro finché necessario. L'algoritmo è basato sul metodo di scalatura e quadratura e riduce la complessità delle tecniche di option pricing che necessitano di questo tipo di calcolo incrementale dei momenti.

Nella terza parte sviluppiamo una tecnica per ridurre la complessità degli algoritmi di option pricing in dimensione alta. A tal fine, consideriamo il prezzo di un'opzione come funzione dei parametri del modello e del payoff. In seguito applichiamo l'interpolazione multivariata di Chebyshev sullo spazio dei parametri per aumentare l'efficienza di calcolo dei prezzi, mantenendo l'accuratezza necessaria. L'alta dimensionalità del problema viene trattata esprimendo l'interpolazione in formato tensor train e derivando una maniera efficace, basata sul completamento di tensori, per approssimare i coefficienti dell'interpolazione.

Infine, proponiamo un metodo per prezzare opzioni single e multi-asset che combina la simulazione Monte Carlo con l'approssimazione di funzioni. Inoltre, per problemi di dimensione molto alta riduciamo la memoria necessaria per eseguire il metodo combinandolo con una strategia di campionamento ottimale e usando un algoritmo randomizzato.

I risultati numerici ottenuti dimostrano l'efficacia degli algoritmi sviluppati in questa tesi.

Parole Chiave. Prezzatura delle opzioni, modelli polinomiali, matrice esponenziale, limiti polinomiali, metodo di scalatura e quadratura, riduzione della complessità, interpolazione multivariata di Chebyshev, problemi multidimensionali, approssimazione di rango basso di tensori, metodi di Monte Carlo.

Contents

1 Introduction and background

This thesis deals with the numerical computation of option prices. In particular, we develop efficient and accurate numerical algorithms for low and high-dimensional option pricing problems for a specific class of models: polynomial models. In this framework, the underlying asset prices are modeled via polynomial (jump-)diffusions which have interesting properties that can be exploited to design algorithms for option pricing. The main property is that conditional moments are given in closed form. The proposed numerical approaches are based on different techniques: optimization, numerical approximation of matrix exponentials, tensorized Chebyshev interpolation, and Monte Carlo simulation. Despite their different nature, most of the approaches have a common underlying idea: they are based on polynomial approximation.

The thesis is organized as follows. In Chapter 2 we introduce the class of polynomial models. Therein, we provide a formal definition of polynomial (jump-)diffusions, and we summarize their main properties. In Chapter 3 we introduce a pricing technique based on the computation of polynomial bounds for option prices. An efficient algorithm for the incremental computation of the conditional moments of polynomial (jump-)diffusions is developed in Chapter 4. Chapter 5 and Chapter 6 focus on numerical methods for high-dimensional option pricing problems. In particular, in Chapter 5 we present a complexity reduction technique based on the tensorized Chebyshev interpolation. Lastly, in Chapter 6 we address high dimensionality via Monte Carlo simulation combined with function approximation.

In the rest of this chapter we introduce the option pricing problem. We start by providing

some basic definitions used throughout the thesis. Then, we provide the so-called risk-neutral pricing formula, and we give a short overview of existing numerical methods for option pricing. We conclude the chapter with the thesis' contributions.

1.1 Basic definitions

The following definitions are standard and can be found in any textbook about options and option pricing methods, see e.g. [1, 74, 19, 70, 108].

In financial markets, a *derivative* is a financial contract whose value depends on the value of an *underlying* asset or of a group of underlying assets. We consider derivatives issued at time $t = 0$ and expiring at time $T > 0$, where T is referred to as *time of maturity* or, in short, *maturity*. At time T the value of the derivative is determined by the price of the underlying assets up to time T.

A specific asset can be sold or bought at any given time t at a certain price S_t, called the *spot price*. All possible prices S_t as functions of t constitute the *asset price process* $(S_t)_{t\geq 0}$. We model the asset price process with a stochastic process. More details about it will be given in the next section.

Options are a type of derivatives. They are sold by one party, *the writer* of the option, to another, the *holder* of the option, for a certain price, the *option price*. The holder of an option has the right, but not the obligation, to make a specified transaction at a specified price prior to or at time of maturity T. There are different types of options, depending on the terms of the contract. Among the most popular are *European call* and *put options*. The holder of a European call option has the right to buy the underlying asset at time T for a specified price, the *strike price* K. Similarly, the European put option gives the right to sell the underlying asset at time T for a strike price K. Note that for these two examples the value of the option depends only on the asset price at time T, and they can be only exercised exactly at maturity T. *American call* and *put options*, instead, give the holder the right to buy, respectively sell, the underlying asset at *any* time $t \geq 0$ before or at maturity T. Since these options are specified by simple rules, they are usually referred to as *plain vanilla options*. Imposing more complicated terms to the contract gives rise to more complicated options, the so-called *exotic options*. Examples of exotic options are the so-called *path-dependent* options, whose value does not only depend on S_T, as

in the above mentioned European case, but on the whole history of the asset price, i.e. $(S_t)_{0\leq t\leq T}$. Examples of path-dependent options are *Asian, lookback* and *barrier* options.

The value of the option at time T is called *payoff* of the option, and the function that describes the payoff is referred to as *payoff function*. For example, for a European call option with strike value K the payoff function is given by

$$f(S_T) := (S_T - K)^+ = \begin{cases} S_T - K & \text{for } S_T > K \\ 0 & \text{otherwise.} \end{cases}$$

This is because if $S_T > K$, the holder exercises the option and buys the asset for the strike price K. The asset can then be immediately sold for the spot price S_T and the holder can make a gain of $S_T - K$. In this situation, the value of the option is $S_T - K$. In the case $K > S_T$ the right is not exercised because the asset can be purchased on the market for the cheaper price S_T. In this case, the option expires and its value is 0. Similarly, the payoff function of the European put option is given by $(K - S_T)^+$.

In the models considered in this thesis, we always assume the existence of a *riskless bank account* with fixed continuously compounded *risk-free interest rate* $r \geq 0$, i.e. putting 1 unit of currency in the riskless bank account at time $t = 0$ yields e^{rt} currency units at time t. Also, we assume that there are no transaction costs or taxes, and that no dividends are paid. Finally, the market is assumed to be *arbitrage-free*. This last concept is related to the risk neutral pricing approach that we explain in the next section.

1.2 Option pricing

The goal of this section is to derive the risk-neutral pricing formula, which yields a concrete way to compute the price of options. First, as already mentioned in the previous section, the price $(S_t)_{0\leq t\leq T}$ of the financial assets of interest is modeled by a stochastic process in continuous time $[0, T]$, where the maturity T is the *time horizon*. We assume that $(S_t)_{0\leq t\leq T}$ is defined on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$, where $\mathbb{F} := (\mathcal{F}_t)_{0\leq t\leq T}$ is a *filtration* that represents the information available in the model. Moreover, we assume that the filtered probability space satisfies the *usual conditions* and that $(S_t)_{0\leq t\leq T}$ is $\mathbb{F}$-*adapted*. In order to simplify the notation, we will denote the stochastic process $(S_t)_{0\leq t\leq T}$ by (S_t). We will use S_t to refer to the random variable with the index t. Similarly for

arbitrary stochastic processes and random variables.

We assume that our market model is arbitrage-free. In general, a market model is arbitrage-free if it is not possible to construct a *self-financing* portfolio whose value process, denoted by (V_t), is such that

- its value at initial time $t = 0$ is zero, i.e. $V_0 = 0$, and
- its value at some time $\widetilde{T} > 0$ is always nonnegative and can be positive, i.e. $V_{\widetilde{T}} \geq 0$ and $\mathbb{P}[V_{\widetilde{T}} > 0] > 0$.

Given an arbitrage-free model one can obtain derivative prices by the principle of *arbitrage-free pricing*, which states that the price process of the derivative should be chosen such that the joint model remains arbitrage-free. For example, consider a European call option whose underlying's price process is given by (S_t) and denote its price at time t by C_t. The idea is to define the price process (C_t), which is itself a stochastic process, such that the joint-model (S_t, C_t) remains free of arbitrage. The key tool to achieve this goal is the *fundamental theorem of asset pricing* (FTAP).

Fundamental theorem of asset pricing. Informally, the fundamental theorem of asset pricing states that:

The market model defined by $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ *and by the asset prices* (S_t) *is arbitrage-free if and only if there exists a probability measure* $\mathbb{Q}$*, which is equivalent to* $\mathbb{P}$ *and such that the discounted asset price process* $(e^{-rt}S_t)$ *is a* $\mathbb{Q}$*-martingale.*

The equivalent measure $\mathbb{Q}$ is usually referred to as *equivalent martingale measure* or *risk-neutral measure*. There are different ways to rigorously define the concept of "arbitrage-free" and to define the technical conditions needed such that the FTAP holds. We refer to the textbook [30] for an overview on the different definitions of arbitrage-free models and the different versions of the FTAP. Since the FTAP is not the focus of this thesis, for the sake of brevity we restrict our attention to the following important implications of the FTAP. First, the implication "$\Leftarrow$" is of practical relevance since it states that, if we construct a model where the discounted asset price process $(e^{-rt}S_t)$ is a martingale under a certain probability measure $\mathbb{Q}$, then our model is guaranteed to be free of arbitrage. The second important implication is that the FTAP yields a way to define the price of derivatives (C_t in our example above), the so-called *risk-neutral pricing formula*.

Risk-neutral pricing formula. Considering the above example of the joint market (S_t, C_t), the FTAP suggests that by specifying the European call option price as $C_t := e^{-r(T-t)}\mathbb{E}_{\mathbb{Q}}[C_T|\mathcal{F}_t]$ we are guaranteed to obtain a joint model (S_t, C_t) which is arbitrage-free. Extending the same reasoning to an arbitrary option with payoff function f and expiring at maturity T, the option price at time $t \in [0, T]$ is given by the *risk-neutral* pricing formula

$$e^{-r(T-t)}\mathbb{E}_{\mathbb{Q}}[f(S_T)|\mathcal{F}_t].$$

In particular, in the case that $\mathcal{F}_0$ is trivial, the option price at time $t = 0$ is given by

$$e^{-rT}\mathbb{E}_{\mathbb{Q}}[f(S_T)]. \tag{1.1}$$

We will use the formula (1.1) several times in this thesis. Moreover, since we will always be working in the risk-neutral setting, we will usually omit the "$\mathbb{Q}$" and write $e^{-rT}\mathbb{E}[f(S_T)]$.

In the case of American options the risk-neutral pricing procedure allows us to write the price at time $t = 0$ of an American option maturing at time T with payoff function f as

$$\sup_{\tau \in \mathcal{S}_{0,T}} \mathbb{E}[e^{-r\tau} f(S_\tau)], \tag{1.2}$$

where $\mathcal{S}_{0,T}$ is the set of all *stopping times* in $[0, T]$.

The risk-neutral pricing formula provides an easy way to establish the relation between the price of European put and call options that share the same underlying, strike price, and maturity. Indeed, denoting by C_t the price of a European call option at time t and by P_t the price of a European put option at time t, one can show (see e.g. [74]) that

$$C_t - P_t = S_t - Ke^{-r(T-t)}. \tag{1.3}$$

We refer to this formula as *put-call parity* and we will use it in some of our numerical examples in the next chapters.

Challenges in option pricing. On the one hand, the FTAP is a strong theoretical tool to derive the price of financial derivatives and, in particular, of options. On the other hand, new challenges in the development of models and option pricing techniques arise. These are the tasks that we need to carry out:

1. Develop a model (S_t) for the underlying assets that is rich enough in order to capture important empirical features that are observed in the financial markets.
2. At the same time, the model has to be tractable, simple enough, and should possess properties that can be used to compute expectations of the form (1.1) or (1.2).
3. With the developed model and its properties at hand, we can then develop numerical techniques for option pricing.

Furthermore, an option pricing technique should also satisfy some computational properties that are of practical relevance. For example, real-world applications request the delivery of computed option prices within short periods of time. Therefore:

4. The developed option pricing technique has to be as computationally efficient as possible.

Another important aspect that arises concerns the high dimensionality of the pricing problem: the underlying stochastic model developed in the first step might depend on many parameters, forcing us to work with high-dimensional parameter spaces. Also, high-dimensional pricing problems arise when considering multi-asset options. Thus:

5. The pricing technique needs to be able to handle high-dimensional problems.

Before explaining what the thesis' contributions are and how they are related to the items 1-5, we briefly summarize some classical option pricing techniques. Since it is impossible to consider all the existing techniques, we limit our summary to the ones that we will use as reference methods throughout the thesis. Furthermore, the following descriptions focus more on giving the general idea of the methods rather than on the technical details.

Closed-form formulas. For some specific models and types of options, quantities of the form (1.1) can be computed explicitly. In those cases, the option prices are given in closed form. A well known example is the one of European call and put options in the Black-Scholes model. In this model, which we present in detail in Chapter 2, the asset price S_t is log-normally distributed and the price at time t of a European call option expiring at time T with strike K is explicitly given by

$$S_t\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2), \tag{1.4}$$

where Φ is the cumulative distribution function of the standard normal distribution, and d_1 and d_2 are defined as

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left(\log\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right), \quad d_2 = d_1 - \sigma\sqrt{T-t}.$$

Here, σ denotes the volatility. A closed-form formula for the price of European put options can be derived by applying the put-call parity (1.3) to (1.4). Unfortunately, closed-form formulas are rare and most of the time one needs numerical techniques to evaluate the price of options. An example is that of Monte Carlo methods.

Monte Carlo methods. The core idea of the standard Monte Carlo method to compute expectation of random variables and, in particular, quantities of the form (1.1) is very simple and can be described as follows. First, we simulate N realizations of the random variable S_T and we denote them by S_T^i, for $i = 1, \ldots, N$. Then, (1.1) is approximated by

$$e^{-rT}\mathbb{E}[f(S_T)] \approx e^{-rT}\frac{1}{N}\sum_{i=1}^{N} f(S_T^i).$$

The simulation step is performed according to the information one has about the distribution of S_T. For example, if it is given explicitly one can simply generate random realizations according to it. If it is not known but (S_t) is given as a solution of a stochastic differential equation (SDE), then one can sample by discretizing the governing SDE of (S_t). More details will be given in Chapter 6.

Monte Carlo methods have been widely used in finance and several developments and variations have been proposed. Some examples are variance-reduction techniques, Quasi-Monte Carlo methods, multilevel Monte Carlo algorithms, and others. The textbook [50] gives a good overview on Monte Carlo methods for finance and we refer the reader to it for detailed descriptions of the algorithms. Furthermore, Monte Carlo methods can be also applied to compute the price (1.2) of American options. One of the main methods that performs this task is the one by Longstaff and Schwartz, developed in [94].

Fourier transform techniques. Fourier transform methods allow us to compute option prices under the assumption that the characteristic function of the asset price S_T is known, given explicitly, or that it can be easily computed. We recall that the characteristic function of an arbitrary random variable X is defined as $\phi_X(u) := \mathbb{E}[e^{iuX}]$. The idea

consists of expressing the option price (1.1) in terms of ϕ_{S_T} and of the Fourier transform of the payoff function. More precisely, let $\hat{f}$ be the Fourier transform of the payoff function, i.e.

$$\hat{f}(u) = \int_{\mathbb{R}} e^{ius} f(s) ds.$$

Then, under suitable integrability conditions, the Plancherel-Parseval's identity (see e.g. [104]) allows us to write the option price as

$$e^{-rT}\mathbb{E}[f(S_T)] = \frac{e^{-rT}}{2\pi} \int_{\mathbb{R}} \hat{f}(-u)\phi_{S_T}(u) du. \tag{1.5}$$

The last integral in (1.5) can then be numerically computed in different ways, e.g. by numerical integration or by Fast-Fourier-Transform algorithms, see e.g. [23].

Remark 1.1. *It is worth mentioning that the Fourier transform method can be used whenever the option price can be written in the form $\mathbb{E}[f(Y)]$, with a random variable Y whose characteristic function is available. For example, for some asset price models the characteristic function of the log-asset price X_t, defined through $e^{X_t} := S_t$, is available. In such a case, one can write $\mathbb{E}[f(S_T)]$ as $\mathbb{E}[\tilde{f}(X_T)]$ for $\tilde{f}(x) := f(e^x)$ and the Fourier method can be directly applied as described before.*

PDE approaches. Approaches based on partial differential equations (PDE) are an other important branch of methods for option pricing. If the price process (S_t) is *Markov* with respect to the filtration $\mathbb{F}$, the European option price (1.1) at time t can be written as a function $v(t,x)$ of t and $x := S_t$. More precisely, we can define it as $v(t,x) := e^{-r(T-t)}\mathbb{E}[f(S_T)|\mathcal{F}_t] = e^{-r(T-t)}\mathbb{E}[f(S_T)|S_t = x]$. Then, the core idea is to relate $v(t,x)$ to a PDE. The main tool to do it is the Feynman-Kac theorem (see e.g. [19]), which allows us to express $v(t,x)$ as solution of the PDE

$$\begin{aligned} &\partial_t v(t,x) + \mathcal{G}v(t,x) - rv(t,x) = 0, \quad (x,t) \in \mathbb{R}_+ \times [0,T), \\ &v(T,x) = f(x), \end{aligned}$$

where $\mathcal{G}$ is the *generator* of (S_t). For the case of American options, the price (1.2) of the American version of the option described before satisfies (see e.g. [1, 36]) the *partial*

differential complementarity problem (PDCP)

$$\begin{cases} \partial_t v(t,x) + \mathcal{G}v(t,x) - rv(t,x) \geq 0, \\ v(t,x) \geq f(x), \\ \Big(v(t,x) - f(x)\Big)\Big(\partial_t v(t,x) + \mathcal{G}v(t,x) - rv(t,x)\Big) = 0, \\ v(T,x) = f(x), \end{cases} \tag{1.6}$$

where the first three relations hold for $(x,t) \in \mathbb{R}_+ \times [0,T)$ and the last equation for $x \in \mathbb{R}_+$. The inequalities (1.6) are also referred to as *variational inequalities*. With the PDE or the PDCP at hand, the option prices can then be numerically computed by means of any type of PDE solvers, for example by finite difference discretization, see e.g. [36], or finite element discretization, see e.g. [70].

As previously mentioned, the literature of option pricing methods is vast and rich, and a big variety of techniques have been developed. Some further examples are algorithms based on quadrature rules, binomial tree models, and others. We refer to the survey paper [20] and to the textbooks [1, 70, 108] for a more complete overview.

1.3 Contributions of the thesis

We present the contributions of this thesis. For each chapter, we first describe its content. Then, at the end of each section we explain how the contribution is related to the development of asset price models and option pricing techniques, described through the items 1-5 in Section 1.2.

Chapter 2. This is a review chapter where we introduce the polynomial models. In particular, we present the definition and the main properties of polynomial jump-diffusions [39] and of the special case of polynomial diffusions [38]. We introduce the models that are used throughout the thesis and, finally, we present an option pricing technique based on polynomial expansions that is used as reference method in some of our numerical experiments.

This chapter mainly refers to the modeling part 1 and to the properties of the class of models that can be exploited for option pricing, item 2.

Chapter 3. We propose a methodology to numerically compute the price of European and American options in polynomial models. The method exploits the availability of all the conditional moments of the underlying asset price process to produce monotone sequences of lower and/or upper bounds of the option price. In particular, given a fixed moment order n, an upper/lower bound of the price is obtained by numerically solving a polynomial optimization problem that considers the information of all the moments of order at most n. By letting n go to infinity, the method produces the aforementioned monotone sequences of bounds.

For the European case, the starting point is the methodology developed in [14] that considers the same approach but under restrictive assumptions that limit its applications. In particular, it is assumed in [14] that the payoff function is piecewise linear and that only a fixed number of moments is available. In our work, we extend it to consider a more general class of payoff functions and a number of moments n that can go to infinity. This extension is of practical relevance in the setting of polynomial models where payoff functions are often not piecewise polynomial and the moments of all orders are available. In this relaxed setting, we present new convergence results that show that the bounds converge towards the option price under some appropriate assumptions. Moreover, we address the numerical solution of the involved optimization problems. First, we propose to use the same technique as in [14, 122] based on semidefinite programming. Then, we propose a new numerical technique based on the cutting plane procedure that better suits our new relaxed setting.

In the second part of the chapter we propose a new extension of the method that allows us to price American options in polynomial models. For this case, we explain how to adapt the numerical algorithms to solve the new obtained optimization problems. Finally, for both the European and the American case, we show numerical results that illustrate the effectiveness of our method.

This chapter refers to the step on the development of option pricing techniques for specified families of models, item 3.

Chapter 4. Some option pricing techniques for polynomial models require an incremental computation of the moment sequence. In other words, moments of order $0, 1, 2, 3, \ldots$ have to be computed sequentially until a dynamically evaluated criterion tells the procedure to stop. Two of these techniques are the one developed in Chapter 3 and the polynomial

expansion-based technique mentioned in Chapter 2. For both of them, increasing the order of the considered moments improves the quality of the option price approximation. Once a pre-specified precision is reached, the procedure can stop and the required option price is returned. In practice, such a computation translates into an incremental evaluation of a nested sequence of block upper triangular matrix exponentials. In this chapter we propose a new algorithm that performs this task efficiently. More precisely, our method is based on the scaling and squaring algorithm and, by carefully reusing certain intermediate quantities from one step to the next, it efficiently computes the required sequence of matrix exponentials. The choice of the scaling parameter is of particular importance in the scaling and squaring method. In our algorithm, we address this point by developing an adaptive strategy to select it in the most suitable way. Finally, we present numerical results that show the effectiveness of our method in concrete option pricing contexts.

Our new algorithm reduces the complexity of the pricing algorithms that require the aforementioned incremental moment computation and can therefore be related to item 4 of the development procedure presented in the Section 1.2.

Chapter 5. We develop a complexity reduction technique for high-dimensional option pricing. The starting point is the approach developed by Gass et al. in [46]. There, the authors propose a complexity reduction technique for parametric option pricing based on Chebyshev interpolation. The idea is as follows. First, the option price is considered as a function of model and payoff parameters. Then, the classical Chebyshev interpolation is used on the parameter space to increase the efficiency in computing option prices, while maintaining the required accuracy. The method can be split into two parts, an offline phase and an online phase. In the first one, the option prices are computed in the Chebyshev nodes by a reference method and the interpolation coefficients are evaluated. With the interpolation coefficients at hand, in the online phase the price can be efficiently computed in a new arbitrary set of parameters by a quick evaluation of the interpolating polynomial. In [46], this method is shown to be effective for low-dimensional parameter spaces. As the number of parameters increases, however, it is affected by the curse of dimensionality in both phases. In our work, we extend this approach to treat parameter spaces of high dimensions by exploiting low-rank structures. The core idea of our method is to express the tensorized Chebyshev interpolation in the tensor train format and to develop an efficient way, based on tensor completion, to approximate the interpolation coefficients. In particular, in the completion step we combine the approach developed in

[112] with a new adaptive sampling strategy to approximate the prices in the Chebyshev nodes. This results in a new complexity reduction of both the offline and the online phase. We present two numerical experiments. The first one concerns the computation of American option prices, while the second one treats European basket options. In these examples we treat parameter spaces of dimensions up to 25 and the numerical results confirm the effectiveness of our method compared to advanced reference techniques.

The contribution of this chapter is directly related to item 4 since it improves the efficiency of pricing algorithms. At the same time, it allows us to treat high-dimensional parameter spaces and, therefore, it is related to item 5, as well.

Chapter 6. We propose a methodology for pricing single and multi-asset European options. The starting point is the method for multivariate integration recently developed in [97] and denoted by MCLS (Monte Carlo with Least-Squares). There, the author proposes a variance reduction technique for multivariate integration with respect to the Lebesgue measure. The main idea is to combine the Monte Carlo simulation with a function approximation step, where the integrating function is approximated by a least-squares approach. First, we extend MCLS to compute multivariate integrals with respect to general probability measures. This is done by changing the simulation step of MCLS and it allows us to compute quantities of the form (1.1), i.e. the price of (multi-asset) European options. Then, after noticing that for large-scale applications MCLS faces some memory limitations due to the large storage requirement for running it, we propose to combine it with the optimal sampling strategy from [26] (this was already proposed in [97]) along with the randomized extended Kaczmarz algorithm [124] for solving the least-squares problem. We propose a new cost analysis where we show that MCLS asymptotically becomes more accurate than the standard Monte Carlo approach at the same cost. Finally, we apply our extended version of MCLS to numerically compute single and multi-asset European option prices in some polynomial models and a large-scale multivariate integral. The obtained results show the effectiveness of the method and of our extension in both low and high dimensions.

The method offers an efficient way to deal with the high dimensionality arising when treating multi-asset options. This chapter is therefore mainly related to item 5 in the development process of option pricing techniques.

2 Polynomial models

Polynomial models build upon a particular type of stochastic processes: polynomial (jump-)diffusions. As explained in [38, 39], one of the key properties of polynomial (jump-)diffusions is that their conditional moments can be computed in closed form, which renders them computationally tractable and suited for financial asset pricing models. Moreover, their dynamics are shown to be flexible and able to capture important empirical features of financial time series. Thanks to these properties, polynomial models provide a tractable and flexible framework for financial modeling and have become a versatile tool in a wide range of applications in finance. Examples include the modeling of interest rates [31, 123, 40], stochastic volatility [54, 4], exchange rates [87], life insurance liabilities [15], variance swaps [44], credit risk [2], dividend futures [43], commodities [41], and stochastic portfolio theory [27]. Apart from modeling, polynomial (jump-)diffusions can also be used to improve existing computational techniques. For example, a variance reduction technique for option pricing and hedging based on polynomial (jump-)diffusions is developed in [28]. It is also worth mentioning that a large set of commonly used processes are polynomial (jump-)diffusions. Examples are Ornstein-Uhlenbeck processes, square-root diffusions, and Lévy processes.

The goal of this chapter is to introduce all the ingredients required to work in the framework of polynomial models. More precisely, in Section 2.1 we define the classes of polynomial jump-diffusions and polynomial diffusions and we present some of their properties. These properties will be used in the development of our pricing techniques. Then, in Section 2.2 we provide a list of examples, which are used throughout the thesis. Finally, in Section 2.3 we present a technique for European option pricing based on

polynomial expansions. This approach can be used in the setting of polynomial models and we use it as a reference approach in some of our examples throughout the thesis.

2.1 Polynomial jump-diffusions and polynomial diffusions

Polynomial jump-diffusions have appeared in the literature for a long time (see e.g. [120]) and have been applied in financial modeling since the early 2000s (see e.g. [123]). The mathematical foundations for polynomial models in finance have been provided in the recent articles [28, 38, 39]. The following summary is mostly based on [39] for polynomial jump-diffusions, and on [38] for polynomial diffusions.

We start by introducing some notation and basic definitions. A *d-variate polynomial* p on $\mathbb{R}^d$ is a map $\mathbb{R}^d \to \mathbb{R}$ of the form $p(\mathbf{x}) = \sum_{\mathbf{k}} \alpha_{\mathbf{k}} \mathbf{x}^{\mathbf{k}}$, where the sum runs over all multi-indices $\mathbf{k} = (k_1, \dots, k_d) \in \mathbb{N}_0^d$ and only finitely many coefficients $\alpha_{\mathbf{k}} \in \mathbb{R}$ are nonzero. Note that $\mathbf{x}^{\mathbf{k}} := x_1^{k_1} \dots x_d^{k_d}$. The *degree* of p is defined as $\deg(p) = \max\{|\mathbf{k}| : \alpha_{\mathbf{k}} \neq 0\}$, for $|\mathbf{k}| := k_1 + \cdots + k_d$. The degree of the zero polynomial is set to 0 by definition. We denote by $\mathrm{Pol}(\mathbb{R}^d)$ the space of all d-variate polynomials and by $\mathrm{Pol}_n(\mathbb{R}^d)$ the space of all d-variate polynomials of degree at most n, i.e.

$$\mathrm{Pol}_n(\mathbb{R}^d) := \left\{ \sum_{0 \leq |\mathbf{k}| \leq n} \alpha_{\mathbf{k}} \mathbf{x}^{\mathbf{k}} \ : \ \mathbf{x} \in \mathbb{R}^d, \alpha_{\mathbf{k}} \in \mathbb{R} \right\}.$$

We now consider a subset E of $\mathbb{R}^d$. We say that p is a *polynomial on* E if it is the restriction on E of a polynomial $q \in \mathrm{Pol}(\mathbb{R}^d)$, i.e. $p := q|_E$. Its degree is defined as $\deg(p) := \min\{\deg(q) \ : \ p = q|_E, q \in \mathrm{Pol}(\mathbb{R}^d)\}$. Similarly to before, we define $\mathrm{Pol}(E)$ to be the space of all d-variate polynomials on E and $\mathrm{Pol}_n(E)$ the space of all d-variate polynomials on E of degree at most n. If E has a non-empty interior $\mathrm{Pol}_n(E)$ and $\mathrm{Pol}_n(\mathbb{R}^d)$ can be identified and have the same finite dimension. We note that the dimension of $\mathrm{Pol}_n(\mathbb{R}^d)$ is given by $N(n,d) := \binom{n+d}{n}$. Finally, for the rest of the chapter, we fix a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{Q})$ which satisfies the usual conditions.

Polynomial jump-diffusions. We define polynomial jump-diffusions through their generator, as done in [39]. We consider the operator

$$\mathcal{G}f(\mathbf{x}) = \frac{1}{2} \mathrm{Tr}(A \nabla^2 f(\mathbf{x})) + b^T \nabla f(\mathbf{x}) + \int_{\mathbb{R}^d} \Big(f(\mathbf{x} + \xi) - f(\mathbf{x}) - \xi^T \nabla f(\mathbf{x}) \Big) \nu(\mathbf{x}, d\xi),$$

for some measurable maps $A : \mathbb{R}^d \to \mathbb{S}_+^d$ and $b : \mathbb{R}^d \to \mathbb{R}^d$, where $\mathbb{S}_+^d$ is the space of positive semidefinite matrices. Also, $\nu(\mathbf{x}, d\xi)$ is a transition kernel from $\mathbb{R}^d$ into $\mathbb{R}^d$ satisfying $\nu(x, \{0\}) = 0$ and $\int_{\mathbb{R}^d} \|\xi\| \wedge \|\xi\|^2 \nu(\mathbf{x}, d\xi) < \infty$ for all $\mathbf{x} \in \mathbb{R}^d$. We clarify that $\nabla f(\mathbf{x})$ is defined as $\nabla f(\mathbf{x}) := (\frac{\partial}{\partial x_1} f(\mathbf{x}), \cdots, \frac{\partial}{\partial x_d} f(\mathbf{x}))^\top$ and

$$\nabla^2 f(\mathbf{x}) := \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f(\mathbf{x}) & \frac{\partial^2}{\partial x_1 x_2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_1 x_d} f(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 x_1} f(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_2 x_d} f(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_d x_1} f(\mathbf{x}) & \frac{\partial^2}{\partial x_d x_2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_d^2} f(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{d \times d}.$$

We say that the operator $\mathcal{G}$ is *well defined* on $\mathrm{Pol}(E)$ if $\mathcal{G}f = 0$ on E for any $f \in \mathrm{Pol}(\mathbb{R}^d)$ with $f = 0$ on E and if $\int_{\mathbb{R}^d} \|\xi\|^n \nu(\mathbf{x}, d\xi) < \infty$ for all $\mathbf{x} \in E$ and all $n \geq 2$.

We assume that $A(\mathbf{x})$, $b(\mathbf{x})$ and the transition kernel $\nu(\mathbf{x}, d\xi)$ satisfy

$$\begin{aligned} b &\in \mathrm{Pol}_1(E), \\ A + \int_{\mathbb{R}^d} \xi\xi^T \nu(\cdot, d\xi) &\in \mathrm{Pol}_2(E), \\ \int_{\mathbb{R}^d} \xi^{\mathbf{k}} \nu(\cdot, d\xi) &\in \mathrm{Pol}_{|\mathbf{k}|}(E), \end{aligned} \tag{2.1}$$

for all $|\mathbf{k}| \geq 3$. Lemma 2.2 in [39] states that if $\mathcal{G}$ satisfies the conditions (2.1), then it maps $\mathrm{Pol}_n(E)$ to itself for each $n \in \mathbb{N}$, i.e. the invariant property

$$\mathcal{G}\mathrm{Pol}_n(\mathbb{R}^d) \subseteq \mathrm{Pol}_n(\mathbb{R}^d) \tag{2.2}$$

holds. We finally define polynomial jump-diffusions.

Definition 2.1 (Definition 2.1 in [39])**.** *The operator $\mathcal{G}$ is called polynomial on E if it is well defined on* $\mathrm{Pol}(E)$ *and maps* $\mathrm{Pol}_n(E)$ *to itself for each $n \in \mathbb{N}$. Let $(\mathbf{X}_t)$ be an E-valued jump-diffusion with generator $\mathcal{G}$. In this case, we call $(\mathbf{X}_t)$ a polynomial jump-diffusion on E.*

Therefore, a polynomial jump-diffusion is a jump-diffusion with *state space* E whose jump measure admits moments of all orders, with generator that maps polynomials on E to polynomials on E of lower or equal degree.

In the special case of a vanishing jump measure, the process $(\mathbf{X}_t)$ exhibits almost surely continuous paths, i.e. it is a process without jumps. This special class is referred to as the class of **polynomial diffusions** and it is studied in [38]. Polynomial diffusions play an important role in the thesis. We therefore dedicate some space to this special case.

For a polynomial diffusion $(\mathbf{X}_t)$, the corresponding generator takes the form

$$\mathcal{G}f(\mathbf{x}) = \frac{1}{2}\operatorname{Tr}(A\nabla^2 f(\mathbf{x})) + b^T \nabla f(\mathbf{x}),$$

and the conditions (2.1) simplify to

$$A_{ij} \in \operatorname{Pol}_2(\mathbb{R}^d), \qquad b_i \in \operatorname{Pol}_1(\mathbb{R}^d), \quad \text{for} \quad i,j = 1,\dots,d. \tag{2.3}$$

Moreover, if A is positive definite, $(\mathbf{X}_t)$ can also be defined as an E-valued solution of the stochastic differential equation (SDE)

$$d\mathbf{X}_t = b(\mathbf{X}_t)dt + \Sigma(\mathbf{X}_t)d\mathbf{W}_t, \tag{2.4}$$

where $(\mathbf{W}_t)$ is a d-dimensional Brownian motion and Σ is uniquely defined via $A := \Sigma\Sigma^T$.

Next, we derive a closed form formula for computing conditional moments of polynomial jump-diffusions, i.e. quantities of the form $\mathbb{E}[p(\mathbf{X}_T)|\mathcal{F}_t]$ where p is a polynomial. We refer to this formula as *moment formula* and this is the most important result about polynomial jump-diffusions.

The moment formula. Let $(\mathbf{X}_t)$ be an E-valued polynomial jump-diffusion with generator $\mathcal{G}$. We fix a basis of polynomials $\mathcal{H}_n = \{h_1,\dots,h_N\}$ for $\operatorname{Pol}_n(E)$ and we write

$$H_n(\mathbf{x}) = (h_1(\mathbf{x}),\dots,h_N(\mathbf{x})).$$

Thanks to the property (2.2), the restriction of $\mathcal{G}$ to $\operatorname{Pol}_n(E)$ possesses a unique matrix representation G_n with respect to $\mathcal{H}_n$, for any fixed $n \in \mathbb{N}$. More precisely, for any $p \in \operatorname{Pol}_n(E)$ with coordinate vector $\vec{p} \in \mathbb{R}^N$ with respect to $\mathcal{H}_n$, we can write

$$\mathcal{G}p(\mathbf{x}) = H_n(\mathbf{x})G_n\vec{p}. \tag{2.5}$$

In the following, we make use of the notion of generalized conditional expectation, which

is defined for any σ-field $\mathcal{F}' \subseteq \mathcal{F}$ and all random variables Y by

$$\mathbb{E}[Y|\mathcal{F}'] = \begin{cases} \mathbb{E}[Y^+|\mathcal{F}'] - \mathbb{E}[Y^-|\mathcal{F}'], & \text{on } \{\mathbb{E}[|Y||\mathcal{F}'] < \infty\}, \\ +\infty, & \text{elsewhere.} \end{cases}$$

The moment formula for polynomial jump-diffusions is stated and proven in [39, Theorem 2.5], which we review in the following theorem.

Theorem 2.2 (Moment formula)**.** *Assume $\mathcal{G}$ is polynomial on E. Then for any $p \in \mathrm{Pol}_n(E)$ with coordinate representation $\vec{p} \in \mathbb{R}^N$, we have*

$$\mathbb{E}[p(\mathbf{X}_T)|\mathcal{F}_t] = H_n(\mathbf{X}_t)e^{G_n(T-t)}\vec{p}, \quad \text{for } t \leq T. \tag{2.6}$$

We recall that e^B for an arbitrary square matrix $B \in \mathbb{R}^{d\times d}$ represents the matrix exponential of B, defined as

$$e^B := \sum_{\ell=0}^{\infty} \frac{B^\ell}{\ell!}.$$

In the particular case where $\mathcal{F}_0$ is trivial, for $t = 0$ the moment formula (2.6) becomes

$$\mathbb{E}[p(\mathbf{X}_T)] = H_n(\mathbf{X}_0)e^{G_n T}\vec{p}. \tag{2.7}$$

We will use the form (2.7) several times throughout the thesis.

We now want to have more insights into the matrix G_n and its structure. We start having a closer look at it in the next example, where we consider scalar polynomial diffusions.

Example 2.3. *We consider the one-dimensional case $d = 1$. In view of the conditions* (2.3)*, the SDE* (2.4) *of a scalar polynomial diffusion (X_t) must necessarily be of the form*

$$dX_t = (b + \beta X_t)\, dt + \sqrt{a + \alpha X_t + AX_t^2}\, dW_t,$$

for some real values b, β, a, α, A. The generator takes the form

$$\mathcal{G}f(x) = (b + \beta x)\frac{\partial}{\partial x}f(x) + \frac{1}{2}(a + \alpha x + Ax^2)\frac{\partial^2}{\partial x^2}f(x).$$

Let $\{1, x, \cdots, x^n\}$ be the monomial basis of $\mathrm{Pol}_n(\mathbb{R})$. We apply $\mathcal{G}$ to an arbitrary basis

element x^k *and get*

$$\mathcal{G}x^k = k(k-1)\frac{a}{2}x^{k-2} + k\Big(b + (k-1)\frac{\alpha}{2}\Big)x^{k-1} + k\Big(\beta + (k-1)\frac{A}{2}\Big)x^k.$$

The columns of the matrix G_n *are the coordinate vectors of* $\mathcal{G}x^k$ *for* $k = 0, \cdots, n$*. Therefore,* G_n *can be explicitly constructed as*

$$\begin{bmatrix} 0 & b & 2\frac{a}{2} & 0 & \cdots & 0 \\ 0 & \beta & 2\left(b+\frac{\alpha}{2}\right) & 3\cdot 2\frac{a}{2} & 0 & \vdots \\ 0 & 0 & 2\left(\beta+\frac{A}{2}\right) & 3\left(b+2\frac{\alpha}{2}\right) & \ddots & 0 \\ 0 & 0 & 0 & 3\left(\beta+2\frac{A}{2}\right) & \ddots & n(n-1)\frac{a}{2} \\ \vdots & & & 0 & \ddots & n\left(b+(n-1)\frac{\alpha}{2}\right) \\ 0 & & \ldots & & 0 & n\left(\beta+(n-1)\frac{A}{2}\right) \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}. \tag{2.8}$$

Structure of the matrix G_n**.** The structure of the matrix G_n plays an important role in the thesis. In this paragraph, we take a closer look at it and we highlight some important properties. We start by noting that the polynomial subspaces $\mathrm{Pol}_n(E)$ for $n = 0, 1, 2, \cdots$ define a nested sequence of finite-dimensional subspaces of the infinite-dimensional vector space $\mathrm{Pol}(E)$, i.e.

$$\mathrm{Pol}_0(E) \subseteq \mathrm{Pol}_1(E) \subseteq \mathrm{Pol}_2(E) \subseteq \cdots \subseteq \mathrm{Pol}(E).$$

Consequently, the bases $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2 \ldots$ can be chosen to form a sequence of nested bases, i.e.

$$\mathcal{H}_0 \subseteq \mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \ldots. \tag{2.9}$$

Consider now the sequence $G_0, G_1, G_2, \ldots$ of matrix representations of $\mathcal{G}$ restricted to the spaces $\mathrm{Pol}_0(E), \mathrm{Pol}_1(E), \mathrm{Pol}_2(E), \ldots$ as defined in (2.5). Thanks to the property (2.2) which holds for any $n \in \mathbb{N}$, and due to the nestedness (2.9) of the bases, G_n is constructed from G_{n-1} by adding the columns representing the action of $\mathcal{G}$ to $\mathcal{H}_n \setminus \mathcal{H}_{n-1}$. This last expression corresponds to the basis polynomials of exact degree n. Hence, the matrix G_n can be partitioned as

$$G_n = \begin{bmatrix} G_{n-1} & g_n \\ 0 & G_{n,n} \end{bmatrix} \in \mathbb{R}^{N(n,d)\times N(n,d)}.$$

The diagonal block $G_{n,n}$ is of size $\binom{n+d-1}{n}$, the dimension of the space of d-variate polynomials of exact degree n. By iterating the same reasoning, we can write G_n as

$$G_n = \begin{bmatrix} G_{0,0} & G_{0,1} & \cdots & G_{0,n} \\ & G_{1,1} & \cdots & G_{1,n} \\ & & \ddots & \vdots \\ & & & G_{n,n} \end{bmatrix}.$$

Hence, G_n is block upper triangular and the sizes of the square diagonal blocks are

$$1, d, \binom{1+d}{2}, \ldots, \binom{n+d-1}{n}.$$

To conclude this description, we note that the sequence $G_0, G_1, G_2, \ldots$ forms a nested sequence of block upper triangular matrices. This property will prove useful in the rest of the thesis.

The paper [38] presents several other results about polynomial diffusions. For example, existence and uniqueness of solutions to (2.4) on several types of state spaces $E \subseteq \mathbb{R}^d$ and for large classes of Σ and b are studied. Also, the authors provide a large set of examples and other applications. Similarly, [39] contains more results on polynomial jump-diffusions, including examples, applications and other properties. As these results are not explicitly needed in this thesis, we do not present them and we refer to [38, 39] for more details.

2.2 Examples

In this section we present some examples of polynomial models. All of them belong to the class of polynomial diffusion models and are used throughout the thesis to verify the developed pricing techniques. In the context introduced in Section 1, polynomial diffusions are used to model the asset price process (S_t) in the risk-neutral setting. In particular, the discounted price process $(e^{-rt}S_t)$ is a martingale in the considered filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{Q})$ and $\mathbb{Q}$ is a risk-neutral measure.

We present three models: the multi-dimensional Black-Scholes model, the Heston stochastic volatility model and the Jacobi stochastic volatility model. Also, as a particular case of the multi-dimensional Black-Scholes model, we devote some space to the one-dimensional

Black-Scholes model in the log-asset price formulation, which will be of particular importance in the rest of this thesis. For the Jacobi stochastic volatility model, our summary is based on [4]. The Heston model and the Black-Scholes models, being very popular, are described in several textbooks, see e.g. [36].

Multi-dimensional Black-Scholes model. In the d-dimensional Black-Scholes model we consider d assets whose price process $(\mathbf{S}_t) \in \mathbb{R}^d$ is given by

$$dS_t^i = rS_t^i + \sigma_i S_t^i dW_t^i, \quad i = 1, \cdots, d, \tag{2.10}$$

for some volatility values σ_i, $i = 1, \cdots, d$, a risk-free interest rate r and d correlated Brownian motions (W_t^i) with correlation parameters $\rho_{ij} \in [-1, 1]$, for $i \neq j$. For any deterministic starting vector $\mathbf{S}_0 \in \mathbb{R}_+^d$ the price process takes values in the state space $E = \mathbb{R}_+^d$. Solving explicitly (2.10) we can write the price process as

$$S_t^i = S_0^i \exp\Big((r - \frac{\sigma_i^2}{2})t + \sigma_i W_t^i\Big), \quad i = 1, \cdots, d.$$

The generator $\mathcal{G}$ of $(\mathbf{S}_t)$ is given by

$$\mathcal{G}f = \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{ij}s_i s_j\frac{\partial^2}{\partial s_i s_j}f + r\sum_{i=1}^{d}s_i\frac{\partial}{\partial s_i}f.$$

The matrix G_n can be computed with respect to the monomial basis as in the following lemma, and turns out to be diagonal.

Lemma 2.4. *Let $\mathcal{H}_n$ be the monomial basis of $Pol_n(\mathbb{R}_+^d)$. Let*

$$\pi : \mathcal{E} \to \left\{1, \ldots, \binom{n+d}{n}\right\}$$

be an enumeration of the set of tuples $\mathcal{E} = \{\mathbf{k} \in \mathbb{N}_0^d : |\mathbf{k}| \leq n\}$. Then, the matrix representation G_n of the generator of the process $(S_t^1, \cdots, S_t^d)$ with respect to $\mathcal{H}_n$ and restricted to $Pol_n(\mathbb{R}_+^d)$ is diagonal with diagonal entries defined as

$$G_{\pi(\mathbf{k}),\pi(\mathbf{k})} = \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{ij}(k_i k_j \mathbf{1}_{i\neq j} + k_i(k_i - 1)\mathbf{1}_{i=j}) + r\sum_{i=1}^{d}k_i,$$

for all $\mathbf{k} \in \mathcal{E}$.

Proof. We apply the generator $\mathcal{G}$ of $(S_t^1, \cdots, S_t^d)$ to a monomial basis element of the form $s_1^{k_1} \cdots s_d^{k_d}$ and we obtain

$$\mathcal{G} s_1^{k_1} \cdots s_d^{k_d} = s_1^{k_1} \cdots s_d^{k_d} \left(\frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} (k_i k_j \mathbf{1}_{i \neq j} + k_i (k_i - 1) \mathbf{1}_{i=j}) + r \sum_{i=1}^{d} k_i \right).$$

It follows that G_n is diagonal as stated above. □

We apply the moment formula and exploit the diagonal structure of G_n to get the moments of $\mathbf{S}_T$ explicitly.

Corollary 2.5. *The moments of* $\mathbf{S}_T$ *are given by*

$$\mathbb{E}[\mathbf{S}_T^{\mathbf{k}}] = \mathbf{S}_0^{\mathbf{k}} \exp \left(\frac{T}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} (k_i k_j \mathbf{1}_{i \neq j} + k_i (k_i - 1) \mathbf{1}_{i=j}) + rT \sum_{i=1}^{d} k_i \right),$$

for any multi-index $\mathbf{k} \in \mathbb{N}_0^d$.

The log-asset price formulation of the one-dimensional Black-Scholes model is used several times throughout this thesis. We therefore have a closer look at it.

One-dimensional Black-Scholes model. In the case $d = 1$, the price S_t at time t can be written as $S_t = e^{X_t}$, where (X_t) is defined as solution of the SDE

$$dX_t = (r - \frac{\sigma^2}{2})dt + \sigma dW_t, \tag{2.11}$$

where σ is the volatility parameter, $r \geq 0$ a risk-free interest rate and (W_t) is a one-dimensional Brownian motion. We refer to (X_t) as the *log-asset price* process. The state space of X_t is $E = \mathbb{R}$. It can be directly shown that X_t is explicitly given as

$$X_t = X_0 + \left(r - \frac{1}{2} \sigma^2 \right) t + \sigma W_t,$$

where $X_0 \in \mathbb{R}$ is a deterministic starting value. Therefore, X_t is normal distributed with mean $X_0 + \left(r - \frac{1}{2} \sigma^2 \right) t$ and variance $\sigma^2 t$. Its generator is given by

$$\mathcal{G} f = \left(r - \frac{1}{2} \sigma^2 \right) \frac{\partial}{\partial x} f + \frac{1}{2} \sigma^2 \frac{\partial^2}{\partial x^2} f.$$

Since (X_t) is a scalar polynomial diffusion, the matrix G_n can be constructed with respect to the monomial basis as in (2.8) by replacing the variables according to (2.11), i.e.

$$b = r - \frac{1}{2}\sigma^2, \;\; a = \sigma^2, \;\; \alpha = \beta = A = 0.$$

Similarly, it is possible to construct G_n with respect to other bases of $\mathrm{Pol}_n(\mathbb{R})$.

Heston stochastic volatility model. The Heston model is a stochastic volatility model, meaning that the volatility is stochastic itself. The log-asset price process (X_t) is defined as the solution of the SDE

$$dX_t = \Big(r - \frac{V_t}{2}\Big)dt + \rho\sqrt{V_t}dW_t^1 + \sqrt{V_t}\sqrt{1-\rho^2}dW_t^2$$

and the squared stochastic volatility follows

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t}dW_t^1.$$

The two Brownian motions (W_t^1) and (W_t^2) are independent and the model parameters satisfy the conditions $\kappa \geq 0$, $\theta \geq 0$, $\sigma > 0$, $r \geq 0$, $\rho \in [-1, 1]$. For any deterministic starting value $(X_0, V_0) \in \mathbb{R} \times \mathbb{R}_+$ the process (X_t, V_t) takes values in the state space $E = \mathbb{R} \times \mathbb{R}_+$. While the Heston model (X_t, V_t) is a polynomial diffusion in two dimensions, its asset price formulation (S_t, V_t), as originally introduced in [66], is not polynomial. The generator is given by

$$\mathcal{G}f(x,v) = \frac{1}{2}\operatorname{Tr}(A(v)\nabla^2 f(x,v)) + b(v)^T \nabla f(x,v)$$

for

$$b(v) = \begin{bmatrix} r - v/2 \\ \kappa(\theta - v) \end{bmatrix}, \qquad A(v) = \begin{bmatrix} v & \rho\sigma v \\ \rho\sigma v & \sigma^2 v \end{bmatrix}.$$

Hence

$$\mathcal{G}f = \Big(r - \frac{1}{2}v\Big)\frac{\partial}{\partial x}f + \kappa(\theta - v)\frac{\partial}{\partial v}f + \frac{1}{2}\sigma^2 v\frac{\partial^2}{\partial v^2}f + \sigma\rho v\frac{\partial^2}{\partial v \partial x}f + v\frac{\partial^2}{\partial x^2}f.$$

In order to construct the matrix G_n, we consider an arbitrary element $x^p v^q$ of the

monomial basis of $\mathrm{Pol}_n(E)$ and we apply $\mathcal{G}$ to it. We obtain

$$\begin{aligned}\mathcal{G}x^p v^q =& x^p v^{q-1}\Big(\kappa\theta q + \frac{1}{2}\sigma^2 q(q-1)\Big) + x^{p-1}v^q(rp + \rho\sigma qp) + \\ & x^{p-2}v^{q+1}\Big(\frac{1}{2}p(p-1)\Big) + x^p v^q(-\kappa q) + x^{p-1}v^{q+1}\Big(-\frac{1}{2}p\Big).\end{aligned} \tag{2.12}$$

We consider the monomial basis of $\mathrm{Pol}_n(E)$ ordered as in

$$H_n(x,v) = (1, x, v, x^2, xv, v^2, \ldots, x^n, x^{n-1}v, \ldots, v^n) \in \mathbb{R}^{1\times N(n,2)}, \tag{2.13}$$

where $N(n,2)$ is explicitly given by $\frac{(n+1)(n+2)}{2}$. An enumeration $\pi : \mathcal{E} \to \{1, 2, \ldots, N(n,2)\}$ of the set of pairs $\mathcal{E} := \{(p,q) \in \mathbb{N}_0 \times \mathbb{N}_0 : p + q \leq n\}$ corresponding to the basis ordering as in (2.13) is given by $(p,q) \mapsto \pi(p,q) = \frac{1}{2}(p+q+1)(p+q) + q + 1$. From (2.12) we can conclude that G_n has at most 5 nonzero elements in each column $\pi(p,q)$, given by

$$\begin{aligned} G_{\pi(p,q-1),\pi(p,q)} &= \kappa\theta q + \frac{1}{2}\sigma^2 q(q-1), && (q \geq 1)\\ G_{\pi(p-1,q),\pi(p,q)} &= rp + \rho\sigma qp, && (p \geq 1)\\ G_{\pi(p-2,q+1),\pi(p,q)} &= \frac{1}{2}p(p-1), && (p \geq 2)\\ G_{\pi(p,q),\pi(p,q)} &= -\kappa q, \\ G_{\pi(p-1,q+1),\pi(p,q)} &= -\frac{1}{2}p. && (p \geq 1)\end{aligned}$$

This allows us to construct G_n with respect to the monomial basis. In the following corollary, which directly derives from the moment formula (2.7), we give an explicit formula for computing the moments of the log-asset price X_T.

Corollary 2.6. *The first n moments of the log-asset price X_T are given by*

$$\mathbb{E}[X_T^p] = H_n(X_0, V_0)e^{G_n T}e_{\pi(p,0)}, \quad 0 \leq p \leq n, \tag{2.14}$$

where H_n, G_n and π are defined as above, and e_i is the i-th standard basis vector in $\mathbb{R}^{N(n,2)}$.

Jacobi stochastic volatility model. The Jacobi model is another stochastic volatility model. As explained in [4], it contains the Heston model as a limit case, see [4, Theorem

2.3]. The log-asset prices (X_t) and the squared volatility process (V_t) are given by

$$\begin{aligned} dX_t &= \Big(r - \frac{V_t}{2}\Big)dt + \rho\sqrt{Q(V_t)}dW_t^1 + \sqrt{V_t - \rho^2 Q(V_t)}dW_t^2, \\ dV_t &= \kappa(\theta - V_t)dt + \sigma\sqrt{Q(V_t)}dW_t^1, \end{aligned}$$

where

$$Q(v) = \frac{(v - v_{\min})(v_{\max} - v)}{(\sqrt{v_{\max}} - \sqrt{v_{\min}})^2},$$

for some $0 \leq v_{\min} < v_{\max}$. Here, (W_t^1) and (W_t^2) are independent standard Brownian motions and the model parameters satisfy the conditions $\kappa \geq 0$, $\theta \in [v_{\min}, v_{\max}]$, $\sigma > 0$, $r \geq 0$, $\rho \in [-1, 1]$. For any deterministic starting value $(X_0, V_0) \in \mathbb{R} \times [v_{\min}, v_{\max}]$ the process (X_t, V_t) takes values in the state space $E = \mathbb{R} \times [v_{\min}, v_{\max}]$. The generator $\mathcal{G}$ in the Jacobi model takes the form

$$\mathcal{G}f(x, v) = \frac{1}{2}\operatorname{Tr}(A(v)\nabla^2 f(x, v)) + b(v)^T \nabla f(x, v),$$

where

$$b(v) = \begin{bmatrix} r - v/2 \\ \kappa(\theta - v) \end{bmatrix}, \quad A(v) = \begin{bmatrix} v & \rho\sigma Q(v) \\ \rho\sigma Q(v) & \sigma^2 Q(v) \end{bmatrix}.$$

It follows that

$$\mathcal{G}f = \Big(r - \frac{1}{2}v\Big)\frac{\partial}{\partial x}f + \kappa(\theta - v)\frac{\partial}{\partial v}f + \frac{1}{2}\sigma^2 Q(v)\frac{\partial^2}{\partial v^2}f + \sigma\rho Q(v)\frac{\partial^2}{\partial v \partial x}f + v\frac{\partial^2}{\partial x^2}f.$$

We explain how to construct G_n. We consider the same basis (2.13) as in the Heston model and we set $S := (\sqrt{v_{\max}} - \sqrt{v_{\min}})^2$. The action of the generator $\mathcal{G}$ on $x^p v^q$ yields

$$\begin{aligned} \mathcal{G}x^p v^q =& x^{p-2}v^{q+1}\frac{1}{2}p(p-1) - x^{p-1}v^{q+1}p\Big(\frac{1}{2} + \frac{q\rho\sigma}{S}\Big) + x^{p-1}v^q p\Big(r + q\rho\sigma\frac{v_{\max} + v_{\min}}{S}\Big) \\ &- x^{p-1}v^{q-1}\frac{pq\rho\sigma v_{\max}v_{\min}}{S} - x^p v^q q\Big(\kappa + \frac{q-1}{2}\frac{\sigma^2}{S}\Big) \\ &- x^p v^{q-2}q\frac{q-1}{2}\frac{\sigma^2 v_{\max}v_{\min}}{S} + x^p v^{q-1}q\Big(\kappa\theta + \frac{q-1}{2}\sigma^2\frac{v_{\max} + v_{\min}}{S}\Big). \end{aligned}$$

Therefore, G_n has at most 7 nonzero elements in each column $\pi(p,q)$, given by

$$\begin{aligned}
G_{\pi(p,q-1),\pi(p,q)} &= q\left(\kappa\theta + \frac{q-1}{2}\sigma^2\frac{v_{\max}+v_{\min}}{S}\right), && (q \geq 1)\\
G_{\pi(p-1,q),\pi(p,q)} &= p\left(r + q\rho\sigma\frac{v_{\max}+v_{\min}}{S}\right), && (p \geq 1)\\
G_{\pi(p-2,q+1),\pi(p,q)} &= \frac{1}{2}p(p-1), && (p \geq 2)\\
G_{\pi(p-1,q-1),\pi(p,q)} &= -\frac{pq\rho\sigma v_{\max}v_{\min}}{S}, && (p,q \geq 1)\\
G_{\pi(p,q-2),\pi(p,q)} &= -q\frac{q-1}{2}\frac{\sigma^2 v_{\max}v_{\min}}{S}, && (q \geq 2)\\
G_{\pi(p,q),\pi(p,q)} &= -q\left(\kappa + \frac{q-1}{2}\frac{\sigma^2}{S}\right),\\
G_{\pi(p-1,q+1),\pi(p,q)} &= -p\left(\frac{1}{2} + \frac{q\rho\sigma}{S}\right). && (p \geq 1)
\end{aligned}$$

For instance, for $n = 2$, G_2 is explicitly given by

$$G_2 = \left[\begin{array}{c|cc|c|cc}
0 & r & \kappa\theta & 0 & -\frac{\rho\sigma v_{\max}v_{\min}}{S} & -\frac{\sigma^2 v_{\max}v_{\min}}{S}\\
\hline
 & 0 & 0 & 2r & \kappa\theta & 0\\
 & -\frac{1}{2} & -\kappa & 1 & r + \frac{\rho\sigma(v_{\max}+v_{\min})}{S} & 2\kappa\theta + \frac{\sigma^2(v_{\max}+v_{\min})}{S}\\
\hline
 & & & 0 & 0 & 0\\
 & & & -1 & -\kappa & 0\\
 & & & 0 & -\frac{1}{2} - \frac{\rho\sigma}{S} & -2\kappa - \frac{\sigma^2}{S}
\end{array}\right]. \tag{2.15}$$

This form highlights the block upper triangular structure of G_n mentioned in Section 2.1. In [4] it is explained how to construct G_n with respect to a different basis. Corollary 2.6 with G_n as in the Jacobi model allows us to compute the moments of X_T in this setting.

2.3 European option pricing via polynomial expansions

We present a polynomial expansion method for European option pricing. This approach has been used in the literature for pricing options in several polynomial models, see e.g. [42] for the Heston model and [4] for the Jacobi model.

For simplicity, we assume that we are pricing a European option with discounted payoff function $f : \mathbb{R} \to \mathbb{R}$. We assume that the asset price is modeled via a polynomial

jump-diffusion (X_t) on the state space $E = \mathbb{R}$. We fix a maturity $T > 0$ and we assume that X_T has a density $q(x)$. The goal is to compute the option price

$$\pi_f = \int_{\mathbb{R}} f(x)q(x)dx. \tag{2.16}$$

We consider an auxiliary density $w(x)$ on $\mathbb{R}$ such that

$$\int_{\mathbb{R}} \frac{q(x)^2}{w(x)}dx < \infty$$

and we denote by $\ell(x)$ the likelihood ratio

$$\ell(x) = \frac{q(x)}{w(x)}.$$

We consider the weighted Lebesgue space,

$$L^2_w = \left\{ f(x) \mid \|f\|^2_w = \int_{\mathbb{R}} f(x)^2 w(x)dx < \infty \right\},$$

equipped with the scalar product

$$\langle f, q\rangle_w = \int_{\mathbb{R}} f(x)q(x)w(x)dx.$$

Note that L^2_w with the scalar product $\langle\cdot\rangle_w$ is a Hilbert space and $\ell(x)$ is in L^2_w. We assume that the space of polynomials $\mathrm{Pol}(\mathbb{R})$ is dense in L^2_w and we fix an orthonormal basis of polynomials $H_0 = 1, H_1, H_2, \ldots$ of L^2_w, such that $\deg H_n(x) = n$ for all $n \in \mathbb{N}$. Also, we further assume that $f(x)$ is in L^2_w, as well. Then, since both $\ell(x)$ and $f(x)$ lie in L^2_w, basic functional analysis (see e.g. [105]) allows us to write the option price (2.16) via the series representation

$$\pi_f = \int_{\mathbb{R}} f(x)q(x)dx = \sum_{n\geq 0} \langle f, H_n\rangle_w \langle \ell, H_n\rangle_w = \sum_{n\geq 0} f_n \ell_n, \tag{2.17}$$

where we have defined

$$\ell_n := \langle \ell, H_n\rangle_w = \int_{\mathbb{R}} H_n(x)q(x)dx$$

and

$$f_n := \langle f, H_n\rangle_w = \int_{\mathbb{R}} f(x)H_n(x)w(x)dx.$$

A price approximation is obtained by truncating the option price series in (2.17), i.e.

$$\pi_f^{(N)} = \sum_{n=0}^{N} f_n \ell_n \tag{2.18}$$

for some positive integer N. Note that (2.18) can be also written as

$$\pi_f^{(N)} = \sum_{n=0}^{N} f_n \ell_n = \sum_{n=0}^{N} \langle f, \ell_n H_n \rangle_w = \int_{\mathbb{R}} f(x) q^{(N)}(x) dx,$$

where

$$q^{(N)}(x) = \left(\sum_{n=0}^{N} \ell_n H_n(x) \right) w(x)$$

is an approximation of the density $q(x)$. In particular, the first N moments of $q^{(N)}(x)$ match the first N moments of $q(x)$, see the second remark below for details.

In the following we provide some remarks.

- Choosing the auxiliary density $w(x)$ is a difficult task. In particular, $w(x)$ needs to be defined such that $f(x)$ and $\ell(x)$ belong to L^2_w, and such that $\mathrm{Pol}(\mathbb{R})$ is dense in L^2_w. These conditions will guarantee the convergence of the series representation (2.17) (see e.g. [105]). In general, the choice of $w(x)$ depends on the underlying model and, more precisely, on the density $q(x)$ of X_T. In [42], the Heston model is considered and $w(x)$ is assumed to be a bilateral gamma density. In the Jacobi model [4], $w(x)$ is defined as a Gaussian density. In [3] the authors propose to use a mixture of auxiliary densities to define $w(x)$, where a class of stochastic volatility models is studied.

- The density approximation $q^{(N)}(x)$ integrates to one, since each $H_n(x)$ is orthogonal to $H_0 = 1$ in L^2_w for any $n \geq 1$. However, in general $q^{(N)}(x)$ is a signed measure, i.e. it can take negative values. Also, we note that for any basis polynomial $H_i(x)$ with $i \leq N$ one has

  $$\int_{\mathbb{R}} H_i(x) q^{(N)}(x) dx = \sum_{n=0}^{N} \ell_n \int H_n(x) H_i(x) w(x) dx = \ell_i,$$

  where in the last equality we use the orthonormality property of the basis elements.

Hence, the moments of order $0, \dots, N$ of $q(x)$ and $q^{(N)}(x)$ coincide.

- From a practical perspective, the efficiency of this approach depends on how fast the coefficients ℓ_n and f_n can be computed. In the setting of polynomial models, the quantities ℓ_n can be computed in closed form via the moment formula (2.7). This is a big computational advantage.

- The coefficients f_n can be computed via numerical integration. However, for some particular cases, depending on the payoff function f and on the choice of $w(x)$, the coefficients f_n are given in closed form, see examples in [4].

- In the case that $w(x)$ admits closed-form moments, i.e. quantities of the form $\int_{\mathbb{R}} p(x)w(x)dx$ are explicitly given for any arbitrary polynomial p, the orthonormal basis $H_0, H_1, H_2, \dots$ can be constructed by applying a Gram-Schmidt orthogonalization procedure to an arbitrary basis of polynomials. In most cases, one can start from the basis of monomials.

- If $w(x)$ is a Gaussian density, the density approximation $q^{(N)}(x)$ coincides with the Gram-Charlier A series expansion of the density $q(x)$ on the real line, see e.g. [77].

- From a practical point view, the option price approximation $\pi_f^{(N)}$ can be computed in two different ways. 1) One computes the quantities ℓ_n and f_n for $n = 0, \dots, N$ and one evaluates the sum $\sum_{n=0}^{N} f_n \ell_n$. 2) One computes ℓ_n and the density approximation $q^{(N)}$ and one approximates $\int_{\mathbb{R}} f(x)q^{(N)}(x)dx$ via numerical integration. If f_n is given in closed form the first approach is to be preferred. Otherwise, the second one is better since it involves only one numerical integration.

- In [39] this polynomial expansion method is generalized to a setting where path dependent options can be priced. We refer to it for more details.

To conclude this section, we would like to emphasize that since all the conditional moments of polynomial (jump-)diffusions are given in closed form, any moment-based technique for option pricing can be used in polynomial models.

3 Polynomial bounds for European and American option pricing

In this chapter we present a methodology for European and American option pricing. The method only assumes the availability of all the moments of the underlying asset price process. Therefore, since the moments of polynomial (jump-)diffusions are given in closed form via the moment formula (2.6), it applies to the specific case of polynomial models. The core idea is as follows. We fix a polynomial degree n. We define an optimization problem in which a linear function is minimized/maximized over the set of all polynomials of degree at most n. The solution of the optimization problem yields an upper/lower bound of the option price. Then, considering an increasing sequence of polynomial degrees yields a monotone sequence of lower/upper bounds of the option price. Under certain assumptions, the sequence converges to the option price. The numerical solution of the involved optimization problems and the convergence study are of particular interest. We address them in this chapter.

The chapter consists of two parts. In the first part, Section 3.1, we introduce the method in the context of European option pricing. In particular, we explain how to define the optimization problems to obtain the bounds of the option price in Section 3.1.1. Then, we study the convergence of the bounds and we explain how to numerically solve the optimization problems, Section 3.1.2 and Section 3.1.3, respectively. We provide numerical results in polynomial models, Section 3.1.4. Finally, we present a black box algorithm for European option pricing, Section 3.1.5. This part is mostly based on the preprint [110].

In the second part, Section 3.2, we adapt the methodology to price American options in polynomial models. We define the sequence of optimization problems in Section 3.2.1.

Then, in Section 3.2.2 we explain how to numerically solve them. Finally, we show numerical experiments in the context of polynomial models in Section 3.2.3.

3.1 European option pricing

We consider the problem of computing the price of European options in the setting where the moments of the underlying asset price process at maturity exist and are available. As introduced in Chapter 1, Section 1.2, this translates to computing a quantity of the form

$$\mathbb{E}[f(\mathbf{X})], \tag{3.1}$$

where f is a scalar function, and $\mathbf{X}$ is a multidimensional random variable, for which we assume the availability of moments.

The core idea of the methodology is to use the moments of $\mathbf{X}$ in order to set up two optimization problems whose solutions yield a lower and an upper bound of (3.1). In particular, for a fixed $n \in \mathbb{N}$, an upper bound of (3.1) can be obtained by minimizing $\mathbb{E}[p(\mathbf{X})]$ over the set of all multivariate polynomials p of total degree at most n that bound f from above, i.e.

$$\inf_p \{\mathbb{E}[p(\mathbf{X})] \mid p \text{ polynomial s.t. } \deg(p) \leq n \text{ and } p(\mathbf{x}) \geq f(\mathbf{x}),\ \forall \mathbf{x} \in E\}, \tag{3.2}$$

where E is the state space of $\mathbf{X}$. Similarly, a lower bound is derived by solving

$$\sup_p \{\mathbb{E}[p(\mathbf{X})] \mid p \text{ polynomial s.t. } \deg(p) \leq n \text{ and } p(\mathbf{x}) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in E\}. \tag{3.3}$$

Solving the problems (3.2) and (3.3) for an increasing sequence of polynomial degrees n yields a monotone sequence of upper and lower bounds. Moreover, writing the moments of $\mathbf{X}$ in a vector γ allows to write the objective function as the linear function $\gamma^\top \vec{p}$, where $\vec{p}$ is the coordinate vector of the polynomial p. In turn, the optimization problems (3.2) and (3.3) are linear semi-infinite programming problems, in the sense that a linear objective function is minimized/maximized subject to an infinite number of constraints. Having the problems (3.2) and (3.3) at hand, two questions arise:

(a) What is the quality of the obtained bounds and what happens as $n \to \infty$?

(b) How do we solve the optimization problems (3.2) and (3.3) numerically?

Partial answers to these questions can be found in the existing literature, where (3.2) and (3.3) have been already studied, together with their associated dual problems. In particular, a very similar approach is considered in [14, 122, 63], in the setting of option pricing as well. There, the underlying assumption is that only a fixed number of moments of $\mathbf{X}$ is available, and the required bounds are computed by considering the same optimization problems as described above. The numerical approach pursued in these works is based on rewriting (3.2) and (3.3) as semidefinite programming problems, whose numerical solutions can be computed via standard algorithms. The dual version of the optimization problems (3.2) and (3.3) is considered in [37, 89], again in the context of option pricing. There, the methodology is extended further to price exotic options, such as Asian or barrier options. Similarly to our setting, the case where all the moments of $\mathbf{X}$ are available is studied, leading to a convergence study of the bounds for $n \to \infty$ for some explicit choices of the payoff function f. In [89], the optimization problem is again solved via a semidefinite programming approach, while in [37] a linear programming approach is used.

In the works mentioned above, however, the function f is always assumed to be piecewise polynomial when considering the problem of pricing European options. This a severe restriction for the application of the methodology. In this chapter, we extend the approach in order to consider settings where f only needs to be upper bounded by a sequence of piecewise polynomials. This generalization is necessary in the settings where the log-asset price is modeled, as for example in the Jacobi and in the Heston model, see Chapter 2, and in other stochastic volatility models, as the Stein-Stein model [111], and the Hull-White model [75]. Indeed, the payoff functions usually become piecewise exponential in the log-asset price formulation, as for example $(1 - e^x)^+$, the payoff function of the European put option with strike price 1. Our extension applies to these settings.

We prove the convergence of the bounds for the cases where X is a scalar random variable, under suitable assumptions on f and on the probability distribution of X. This will give us an answer to the question (a). Concerning question (b), we first propose to use again the semidefinite programming approach developed in [14, 122, 63]. Second, we introduce a new algorithm for the numerical solution of (3.2) and (3.3). This approach, based on the cutting plane technique, is intuitive and can be applied to a vast choice of functions f.

3.1.1 Polynomial bounds via optimization

Fix a probability space $(\Omega, \mathcal{F}, \mathbb{Q})$ together with an E-valued random variable $\mathbf{X}$, where E is a Borel subset of $\mathbb{R}^d$. Let μ be the distribution of $\mathbf{X}$ and assume that its support is given by E. We consider the set $\mathrm{Pol}_n(E)$ of d-variate polynomials of degree at most n ($n \in \mathbb{N}$), as defined in Chapter 2. We assume that all the moments of $\mathbf{X}$ exist, i.e.

$$\mathbb{E}[\mathbf{X}^{\mathbf{k}}] < \infty$$

for any multi-index $\mathbf{k} \in \mathbb{N}_0^d$, and that they are available or can be easily computed. The goal is to find an upper and a lower bound for a quantity of the form

$$\mathbb{E}[f(\mathbf{X})], \tag{3.4}$$

where $f : E \to \mathbb{R}$ is an arbitrary measurable function, which is integrable with respect to μ.

The starting point consists of defining two optimization problems whose solutions represent the desired bounds for (3.4). More precisely, fix a value $n \in \mathbb{N}$ for the considered polynomial degree. Then, the solutions of the optimization problems

$$\mathrm{UB}_n(f) := \inf_{p \in S_n(f)} \mathbb{E}[p(\mathbf{X})], \quad S_n(f) := \{p \in \mathrm{Pol}_n(E) \text{ so that } p(\mathbf{x}) \geq f(\mathbf{x}),\ \forall \mathbf{x} \in E\}, \tag{3.5}$$

$$\mathrm{LB}_n(f) := \sup_{p \in C_n(f)} \mathbb{E}[p(\mathbf{X})], \quad C_n(f) := \{p \in \mathrm{Pol}_n(E) \text{ so that } p(\mathbf{x}) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in E\}. \tag{3.6}$$

satisfy

$$\mathrm{LB}_n(f) \leq \mathbb{E}[f(\mathbf{X})] \leq \mathrm{UB}_n(f).$$

In other words, solving (3.5) and (3.6) corresponds to finding the best upper and lower bounding polynomials of degree at most n of the function f. Note that

$$\begin{aligned} \mathrm{LB}_n(f) &= \sup_{p \in C_n(f)} \mathbb{E}[p(\mathbf{X})] = - \inf_{p \in C_n(f)} \mathbb{E}[-p(\mathbf{X})] \\ &= - \inf_{-p \in S_n(-f)} \mathbb{E}[-p(\mathbf{X})] = - \inf_{p \in S_n(-f)} \mathbb{E}[p(\mathbf{X})] = -\mathrm{UB}_n(-f). \end{aligned} \tag{3.7}$$

The first goal is to study the convergence of the bounds $\mathrm{UB}_n(f)$, $\mathrm{LB}_n(f)$ for $n \to \infty$ under suitable conditions on f and μ. In particular, we aim to prove

$$\lim_{n\to\infty} \mathrm{UB}_n(f) = \mathbb{E}[f(\mathbf{X})], \quad \text{monotonically from above, and} \tag{3.8}$$

$$\lim_{n\to\infty} \mathrm{LB}_n(f) = \mathbb{E}[f(\mathbf{X})], \quad \text{monotonically from below.} \tag{3.9}$$

Due to the relation (3.7), under the requirement that both f and $-f$ satisfy the needed assumptions, proving (3.8) is equivalent to proving (3.9). Therefore, from now on, we restrict our analysis to the minimization problem $\mathrm{UB}_n(f)$.

The study of the convergence requires the formulation of the dual problem of (3.5), which we review in the next chapter.

Duality

In this section we derive the dual problem of the optimization problem (3.5). For fixed chosen polynomial degree n, consider a basis $\mathcal{H}_n := \{h_1, \ldots, h_N\}$ of $\mathrm{Pol}_n(E)$ and write

$$H_n(\mathbf{x}) = (h_1(\mathbf{x}), \cdots, h_N(\mathbf{x})) \in \mathbb{R}^N,$$

in line with the notation introduced in Chapter 2. Consider the vector $\gamma \in \mathbb{R}^N$ of mixed moments of the distribution μ of $\mathbf{X}$ corresponding to $\mathcal{H}$. More precisely, for an enumeration

$$\pi : \mathcal{E} \to \{1, \ldots, N\}$$

of the multi-index set $\mathcal{E} = \{\mathbf{k} \in \mathbb{N}_0^d \mid |\mathbf{k}| \leq n\}$ the entries of γ are defined as

$$\gamma_{\pi(\mathbf{k})} := \int_E h_{\pi(\mathbf{k})}(\mathbf{x}) d\mu.$$

This allows us to rewrite and rename (3.5) as

$$P_n := \inf_{\vec{p}} \{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^N \text{ so that } H_n(\mathbf{x})\vec{p} \geq f(\mathbf{x}), \ \forall \mathbf{x} \in E\}. \tag{3.10}$$

Now, consider an arbitrary finite Borel measure ν on E that satisfies the moment conditions

$$\int_E h_{\pi(\mathbf{k})}(\mathbf{x}) d\nu = \gamma_{\pi(\mathbf{k})}, \quad \text{for all } \mathbf{k} \in \mathcal{E}. \tag{3.11}$$

Then, for all vectors $\vec{p} \in \mathbb{R}^N$ contained in the constraint set of (3.10) we have

$$\int_E f(\mathbf{x}) d\nu \leq \int_E H_n(\mathbf{x}) \vec{p} d\nu = \gamma^\top \vec{p}, \tag{3.12}$$

where the inequality comes from the constraint conditions in (3.10) and the equality from (3.11).

Intuitively, the inequality (3.12) implies that finding a finite Borel measure ν that satisfies the moment conditions (3.11) allows us to find a lower bound for the solution value of (3.10). Following standard results in duality theory, see e.g. [88], we define the dual problem of P_n as finding the largest lower bound of $\gamma^\top \vec{p}$ over the set of all finite Borel measures the satisfy (3.11), i.e. the dual problem of P_n is given by

$$D_n := \sup_\nu \{ \int_E f(\mathbf{x}) d\nu \ \mid \ \nu \in \mathcal{B}(E) \text{ satisfying } \int_E h_{\pi(\mathbf{k})}(\mathbf{x}) d\nu = \gamma_{\pi(\mathbf{k})}, \ \mathbf{k} \in \mathcal{E} \},$$

where $\mathcal{B}(E)$ denotes the set of all finite Borel measures on E.

The inequality (3.12) already shows that *weak duality* between P_n and D_n holds, i.e. $D_n \leq P_n$ whenever the two feasible sets are not empty. In order to find conditions for *strong duality* to hold, i.e. $P_n = D_n$, we write P_n and D_n as conic optimization problems, following the approach from [88, Chapter 1]. Define the convex sets

$$P(E) := \{ (\vec{p}, \tilde{p}_0) \in \mathbb{R}^{N+1} \mid H_n(\mathbf{x}) \vec{p} + \tilde{p}_0 f(\mathbf{x}) \geq 0, \text{ for all } \mathbf{x} \in E \}, \tag{3.13}$$

$$C(E) := \{ (\gamma, \tilde{\gamma}_0) \in \mathbb{R}^{N+1} \mid \exists \nu \in \mathcal{B}(E) \text{ s.t. } \tilde{\gamma}_0 = \int_E f d\nu \text{ and } \int_E h_{\pi(\mathbf{k})}(\mathbf{x}) d\nu = \gamma_{\pi(\mathbf{k})}, \ \mathbf{k} \in \mathcal{E} \}. \tag{3.14}$$

Then, P_n and D_n can be rewritten as

$$P_n = \inf_{\vec{p}} \{ \gamma^\top \vec{p} \ \mid \ (\vec{p}, -1) \in \overline{P(E)} \},$$
$$D_n = \sup_{\tilde{\gamma}_0} \{ \tilde{\gamma}_0 \ \mid \ (\gamma, \tilde{\gamma}_0) \in \overline{C(E)} \}.$$

Finally, standard results of conic duality in convex optimization (see e.g. [18]) yield the following theorem about the strong duality.

Theorem 3.1. *(Theorem 1.2 in [88]). If $(\gamma, \tilde{\gamma}_0) \in C(E)$ for some $\tilde{\gamma}_0$ and there exists $\vec{p} \in \mathbb{R}^N$ such that $(\vec{p}, -1)$ lies in the interior of $P(E)$, then $P_n = D_n$ and both problems*

have an optimal solution, that is, the sup *and the* inf *are attained whenever they are finite.*

3.1.2 Convergence results for the case $d = 1$

In this section we analyze the convergence of the bounds $\mathrm{UB}_n(f)$ for $n \to \infty$, i.e. we prove (3.8). In particular, we give sufficient conditions on μ, E and f for which (3.8) holds. We restrict the analysis to the one-dimensional case $d = 1$. As already mentioned, the convergence of $\mathrm{UB}_n(f)$ and its dual has already been addressed for some cases in the literature, see e.g. [89, 88]. In particular, in [89] the convergence of the dual problems is shown in the framework of European option pricing, where f is assumed to be the payoff function of the European call option, i.e. $f(x) = (x - K)^+$ for a strike price K, and μ is assumed to be moment-determinate, meaning that it is uniquely determined by its moments, see Definition 3.2 below.

All of the convergence results presented so far in the literature assume, however, a polynomial or piecewise polynomial function f. In this work, we are interested in relaxing this condition to consider f of a more general form. The need for a non-piecewise polynomial f arises, for example, when one models the log-asset price instead of the price. Examples are the Jacobi and the Heston stochastic volatility models, introduced in Chapter 2. There, pricing European options often boils down to computing the expectation of piecewise exponential payoff functions, as for example $f(x) = (e^k - e^x)^+$, see Section 3.1.4. In this case, the convergence (3.8) is not guaranteed by existing results.

We start by assuming, without loss of generality, that $E = \mathbb{R}$. The following results can be similarly obtained for any Borel subset E of $\mathbb{R}$. For simplicity, we use the monomial basis for the space $\mathrm{Pol}_n(\mathbb{R})$, i.e.

$$H_n(x) = (1, x, x^2, \cdots, x^n).$$

Note that the following results are, however, independent of the choice of the basis. We assume that $f : \mathbb{R} \to \mathbb{R}_+$ and the distribution μ of X satisfy the following conditions:

C1 There exists a sequence $\{q_m\}_{m\in\mathbb{N}}$ of continuous piecewise polynomials of degree at most m that bound f from above, i.e.

$$f(x) \leq q_m(x) \quad \text{for all } x \in \mathbb{R} \text{ and for every } m.$$

We define the degree of a piecewise polynomial as the highest degree amongst the degrees of the different pieces of polynomials.

C2 The measure μ is moment-determinate. See Definition 3.2 below.

C3 The measure μ and the sequence $\{q_m\}_{m\in\mathbb{N}}$ are defined such that the condition

$$\lim_{m\to\infty} \mathbb{E}[f(X) - q_m(X)] = 0$$

holds.

We define the moment determinacy property in the following (see also [89, Definition 3.2]).

Definition 3.2. *Let μ be measure on $\mathbb{R}^d$ with finite moments of all orders. The measure μ is said to be moment-determinate if $\mu = \nu$ whenever*

$$\int_{\mathbb{R}^d} \mathbf{x}^{\mathbf{k}} d\mu = \int_{\mathbb{R}^d} \mathbf{x}^{\mathbf{k}} d\nu, \quad \textit{for all } \mathbf{k} \in \mathbb{N}_0^d,$$

for some measure ν on $\mathbb{R}^d$.

For each bounding polynomial q_m and for an arbitrary fixed $r \in \mathbb{N}$ we define the optimization problems

$$D_{2r,m} := \sup_{\nu}\{\int_{\mathbb{R}} q_m(x)d\nu \;\mid\; \nu \in \mathcal{B}(\mathbb{R}) \text{ satisfying } \int_{\mathbb{R}} x^i d\nu = \gamma_i,\;\; i = 0, \cdots, 2r\},$$

where γ is the vector of moments of μ, as defined in Section 3.1.1.

We now show that the ad-hoc Conditions C1-C3 are sufficient to obtain the convergence (3.8) of the bounds. We start by showing that $D_{2r,m}$ converges towards $\mathbb{E}[q_m(X)]$ as $r \to \infty$ and for every fixed m. For the particular case $q_m(x) := (x - K)^+$ (for some real K) the convergence is proven in [89, Theorem 5.2]. However, the arguments in the proof apply more generally to piecewise polynomial functions q_m. This yields the following lemma.

Lemma 3.3. *Under the Condition C2, for any fixed m, one has*

$$\lim_{\substack{r\to\infty \\ r\geq m}} D_{2r,m} = \mathbb{E}[q_m(X)],$$

from above, monotonically.

Before proving Lemma 3.3 we review the so-called truncated Stieltjes moment problem (see e.g. [5]), which gives sufficient conditions for an arbitrary vector to represent the vector of moments of a measure supported on an interval of the form $[a, \infty)$, $a \in \mathbb{R}$.

Theorem 3.4 (Truncated Stieltjes moment problem)**.** *For a fixed $r \in \mathbb{N}$, let $y = (y_0, y_1, \ldots, y_{2r}) \in \mathbb{R}^{2r+1}$ be an arbitrary vector. Then, the elements of y are the first $2r+1$ moments of a measure ν supported on $[a, \infty)$ if*

$$M_r(y) \succ 0 \text{ and } M_{r-1}(g, y) \succ 0$$

where $g(x) := x - a$ and $M_r(y), M_{r-1}(g, y)$ are respectively the moment and the localizing matrices, whose entries are defined as

$$\begin{aligned} M_r(y)_{i,j} &:= y_{i+j-2}, \quad i, j = 1, \cdots, r+1, \\ M_{r-1}(g, y)_{i,j} &:= y_{i+j-1} - a y_{i+j-2}, \quad i, j = 1, \cdots, r. \end{aligned}$$

We now prove Lemma 3.3.

Proof. We assume without loss of generality that q_m is of the form

$$q_m(x) = \begin{cases} q_m^1(x) & x < x_0, \\ q_m^2(x) & x \geq x_0, \end{cases} \tag{3.15}$$

for some $x_0 \in \mathbb{R}$ and for two polynomials $q_m^1(x) := \sum_{i=0}^m \alpha_i^1 x^i$ and $q_m^2 := \sum_{i=0}^m \alpha_i^2 x^i$ with coefficients α_i^1 and α_i^2 for $i = 0, \cdots, m$, respectively. For a piecewise polynomial q_m of the more general form (for $n > 0$)

$$q_m(x) = \begin{cases} q_m^1(x) & x < x_0, \\ q_m^{i+2}(x) & x_i \leq x < x_{i+1}, \quad i = 0, \ldots, n-1, \\ q_m^{n+2}(x) & x \geq x_n, \end{cases} \tag{3.16}$$

for some $x_i \in \mathbb{R}$ $(i = 0, \ldots, n)$ and for polynomials $q_m^i(x)$ $(i = 1, \ldots, n+2)$, the same procedure as the following one can be applied by using the truncated Hausdorff moment

problem[1] (see e.g. [5]) together with the truncated Stieltjes moment problem.

We define the restricted measures ν_1 and ν_2 as $\nu_1 = \nu|_{(-\infty,x_0)}$ and $\nu_2 = \nu|_{[x_0,\infty)}$. Then, we write the objective function as

$$\begin{aligned}\int_{\mathbb{R}} q_m(x)d\nu &= \int_{(-\infty,x_0)} q_m^1(x)d\nu_1 + \int_{[x_0,\infty)} q_m^2(x)d\nu_2 \\ &= \sum_{i=0}^{m} \alpha_i^1 \nu_1^i + \sum_{i=0}^{m} \alpha_i^2 \nu_2^i =: L(\nu_1^0, \dots, \nu_1^m, \nu_2^0, \dots, \nu_2^m)\end{aligned}$$

where ν_i^k is the k-th moment of the i-th measure. We rewrite the problem $D_{2r,m}$ as

$$D_{2r,m} := \begin{cases} \sup & L(\nu_1^0, \dots, \nu_1^m, \nu_2^0, \dots, \nu_2^m) \\ \text{subject to} & \nu_1^j + \nu_2^j = \gamma_j, \quad j = 0, \cdots, 2r \\ & \nu_1 \text{ is a Borel measure on } (-\infty, x_0], \\ & \nu_2 \text{ is a Borel measure on } [x_0, \infty). \end{cases}$$

Note that, according to the Remark 3.5 below, we may consider ν_1 to be supported on $(-\infty, x_0]$ instead of $(-\infty, x_0)$. The two associated problems are equivalent. Rewriting the last two conditions according to the truncated Stieltjes moment problem, Theorem 3.4, yields

$$\tilde{D}_{2r,m} := \begin{cases} \sup & L(\nu_1^0, \dots, \nu_1^m, \nu_2^0, \dots, \nu_2^m) \\ \text{subject to} & \nu_1^j + \nu_2^j = \gamma_j, \quad j = 0, \cdots, 2r \\ & M_r(\nu_1) \succeq 0, M_{r-1}(-g, \nu_1) \succeq 0, \\ & M_r(\nu_2) \succeq 0, M_{r-1}(g, \nu_2) \succeq 0, \end{cases}$$

where $g(x) := x - x_0$. The problem $\tilde{D}_{2r,m}$ is a relaxation of $D_{2r,m}$ in the sense that the last two conditions are only necessary conditions for the vectors $(\nu_1^0, \dots, \nu_1^{2r})$ and $(\nu_2^0, \dots, \nu_2^{2r})$ to be moments of measures supported on $(-\infty, x_0]$ and $[x_0, \infty)$, respectively. This is because we are imposing positive semidefiniteness instead of positive definiteness of the moment and localizing matrices. The last step consists of showing

$$\lim_{\substack{r\to\infty \\ r\geq m}} \tilde{D}_{2r,m} = \mathbb{E}[q_m(X)], \tag{3.17}$$

[1]The truncated Hausdorff moment problem gives sufficient conditions for an arbitrary vector to represent the vector of moments of a measure supported on an interval of the form $[a, b]$, for some $a, b \in \mathbb{R}$. In our setting, it can be applied on the intervals $[x_i, x_{i+1}]$, $i = 0, \dots, n-1$, to obtain the relaxed problem $\tilde{D}_{2r,m}$ corresponding to q_m as in (3.16).

from above, monotonically. The proof of (3.17) can be carried out along the lines of [89, Theorem 5.2], which shows (3.17) for the special case $q_m(x) = (x - K)^+$ under the assumption C2. In practice, the proof for $q_m(x)$ as in (3.15) is obtained by replacing K with x_0 and substituting $\nu_2^1 - K\nu_2^0$ with $L(\nu_1^0, \ldots, \nu_1^m, \nu_2^0, \ldots, \nu_2^m)$ in [89, Theorem 5.2]. Finally, (3.17) in turn implies the statement of our lemma since $\tilde{D}_{2r,m} \geq D_{2r,m}$. □

Remark 3.5. *We follow the same reasoning as that of [88, Section 9.1]. Suppose that ϕ_1 supported on $(-\infty, x_0]$, and ϕ_2 supported on $[x_0, \infty)$ are the optimal solution of $D_{2r,m}$. Assume also that $\phi_1(x_0) > 0$. Then we construct a pair of measures $(\tilde{\phi_1}, \tilde{\phi_2})$ such that*

- *$(\tilde{\phi_1}, \tilde{\phi_2})$ is feasible for the problem $D_{2r,m}$,*
- *one has $\tilde{\phi_1}(x_0) = 0$, and*
- *the associated objective value coincides with $L(\phi_1^1, \ldots, \phi_1^m, \phi_2^1, \ldots, \phi_2^m)$, i.e.*

$$L(\tilde{\phi_1}^1, \ldots, \tilde{\phi_1}^m, \tilde{\phi_2}^1, \ldots, \tilde{\phi_2}^m) = L(\phi_1^1, \ldots, \phi_1^m, \phi_2^1, \ldots, \phi_2^m).$$

By constructing such a pair we show that we may assume $(-\infty, x_0]$ instead of $(-\infty, x_0)$.

We write ϕ_1 as $\phi_1 = \phi_{1,1} + \phi_{1,2}$, where $\phi_{1,1}(B) := \phi_1(B \cap \{x_0\})$ and $\phi_{1,2}(B) := \phi_1(B \cap (-\infty, x_0))$ for any Borel set $B \in (-\infty, x_0]$. Then, we define the pair $(\tilde{\phi_1}, \tilde{\phi_2})$ as $(\tilde{\phi_1}, \tilde{\phi_2}) := (\phi_{1,2}, \phi_2 + \phi_{1,1})$. We note that $(\tilde{\phi_1}, \tilde{\phi_2})$ is feasible for $D_{2r,m}$. Moreover, $\tilde{\phi_1}$ is a measure defined on $(-\infty, x_0)$ that satisfies $\tilde{\phi_1}(x_0) = 0$, and $\tilde{\phi_2}$ is a measure defined on $[x_0, \infty)$. Lastly, one has

$$\begin{aligned}
L(\tilde{\phi_1}^1, \ldots, \tilde{\phi_1}^m, \tilde{\phi_2}^1, \ldots, \tilde{\phi_2}^m) &= \int_{(-\infty, x_0)} q_m^1(x) d\tilde{\phi_1} + \int_{[x_0, \infty)} q_m^2(x) \tilde{\phi_2} \\
&= \int_{(-\infty, x_0)} q_m^1(x) d\tilde{\phi_1} + \int_{[x_0, \infty)} q_m^2(x) d\phi_2 + q_m^2(x_0)\phi_{1,1}(\{x_0\}) \\
&= \int_{(-\infty, x_0)} q_m^1(x) d\phi_{1,2} + q_m^1(x_0)\phi_{1,1}(\{x_0\}) + \int_{[x_0, \infty)} q_m^2(x) d\phi_2 \\
&= \int_{(-\infty, x_0]} q_m^1(x) d\phi_1 + \int_{[x_0, \infty)} q_m^2(x) d\phi_2 \\
&= L(\phi_1^1, \ldots, \phi_1^m, \phi_2^1, \ldots, \phi_2^m),
\end{aligned}$$

where in the third equality we use the continuity of $q_m(x)$ at the point x_0.

Consider now the dual problem of $D_{2r,m}$, defined as

$$P_{2r,m} := \inf_{\vec{p}} \{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^{2r+1} \text{ so that } H_{2r}(x)\vec{p} \geq q_m(x), \ \forall x \in \mathbb{R}\}. \tag{3.18}$$

Using the strong duality result reviewed in Section 3.1.1 we show that the sequence $P_{2r,m}$ converges to $\mathbb{E}[q_m(X)]$, as well.

Lemma 3.6. *Under the Conditions C1 and C2 one has*

$$\lim_{\substack{r\to\infty \\ r\geq m}} P_{2r,m} = \mathbb{E}[q_m(X)], \tag{3.19}$$

from above, monotonically.

Proof. We want to show that the assumptions of Theorem 3.1 hold. The statement follows then directly from the strong duality combined with Lemma 3.3.

Condition C1 implies, in particular, that f is polynomially bounded over $\mathbb{R}$ (i.e. there exists a polynomial p satisfying $p(x) \geq f(x)$ for all $x \in \mathbb{R}$). This implies the existence of a $\vec{p}$ such that $(\vec{p}, -1)$ lies in the interior of $P(\mathbb{R})$ (see Definition (3.13)). Also, X is a random variable with probability distribution μ and all finite moments, which implies that $(\gamma, \tilde{\gamma}_0) \in C(\mathbb{R})$ for $\tilde{\gamma}_0 := \int_E f d\mu$ (see Definition (3.14)). Hence, the conditions of Theorem 3.1 are satisfied and the strong duality holds. □

Combining Lemma 3.6 with the Condition C3 yields the main convergence result.

Theorem 3.7. *Assume that Conditions C1, C2 and C3 hold. Then,*

$$\lim_{n\to\infty} \inf_{p\in S_n(f)} \mathbb{E}[p(X) - f(X)] = 0,$$

where $S_n(f) := \{p \in Pol_n(\mathbb{R}) \text{ so that } p(x) \geq f(x), \ \forall x \in \mathbb{R}\}$.

Proof. Fix an arbitrary $\epsilon > 0$. Then, Condition C3 implies that there exists $\tilde{m}$ such that for all $m > \tilde{m}$, one has

$$\mathbb{E}[f(X) - q_m(X)] < \frac{\epsilon}{2}. \tag{3.20}$$

Additionally, (3.19) implies that, for any fixed value m, there exists $\tilde{r} \in \mathbb{N}$ such that for

all $r > \tilde{r}$, we have

$$P_{2r,m} - \mathbb{E}[q_m(X)] < \frac{\epsilon}{2}. \tag{3.21}$$

Both relations (3.20) and (3.21) imply that there exist some finite values $\bar{m}, \bar{r} \in \mathbb{N}$ such that

$$P_{2\bar{r},\bar{m}} - \mathbb{E}[f(X)] < \epsilon.$$

Now, Condition C1 ensures that the polynomial $p^\star_{2\bar{r},\bar{m}}$ defined as the solution argument of $P_{2\bar{r},\bar{m}}$ is in the set $S_{2\bar{r}}(f)$. Therefore, one has

$$\inf_{p \in S_{2\bar{r}}(f)} \mathbb{E}[p(X) - f(X)] \leq \mathbb{E}[p^\star_{2\bar{r},\bar{m}}(X) - f(X)] < \epsilon.$$

Since ϵ was arbitrary chosen, this last relation implies the statement of the theorem. □

In the rest of the section we focus on the special case where E is compact, in particular of the form $[a, b]$ for some real values $a < b$. The convergence result of Theorem 3.7 holds in this case as well and the Condition C2 is automatically satisfied, since all the measures with compact support are moment-determinate, see e.g. [13]. Moreover, if we assume f to be continuous, then the Conditions C1 and C3 directly hold, as well. Indeed, if f is continuous and E is compact, the Weierstrass approximation theorem (see e.g. [116]) implies that it is possible to define a sequence $\{q_m\}_{m \in \mathbb{N}}$ of upper bounding polynomials that satisfies the Condition C3. Even if the convergence is already guaranteed by Theorem 3.7, the following direct argument gives us more insights on the speed of convergence of the bounds $\mathrm{UB}_n(f)$ in terms of the *modulus of continuity of* f, which is defined as

$$\omega(\delta) := \sup_{\substack{y,z \in [a,b], \\ |y-z| < \delta}} |f(y) - f(z)|.$$

This approach does not need the dual formulation of $\mathrm{UB}_n(f)$ and uses standard results from approximation theory. The first lemma we consider is a small modification of Jackson's theorem (see e.g. [47]). We state it in the following.

Lemma 3.8. *Let f be a continuous function on $E := [a, b] \subseteq \mathbb{R}$ for some $a < b$, and let $n \in \mathbb{N}$ be a fixed polynomial degree. Then,*

$$\inf_{p \in S_n(f)} \Big(\sup_{x \in [a,b]} (p(x) - f(x)) \Big) \leq C\omega(1/n),$$

for a constant C which is independent of f and n. Here, $S_n(f)$ is defined as in (3.5) *and ω is the modulus of continuity of f.*

Proof. Jackson's theorem states that for a continuous function f on $[a,b]$, the sequence of *best polynomial approximations* $B_n(f)$ satisfies

$$\sup_{x\in[a,b]} |B_n(f)(x) - f(x)| \leq \tilde{C}\omega(1/n), \tag{3.22}$$

for a constant $\tilde{C}$ which is independent of f and n. Since $B_n(f)$ is a polynomial of degree n for every n, (3.22) implies

$$\inf_{p\in \mathrm{Pol}_n([a,b])} \Big(\sup_{x\in[a,b]} |p(x) - f(x)| \Big) \leq \tilde{C}\omega(1/n).$$

We want to obtain the estimate for the infimum taken over $S_n(f)$. Define $\tilde{B}_n(f) := B_n(f) + \tilde{C}\,\omega(1/n)$. Then, $\tilde{B}_n(f)$ is a polynomial of degree n satisfying $\tilde{B}_n \geq f$ for all $x \in [a,b]$. Hence $\tilde{B}_n(f) \in S_n(f)$ and

$$\sup_{x\in[a,b]} (\tilde{B}_n(f)(x) - f(x)) \leq 2\tilde{C}\omega(1/n),$$

The statement follows by taking the infimum over all polynomials in $S_n(f)$ and defining $C := 2\tilde{C}$. □

We finally propose a convergence result for the case where E is compact and f is continuous, based on the above direct approach.

Theorem 3.9. *Let f be a continuous function on $E := [a,b] \subseteq \mathbb{R}$ for some $a < b$ and let X be an E-valued random variable. Then,* (3.8) *holds.*

Proof. From Lemma 3.8 we obtain

$$\begin{aligned}\inf_{p\in S_n} \mathbb{E}[p(X) - f(X)] &\leq \mu([a,b]) \inf_{p\in S_n} \Big(\sup_{x\in[a,b]} (p(x) - f(x)) \Big) \\ &\leq C\omega(1/n).\end{aligned}$$

Since f is continuous, its modulus of continuity goes to 0 for $n \to \infty$, implying the statement of the theorem. □

Remark 3.10. *The statement of Theorem 3.9 could be proven more directly using Weierstrass approximation theorem, which states that for any $\epsilon > 0$ there exists a polynomial p so that $\sup_{x\in[a,b]} |f(x) - p(x)| \leq \epsilon$. However, the considered proof allows us to have a rough idea of the speed of convergence of the bounds towards $\mathbb{E}[f(X)]$, as $n \to \infty$. Indeed, the inequality*

$$\inf_{p\in S_n} \mathbb{E}[p(X) - f(X)] \leq C\omega(1/n)$$

implies that the convergence rate is given by the modulus of continuity of f. For example, in the case of the European put option with $f(x) = (e^k - e^x)^+$ for some log-strike value k (see also Section 3.1.4), the modulus of continuity is given by

$$\omega(1/n) = \mathcal{O}(1/n).$$

Hence, we would expect the convergence rate to be $1/n$.

3.1.3 Numerical algorithms for the optimization problems

In Section 3.1.1, we have introduced the general optimization problems that allow us to find bounds for the quantity $\mathbb{E}[f(\mathbf{X})]$. Solving them is, however, a difficult task. In this section we present two algorithmic techniques to compute their solution numerically.

Semidefinite programming approach

In this subsection we explain the first strategy to numerically solve (3.5) (and (3.6)) for a fixed polynomial degree n. The idea is to rewrite the optimization problem as a semidefinite programming (SDP) problem, which can be numerically solved via standard algorithms. This approach has already been used in the literature, see e.g. [14, 88]. In the following, we review the main steps of the methodology for an arbitrary dimension d. This approach is developed for solving (3.5) in the cases where f is piecewise polynomial and the state space E can be partitioned in semialgebraic sets (see below for details). Even if we are interested in solving problems for more general forms of f, this method can be applied to numerically solve the problems (3.18), whose solutions tends to $\mathbb{E}[f(X)]$ (for $d = 1$), see Theorem 3.7. It is therefore worth reviewing it.

Consider the problem formulation (3.10). The first step consists of rewriting the constraint

set such that both sides of the inequality are polynomials on suitable specified subsets of E. In particular, define a disjoint partition $\{E_j, j = 1, \cdots, k\}$ of E such that f can be written as

$$f(\mathbf{x}) = \sum_{j=1}^{k} f_j(\mathbf{x}) \mathbb{1}_{E_j}, \ \ \mathbf{x} \in E,$$

for some polynomials $\{f_j, j = 1, \ldots, k\}$. Then, (3.10) can be rewritten as

$$\inf_{\vec{p}} \{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^N \text{ so that } H_n(\mathbf{x})\vec{p} - f_j(\mathbf{x}) \geq 0, \ \ \forall \mathbf{x} \in E_j, \ \ j = 1, \cdots, k\}. \tag{3.23}$$

In order to obtain the SDP formulation of (3.23), we aim to find an equivalent characterization for the non-negativity of the polynomials $H_n(\mathbf{x})\vec{p} - f_j(\mathbf{x})$ on the corresponding sets E_j. In the case that the sets E_j's are semialgebraic, this characterization can be given in terms of sum of squares polynomials, whose definition and properties are reviewed in the following. Note that these results can be mainly found in [88].

Definition 3.11. *A polynomial $p \in Pol(\mathbb{R}^d)$ is a sum of squares (in short s.o.s.) if it can be written as*

$$p(\mathbf{x}) = \sum_{i \in I} p_i(\mathbf{x})^2, \quad \mathbf{x} \in \mathbb{R}^d,$$

for some finite family of polynomials $\{p_i : i \in I\}$, where I is an index set.

Remark 3.12. *Note that the degree of p must be even and the degree of the polynomials p_i is necessarily bounded by half of that of p.*

An equivalent characterization for a polynomial on $\mathbb{R}^d$ to be a sum of squares is given in the following lemma.

Lemma 3.13. *Let $H_n(\mathbf{x})$ be a basis vector of $Pol_n(\mathbb{R}^d)$. Then, a polynomial $p \in Pol_{2n}(\mathbb{R}^d)$ is s.o.s. if and only if there exists a real symmetric positive semidefinite matrix $Q \in \mathbb{R}^{N \times N}$ such that $p(\mathbf{x}) = H_n(\mathbf{x}) Q H_n(\mathbf{x})^\top, \forall \mathbf{x} \in \mathbb{R}^d$.*

Proof. The proof of Proposition 2.1 in [88] shows the statement for the monomial basis vector $B_n(\mathbf{x}) = (1, x_1, \cdots, x_d, x_1^2, x_1 x_2, \cdots, x_{n-1} x_n, x_1^n, \cdots, x_d^n)$. For a general basis vector $H_n(\mathbf{x})$ the statement follows by an appropriate change of basis of the form $H_n(\mathbf{x}) = L B_n(\mathbf{x})$ for some transformation matrix $L \in \mathbb{R}^{N \times N}$. □

Being s.o.s. is clearly a sufficient condition for a polynomial $p \in \mathrm{Pol}(\mathbb{R}^d)$ to be non-negative on $\mathbb{R}^d$. On the other side, the non-negativity property does not necessarily imply that p can be written as a sum of squares. More specifically, for an arbitrary $d > 1$, one can construct a non-negative polynomial which is not s.o.s., see Example 3.15. For the case $d = 1$, instead, being s.o.s. is equivalent to being non-negative, as stated next.

Theorem 3.14 (Theorem 2.5 in [88])**.** *For any polynomial $p \in Pol(\mathbb{R})$ the relation*

$$p(x) \geq 0, \forall x \in \mathbb{R} \iff p(x) \text{ is s.o.s.}$$

holds.

Example 3.15. *Consider the polynomial*

$$p(x_1, x_2, x_3) = x_1^2 x_2^2 (x_1^2 + x_2^2 - x_3^2) + 6x_3^6.$$

Then, p is non-negative on $\mathbb{R}^3$ but it cannot be written as a sum of squares.

Proof. We first show that p is non-negative on $\mathbb{R}^3$. It is clear that p is always positive in the region $\{\mathbf{x} \in \mathbb{R}^3 \mid x_1^2 + x_2^2 \geq x_3^2\}$. Let us consider all values of $\mathbf{x}$ such that $x_1^2 + x_2^2 \leq x_3^2$. Then,

$$x_1^2 + x_2^2 \leq x_3^2 \implies x_1^2 + x_2^2 + x_3^2 \leq 2x_3^2 \implies (x_1^2 + x_2^2 + x_3^2)^3 \leq 8x_3^6,$$

which implies

$$8x_3^6 \geq 6x_1^2 x_2^2 x_3^2.$$

Therefore,

$$p(x_1, x_2, x_3) = x_1^2 x_2^2 (x_1^2 + x_2^2 - x_3^2) + 6x_3^6 \geq x_1^4 x_2^2 + x_1^2 x_2^4 + \frac{14}{3} x_3^6 \geq 0.$$

Hence, p is non-negative. We now show that p cannot be written as a s.o.s. polynomial. Since all the monomials of p are of total degree 6, we attempt to write p in the form

$$\sum_n (A_n x_1^3 + B_n x_1^2 x_2 + C_n x_1^2 x_3 + D_n x_1 x_2^2 + E_n x_1 x_2 x_3 + F_n x_1 x_3^2 + G_n x_2^3 + H_n x_2^2 x_3 + I_n x_2 x_3^2 + J_n x_3^3)^2.$$

Now, since there is no x_1^6 nor x_2^6 term in p, we must have $A_n = G_n = 0$. A similar argument applied to the monomials $x_1^4 x_3^2, x_2^4 x_3^2, x_1^2 x_3^4, x_2^2 x_3^4$ yields $C_n = H_n = F_n = I_n = 0$.

Hence

$$p(x_1, x_2, x_3) = \sum_n (B_n x_1^2 x_2 + D_n x_1 x_2^2 + E_n x_1 x_2 x_3 + J_n x_3^3)^2.$$

The last equation implies that the coefficient of the monomial term $x_1^2 x_2^2 x_3^2$ is given by

$$\sum_n E_n^2 \geq 0,$$

which is a contradiction since the the coefficient of $x_1^2 x_2^2 x_3^2$ in p is -1. Hence, p is not a s.o.s. polynomial. □

As already mentioned, a s.o.s. polynomial is non-negative on $\mathbb{R}^d$. We are now interested in finding sufficient conditions (in terms of s.o.s. polynomials) for a polynomial to be non-negative on an arbitrary semialgebraic set $S \subseteq \mathbb{R}^d$ of the form

$$S = \{\mathbf{x} \in \mathbb{R}^d \mid g_j(\mathbf{x}) \geq 0, j = 1, \cdots, m\}, \tag{3.24}$$

for some polynomials $g_j(\mathbf{x})$.

Lemma 3.16. *Let $S \subseteq \mathbb{R}^d$ be a semialgebraic set of the form* (3.24)*. Then, a polynomial $p \in Pol(\mathbb{R}^d)$ is non-negative on S if it can be written in the form*

$$p(\mathbf{x}) = h_0(\mathbf{x}) + h_1(\mathbf{x})g_1(\mathbf{x}) + \cdots + h_m(\mathbf{x})g_m(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d,$$

for some s.o.s. polynomials $h_j(\mathbf{x}), j = 0, \cdots, m$.

Proof. For simplicity, let us assume $m = 1$. A more general proof will follow directly. We first impose the following inequality on p

$$p(\mathbf{x}) \geq h_1(\mathbf{x})g_1(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d,$$

for a s.o.s. polynomial h_1. This inequality will ensure us that p will be non-negative on S. Next, we just use the fact that a s.o.s. polynomial is non-negative on the whole space $\mathbb{R}^d$. Hence, we impose

$$p(\mathbf{x}) - h_1(\mathbf{x})g_1(\mathbf{x}) = h_0(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d,$$

for a second s.o.s. polynomial h_0. The statement follows. □

For the case $d = 1$, the following two theorems give a characterization in terms of s.o.s. polynomials for p to be non-negative on intervals of the form $[a, b]$, $[a, \infty)$, or $(-\infty, b]$, for some $a < b$.

Theorem 3.17. *(Theorem 2.6 in [88]) Let $p \in Pol_n(\mathbb{R})$. Let $g(x) := (x - a)(b - x)$ for some real values $a < b$. Then $p \geq 0$ on $[a, b]$ if and only if*

$$p(x) = f(x) + g(x)h(x),$$

for some s.o.s. polynomials $f(x)$ and $h(x)$, with both summands of degree less than n.

Theorem 3.18. *(Theorem 2.7 in [88]) Let $p \in Pol(\mathbb{R})$ be non-negative on $[a, \infty)$ for some real value a. Then*

$$p(x) = f(x) + (x - a)h(x),$$

for two s.o.s. polynomials $f(x)$ and $h(x)$. Similarly, if $p \in Pol(\mathbb{R})$ is non-negative on $(-\infty, b]$ for some real value b, then

$$p(x) = g(x) + (b - x)l(x),$$

for two s.o.s. polynomials $g(x)$ and $l(x)$. In both cases the degree of both summands is bounded by the degree of $p(x)$.

Assume now that each set E_j of the partition $\{E_j, j = 1, \cdots, k\}$ of E is a semialgebraic set of the form

$$E_j = \{\mathbf{x} \in \mathbb{R}^d \mid g_i^j(\mathbf{x}) \geq 0, i = 1, \cdots, m^j\}.$$

Moreover, assume that n is even, of the form $n = 2r$ for some $r \in \mathbb{N}$. Finally, by exploiting the previous lemmas and theorems, we rewrite the optimization problem (3.23) as

$$\begin{cases} \inf_{\vec{p} \in \mathbb{R}^N} \gamma^\top \vec{p}, \quad \text{such that} \\ \\ H_n(\mathbf{x})\vec{p} - f_1(\mathbf{x}) = H_r(\mathbf{x})Q_{0,1}H_r(\mathbf{x})^\top + g_1^1 H_r(\mathbf{x})Q_{1,1}H_r(\mathbf{x})^\top + \cdots + g_1^{m^1} H_r(\mathbf{x})Q_{m^1,1}H_r(\mathbf{x})^\top, \\ \qquad\qquad \vdots \\ H_n(\mathbf{x})\vec{p} - f_k(\mathbf{x}) = H_r(\mathbf{x})Q_{0,k}H_r(\mathbf{x})^\top + g_k^1 H_r(\mathbf{x})Q_{1,k}H_r(\mathbf{x})^\top + \cdots + g_k^{m^k} H_r(\mathbf{x})Q_{m^k,k}H_r(\mathbf{x})^\top, \\ Q_{i,j} \succeq 0, \quad \text{for all } i, j. \end{cases} \tag{3.25}$$

This defines an SDP problem. Note that, depending on d and on the specific form of

the sets E_j, the problem (3.25) can be either equivalent to (3.23), or a relaxation of it, since the non-negativity conditions imposed on the polynomials $H_n(\mathbf{x})\vec{p} - f_j(\mathbf{x})$ are only sufficient in some cases, see Lemma 3.16.

In order to numerically solve (3.25), one first needs to rewrite the equality constraints by getting rid of the $\mathbf{x}$ variable. This can be done by comparing the coefficients of all basis elements stored in the basis vectors $H_r(\mathbf{x})$ and $H_n(\mathbf{x})$. In [14] the explicit SDP problem is constructed for some specific forms of f, for instance $f(x) = (x - K)^+$ (payoff of the European call option). The solution of (3.25) can then be numerically computed by standard SDP solvers, such as the interior-point method.

To conclude this part of the first numerical approach, we would like to comment further on the s.o.s. conditions that we impose for replacing the non-negativity conditions in (3.23).

Remark 3.19. *How do the bounds change if we replace the non-negativity conditions by s.o.s. conditions? Theorems 3.17 and 3.18 imply that non-negativity and s.o.s. conditions are equivalent in the case $d = 1$, for E_j of the form $[a, b]$, $[a, \infty)$, or $(-\infty, b]$. For the multivariate case $d > 1$, Theorem 2.4 in [88] states that the space of s.o.s. polynomials is dense in the space of non-negative multivariate polynomials. This implies that, even if the form* (3.25) *is a relaxation of* (3.10) *due to the fact that we impose only some sufficient conditions for non-negativity, we still expect to obtain sharp bounds for $\mathbb{E}[f(\mathbf{X})]$ as $n \to \infty$ when solving* (3.25).

Cutting plane algorithm

The second way we propose to numerically solve (3.5) and (3.6) (for a fixed n) is based on the cutting plane (CP) technique. This algorithm is more direct and intuitive, and does not require us to rewrite the constraint set of the optimization problem. A description of a general cutting plane algorithm can be found, for example, in [17]. Note that an algorithm based on the cutting plane strategy has been used in the context of European option pricing in [122]. There, however, the strategy is applied to an SDP formulation of (3.5) for $d = 1$ and a specific choice of f. Here, we do not use any SDP formulation. Instead, we design the algorithm directly for solving the problem (3.10). In principle, this technique can be applied for any choice of f and d, making it very tractable and suitable for our extension.

The algorithm is iterative and at iteration l we perform the following steps.

1. Define a finite discrete subset $E^l = \{\mathbf{x}^{l,1}, \cdots, \mathbf{x}^{l,m}\}$ of E and impose the inequality constraint in (3.10) only in E^l. We obtain a linear program (LP) of the form

$$\min_{\vec{p}} \{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^N \text{ so that } \mathbf{H}\vec{p} \geq \mathbf{f}\}, \tag{3.26}$$

   where $\mathbf{H} := (H_n(\mathbf{x}^{l,1})|\cdots|H_n(\mathbf{x}^{l,m}))^\top \in \mathbb{R}^{m\times N}$ contains the basis vector $H_n(\mathbf{x})$ evaluated in all points of E^l, and $\mathbf{f} = (f(\mathbf{x}^{l,1}), \cdots, f(\mathbf{x}^{l,m}))^\top$.

2. Solve the LP problem (3.26) using standard techniques. Denote by $p_l^\star$ the resulting optimal coefficient vector.

3. Find the point $\mathbf{x}^v \in E$ defined as $\mathbf{x}^v = \text{argmin}_{\mathbf{x}\in E}(p_l^\star(\mathbf{x}) - f(\mathbf{x}))$ where the inequality constraint is violated the most.

4. If $p_l^\star(\mathbf{x}^v) - f(\mathbf{x}^v) < 0$, then insert $\mathbf{x}^v$ in E^l, defining a new finite discrete subset $E^{l+1} := E^l \cup x^v$ and restart from Step 1 with E^{l+1}. If $p_l^\star(\mathbf{x}^v) - f(\mathbf{x}^v) \geq 0$ stop the iteration and return $\gamma^\top p_l^\star$.

This algorithm is in principle very intuitive and easy to implement. However, some steps require attention. For example, the existence of a solution of (3.26) or of the minimization problem $\text{argmin}_{\mathbf{x}\in E}(p_l^\star(\mathbf{x}) - f(\mathbf{x}))$ is not guaranteed a priori. These points need to be addressed for the specific type of application. In Section 3.1.4 we analyze the algorithm in the context of pricing European put options.

3.1.4 European option pricing in polynomial models

We now apply the methodology developed in the previous sections to price European options in polynomial models. Recalling the risk-neutral pricing setting of Chapter 1 and the polynomial models introduced in Chapter 2, our goal is to evaluate an expression of the form

$$e^{-rT}\mathbb{E}[f(\mathbf{X}_T)],$$

where $(\mathbf{X}_t)$ is a polynomial (jump-)diffusion that models the asset price.

We assume that the initial value $\mathbf{X}_0$ of $(\mathbf{X}_t)$ is deterministic and that $\mathcal{F}_0$ is trivial. Then, formula (2.7) implies that all the mixed moments of $\mathbf{X}_T$ (γ from Section 3.1.1) exist and

can be computed as

$$\gamma^\top = H_n(\mathbf{X}_0)e^{G_n T}. \tag{3.27}$$

In the following, we apply the methodology to the one-dimensional Black-Scholes model and to the Jacobi stochastic volatility model, whose definition and main properties are summarized in Chapter 2, Section 2.2.

European put option - Theoretical setting

We apply the technique developed above to the specific case of the European put option in the log-asset price setting. In particular, consider the payoff function $f(x) = (e^k - e^x)^+$ for a log-strike value k. The goal is to compute, or better, to find polynomial bounds for the quantity

$$e^{-rT}\mathbb{E}[f(X_T)].$$

We first discuss the convergence of the polynomial bounds. Then, we explain how to set up the optimization routines and, finally, we show some numerical results.

Convergence

Showing the convergence (3.8) of the upper bounds boils down for this particular case to showing

$$\lim_{n\to\infty} \inf_{p\in S_n(f)} \mathbb{E}[p(X) - (e^k - e^X)^+] = 0, \tag{3.28}$$

where $S_n(f) := \{p \in \mathrm{Pol}_n(\mathbb{R}) \text{ so that } p(x) \geq (e^k - e^x)^+, \ \forall x \in \mathbb{R}\}$. Note that we omit the discounting factor e^{-rT} and we write X instead of X_T, in line with the notation of Section 3.1.1.

Without loss of generality, we assume $k = 0$. The goal is to apply Theorem 3.7. Hence, we first aim to construct a sequence of piecewise polynomials $\{q_m\}_{m\in\mathbb{N}}$ that satisfy the Condition C1 in Section 3.1.2. First, we consider the Taylor series $T_{2m-1}(x)$ of order $2m-1$ around 0 of the function $1 - e^x$, which are explicitly given by

$$T_{2m-1}(x) = -\sum_{k=1}^{2m-1} \frac{x^k}{k!}. \tag{3.29}$$

Then, we define q_{2m-1} for $m \in \mathbb{N}$ (we consider only odd polynomial degrees) as the

positive part of $T_{2m-1}(x)$, i.e. $q_{2m-1} := T_{2m-1}(x)^+$. The following lemma summarizes some properties of (3.29) and, in particular, shows that the Condition C1 is satisfied for our choice $\{q_{2m-1}\}_{m\in\mathbb{N}}$.

Lemma 3.20. *Let $T_{2m-1}(x)$ be defined as in (3.29) for some $m \in \mathbb{N}$. Then,*

1. *$T_{2m-1}(x) \geq 1 - e^x$, for all $x \in \mathbb{R}$,*
2. *$T_{2m-1}(x)$ has only one real zero, which is given by $x = 0$.*

In particular, it follows that $T_{2m-1}(x)^+ \geq (1 - e^x)^+$ for all $x \in \mathbb{R}$ and every m.

Proof. 1. Expanding the exponential in $1 - e^x$ gives us the expression

$$T_{2m-1}(x) - (1 - e^x) = \sum_{k=2m}^{\infty} \frac{x^k}{k!},$$

which clearly implies $T_{2m-1}(x) \geq 1 - e^x$ for all $x \geq 0$.

To get the inequality on the negative axis, consider again the difference

$$s_{2m-1}(x) := T_{2m-1}(x) - (1 - e^x) = e^x - 1 + T_{2m-1}(x).$$

A simple computation shows that the l-th derivative of $s_{2m-1}(x)$ is given by $s^{(l)}_{2m-1}(x) = e^x - 1 + T_{2m-1-l}(x)$ for $0 < l < 2m - 1$, and $s^{(2m-1)}_{2m-1}(x) = e^x - 1$ which implies $s^{(l)}_{2m-1}(0) = 0$ for $0 < l \leq 2m - 1$. Moreover, $s^{(2m)}_{2m-1}(x) = e^x$. We now show that $s_{2m-1}(x)$ has no real negative zeros, arguing by contradiction. Assume that $s_{2m-1}(x)$ has a real negative zero in some point $x_1 < 0$. Then, since $s_{2m-1}(0) = 0$, Rolle's theorem implies that there exists a point $y_1 \in (x_1, 0)$ where $s^{(1)}_{2m-1}(y_1) = 0$. Now, since $s^{(1)}_{2m-1}(0) = 0$, we can again apply Rolle's theorem to find a point $y_2 \in (y_1, 0)$ so that $s^{(2)}_{2m-1}(y_2) = 0$. Applying inductively the same argument, we get a point $y_{2m} < 0$ satisfying $s^{(2m)}_{2m-1}(y_{2m}) = 0$, which is clearly a contradiction since $s^{(2m)}_{2m-1}(x) = e^x$. Hence, $T_{2m-1}(x)$ doesn't cross $1 - e^x$ on the negative real axis. Moreover, since the leading coefficient of $T_{2m-1}(x)$ is negative, one has

$$\lim_{x\to-\infty} s_m(x) = \infty.$$

It follows that $s_{2m-1}(x) > 0$, and hence, $T_{2m-1}(x) > 1 - e^x$ for $x < 0$.

2. From equation (3.29) one can easily see that 0 is a zero of $T_{2m-1}(x)$. Since all the coefficients of $T_{2m-1}(x)$ are negative, the polynomial can not have strictly positive zeros. Finally, Statement 1 tells us that $T_{2m-1}(x)$ is larger than $1 - e^x$, which is strictly positive on the negative real axis. This implies that $T_{2m-1}(x)$ cannot have negative real zeros. Therefore, 0 is the only real zero of $T_{2m-1}(x)$.

□

We now address the Condition C3. Rather then showing that this condition holds for the chosen polynomial models, we first give some sufficient condition on the distribution μ of X so that it holds.

Lemma 3.21. *Let X be a $\mathbb{R}$-valued random variable whose distribution μ satisfies*

$$\lim_{\substack{l\to\infty \\ l\in\mathbb{N}}} \int_{-\infty}^{0} \frac{x^{2l}}{(2l)!} d\mu = 0. \tag{3.30}$$

Then,

$$\lim_{\substack{m\to\infty \\ m\in\mathbb{N}}} \mathbb{E}[T_{2m-1}(X)^+ - (1 - e^X)^+] = 0.$$

Proof. Property 2 of Lemma 3.20 implies

$$\left|\mathbb{E}[T_{2m-1}(X)^+ - (1 - e^X)^+]\right| = \left|\mathbb{E}[(T_{2m-1}(X) - (1 - e^X))\mathbb{I}_{x\leq 0}]\right|.$$

Then, since all derivatives of $1 - e^x$ are upper bounded by 1 and lower bounded by 0 (in absolute value) on the negative real axis, the Lagrange form of the Taylor remainder $T_{2m-1}(x) - (1 - e^x)$ implies

$$\left|\mathbb{E}[(T_{2m-1}(X) - (1 - e^X))\mathbb{I}_{x\leq 0}]\right| \leq \left|\int_{-\infty}^{0} \frac{x^{2m}}{(2m)!} d\mu\right| = \int_{-\infty}^{0} \frac{x^{2m}}{(2m)!} d\mu,$$

which goes to 0 as $m \to \infty$ thanks to the assumption (3.30). □

Now that we have found conditions on μ such that the Condition C3 is satisfied, we can give explicit sufficient conditions so that the convergence (3.28) holds. The following convergence result comes as a corollary of Theorem 3.7.

Corollary 3.22. *Let X be an $\mathbb{R}$-valued random variable and assume that all of its moments are finite. Assume that the distribution μ of X is moment-determinate and satisfies* (3.30)*. Then,*

$$\lim_{n\to\infty} \inf_{p\in S_n(f)} \mathbb{E}[p(X) - (1 - e^X)^+] = 0,$$

where $S_n(f) := \{p \in Pol_n(\mathbb{R}) \text{ s.t. } p(x) \geq (1 - e^x)^+, \ \forall x \in \mathbb{R}\}$.

We further address the Conditions C2 and C3 in order to relate them to the choice of the asset price models. In the case of the Black-Scholes model, the log-asset price X_t is normally distributed for any time t, as explained in Section 2.2 of Chapter 2. In this case the condition (3.30) is satisfied. Indeed, for an arbitrary even $l = 2k$ (for some positive integer k) and for any $a > 0$ one has

$$\frac{1}{l!}\int_0^\infty x^l e^{-ax^2} dx = \frac{1}{l!}\int_{-\infty}^0 x^l e^{-ax^2} dx = \frac{(2k-1)!!}{2^{k+1}a^k(2k)!}\sqrt{\frac{\pi}{a}} = \frac{1}{2^{k+1}a^k(2k)!!}\sqrt{\frac{\pi}{a}},$$

which clearly converges to 0 as k (or l) tends to infinity. Moreover, as mentioned in [89], the normal distribution is moment-determinate. This implies that the Conditions C1-C3 are satisfied in the case of the Black-Scholes model. Hence, the convergence (3.28) is guaranteed.

In the following, we give a further sufficient condition on X such that (3.30) holds.

Lemma 3.23. *Let X be a $\mathbb{R}$-valued random variable satisfying*

$$\mathbb{E}[e^{|X|}] < \infty. \tag{3.31}$$

Then, condition (3.30) *is satisfied.*

Proof. Property (3.31) together with the dominated convergence theorem allows us to write

$$\sum_{k=0}^{\infty} \frac{\mathbb{E}[|X|^k]}{k!} = \mathbb{E}[e^{|X|}] < \infty,$$

which in turn implies

$$\frac{\mathbb{E}[|X|^k]}{k!} \to 0, \quad \text{as } k \to \infty,$$

and consequently

$$\frac{\mathbb{E}[|X|^{2l}]}{(2l)!} \to 0, \quad \text{as } l \to \infty. \tag{3.32}$$

Finally, the inequalities

$$0 \leq \int_{-\infty}^{0} \frac{x^{2l}}{(2l)!} d\mu \leq \frac{\mathbb{E}[|X|^{2l}]}{(2l)!}$$

together with (3.32) imply the statement of the lemma. □

It turns out that if X satisfies (3.31), then its distribution μ is also moment-determinate, satisfying the Condition C2 as well. Indeed, for any measure μ on $\mathbb{R}$, the so-called Cramèr condition (see e.g. [114]) states that if

$$\mathbb{E}[e^{c|X|}] < \infty \quad \text{for some } c > 0,$$

then μ is moment-determinate. Therefore, if X satisfies (3.31), then (3.28) is guaranteed. Note that for the case when X_T is defined as in the Jacobi model, Theorem 3.1 in [4] can be used to show that X_T satisfies (3.31), depending on the model parameters. Hence, for both chosen models the convergence (3.28) can be shown.

Formulation of the optimization problems

Since the payoff function f is not a piecewise polynomial in this setting, we cannot directly apply the SDP method developed in Section 3.1.3. Instead, we apply it to $T_{2m-1}(x)^+$. The convergence result of the previous subsection guarantees that if we solve the corresponding SDP for $T_{2m-1}(x)^+$ and we let m and n going to infinity, the bounds will still converge towards $\mathbb{E}[f(X)]$.

Fix a polynomial $T_{2m-1}(x)^+$ for some m, and an even natural number $n = 2r$ for the fixed polynomial degree such that $n > (2m-1)$. Then, consider the optimization problem

$$\inf_{\vec{p}} \{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^{2r+1} \text{ so that } H_n(x)\vec{p} \geq T_{2m-1}(x)^+, \ \forall x \in \mathbb{R}\}, \tag{3.33}$$

where γ is computed using the moment formula (3.27) for polynomial models. In particular, note that for the Black-Scholes model, γ is directly given by $\gamma = H_n(X_0)e^{G_n T}$. For the Jacobi model, however, the latter formula returns the vector of mixed moments in both variable x and v. We need only the moments in the x variables. Therefore, before defining

our optimization problem we define γ containing only the entries of $H_n(X_0, V_0)e^{G_n T}$ corresponding to the moments in x. This can be done as explained in the Corollary 2.6, Chapter 2.

In order to apply the SDP technique we rewrite the optimization problem as explained in Section 3.1.3. In particular we derive an SDP of the form (3.25) which is equivalent to (3.33). Consider the equivalence

$$p(x) \geq (T_{2m-1}(x))^+, \ \forall x \in \mathbb{R} \iff p(x) \geq T_{2m-1}(x), \ \forall x \in \mathbb{R} \quad \text{and} \quad p(x) \geq 0, \ \forall x \in \mathbb{R}. \tag{3.34}$$

Then, according to the Theorem 3.14 and the Lemma 3.13 we can rewrite the non-negativity conditions as s.o.s. conditions and we get the SDP formulation

$$\begin{cases} \inf_{\vec{p} \in \mathbb{R}^N} \ \gamma^\top \vec{p}, \quad \text{such that} \\ \\ H_n(x)\vec{p} = H_r(x) Q_{0,1} H_r(x)^\top \\ H_n(x)\vec{p} - T_{2m-1}(x) = H_r(x) Q_{0,2} H_r(x)^\top, \\ Q_{i,j} \succeq 0, \quad \text{for all } i, j. \end{cases} \tag{3.35}$$

This is the problem that is solved in practice, to obtain the required upper bounds for the European put options in the setting of polynomial models.

For the cutting plane approach explained in Section 3.1.3, there is no need to approximate the payoff function because the method does not require a piecewise polynomial payoff function. Hence, we directly solve the problem

$$\mathrm{UB}_n^{CP}(f) := \inf_{\vec{p}} \{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^{2r+1} \text{ so that } H_n(x)\vec{p} \geq (e^k - e^x)^+, \ \forall x \in \mathbb{R}\}. \tag{3.36}$$

However, it is useful to show that all the steps of the algorithm are well defined, under certain conditions. This will allow the CP routine to output the optimal solution when the stopping criterion is satisfied. This particular example of European put option allows us to exploit the structure of the payoff function to make the CP routine computationally more efficient. In particular, we modify the general CP algorithm of Section 3.1.3 by taking care of the following points.

- In the LP problem (3.26) we include the linear condition

$$\gamma^\top \vec{p} \geq 0 \tag{3.37}$$

in the constraint set. Hence, at each iteration l we solve

$$\min_{\vec{p}}\{\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^N \text{ so that } \mathbf{H}\vec{p} \geq \mathbf{f} \text{ and } \gamma^\top \vec{p} \geq 0\}, \tag{3.38}$$

where $\mathbf{H}$ and $\mathbf{f}$ are defined as in (3.26). This will make the LP problem well defined, as we see below in Lemma 3.24. Moreover, note that (3.37) is a reasonable condition since it implies that the obtain upper bounding price is non-negative.

- The minimization problem $\text{argmin}_{x\in E}(p_l^\star(x) - f(x))$ is often seen, as mentioned in [121], as the bottleneck of the CP routine since it might be too expensive. For our case, this can be efficiently performed by splitting the problem on the two regions $E_1 := \{x \in \mathbb{R} \mid x \leq k\}$ and $E_2 := \{x \in \mathbb{R} \mid x \geq k\}$. More precisely we solve

$$\text{argmin}_{x\leq k} \ (p_l^\star(x) - (e^k - e^x)), \tag{3.39}$$

$$\text{argmin}_{x\geq k} \ p_l^\star(x), \tag{3.40}$$

and we add both resulting points to the discrete set E^l. Problem (3.39) is well defined whenever the leading coefficient of $p_l^\star$ is positive and n is even, as we explain below in the Remark 3.25, and can be efficiently solved using, for example, a standard Newton method since the first derivative of the objective function is easily computable. Problem (3.40) can be efficiently solved since the objective function is a polynomial.

The final complete CP algorithm is summarized in Algorithm 3.1. Note that the value *tol* in the algorithm describes a tolerance value that controls the stopping criterion. In a standard CP algorithm this value is set to $tol = 0$. However, in order to make the computation more efficient one can set, for example, $tol = -10^{-5}$, allowing the constraints to be satisfied up to the tolerance *tol*. Also, in Algorithm 3.1 we have indicated the MATLAB functions we can use for solving the involved optimization problems.

In the following we prove the needed results to show that the CP routine is well defined and it produces an optimal solution of (3.36) whenever it stops. Note that we assume that the obtained bounding polynomial $p_l^\star(x)$ has a positive leading coefficient at every

Algorithm 3.1 CP routine for upper bounding European put option prices

Input: Model and payoff parameters, tolerance *tol*, maximal number of iterations maxiter
Output: Upper bound (3.36) of the option price

Construct G_n and γ according to (3.27)
Define a random discrete set E^1 of E
Build $\mathbf{H}$ and $\mathbf{f}$ as defined in (3.26)
$l = 1$
while $l \leq$ maxiter **do**
  Solve the LP (3.38) and get $p_l^\star$ (using `linprog.m`)
  Solve (3.39) and get x_1^v (using `fmincon.m`)
  Solve (3.40) and get x_2^v (using `roots.m`)
  if $p_l^\star(x_1^v) - f(x_1^v) \geq tol$ and $p_l^\star(x_2^v) - f(x_2^v) \geq tol$ **then**
    Break
  end if
  $E^{l+1} = E^l \cup \{x_1^v, x_2^v\}$
  Update $\mathbf{H}$ and $\mathbf{f}$
  $l = l + 1$
end while
$\mathrm{UB}_n^{CP}(f) = \gamma^\top p_l^\star$

step l of the Algorithm 3.1.

Lemma 3.24. *The optimization problem* (3.38) *is well defined.*

Proof. The goal is to show that the function $\gamma^\top \vec{p}$ has a global minimum on the constraint set $A := \{\vec{p} \in \mathbb{R}^N \text{ so that } \mathbf{H}\vec{p} \geq \mathbf{f} \text{ and } \gamma^\top \vec{p} \geq 0\}$. We distinguish two different cases to prove the statement.

If $A \cap \{\vec{p} \in \mathbb{R}^N \text{ so that } \gamma^\top \vec{p} = 0\} \neq \emptyset$, then the global minimum is given by 0.

If $A \cap \{\vec{p} \in \mathbb{R}^N \text{ so that } \gamma^\top \vec{p} = 0\} = \emptyset$, then

$$\lim_{||\vec{p}|| \to \infty, \vec{p} \in A} \gamma^\top \vec{p} = +\infty.$$

Hence, the objective function is continuous, coercive and bounded from below on the closed set A. The existence of a global minimum follows from standard results on coercive functions, see e.g. Theorem 2.32 in [12]. □

Remark 3.25. *In each step of the CP routine we have to solve the minimization problem*

$$argmin_{x \leq k} \; p(x) - (e^k - e^x).$$

The function $p(x) - (e^k - e^x)$ restricted to $(-\infty, k]$ has a global minimum, whenever p is of even degree and the leading coefficient is positive. Therefore, the above minimization problem is well defined under our assumptions.

Lemma 3.26. *The sequence $\{\gamma^\top p_i^\star | i = 1, \cdots\}$ of solution values of the CP steps is bounded and monotone increasing towards the solution value of* (3.36)*. Moreover, suppose that Algorithm 3.1 stops after l iterations. Then, $\gamma^\top p_l^\star$ is the solution value of the original problem* (3.36) *up to a tolerance tol, i.e.*

$$|UB_n^{CP}(f) - \gamma^\top p_l^\star| \leq tol. \tag{3.41}$$

Proof. Let F be the feasible set of (3.36) and let F_l be the feasible set of the LP problem (3.38). Moreover, let $p^\star$ be the optimal solution of (3.36). Then, since we add a new linear constraint step by step in the CP routine, we have

$$F \subseteq \cdots \subseteq F_l \subseteq \cdots \subseteq F_1$$

which implies that $\{\gamma^\top p_i^\star | i = 1, \cdots\}$ is monotone increasing and upper bounded by $\gamma^\top p^\star$.

Using the tolerance value *tol* makes us solving the perturbed problem

$$\inf_{\vec{p}} \{\gamma^\top \vec{p} + tol \;\mid\; \vec{p} \in \mathbb{R}^{2r+1} \text{ so that } H_n(x)\vec{p} \geq (e^k - e^x)^+, \;\; \forall x \in \mathbb{R}\}.$$

This observation directly implies inequality (3.41). □

Remark 3.27. *Since the payoff function is piecewise exponential, we have to use, for example, a Newton type algorithm to perform Step 7 of Algorithm 3.1. Unfortunately, the algorithm can return the argument value of a local minimum, depending on the starting point. However, the numerical results presented in the Section 3.1.4 show that this does not affect the final results for our application. A more practical way to avoid this problem is to use $T_{2m-1}(x)^+$ instead of $(e^k - e^x)^+$, as in the SDP approach. In this case, Step 7 of Algorithm 3.1 could be also performed using a rootfinder (for instance* `roots.m`*) ensuring the argument of the global minimum on the region $x \leq k$.*

European put option - Numerical experiments

In this section, we finally show some numerical results for the computation of European put option prices in the one-dimensional Black-Scholes model and in the Jacobi model. All algorithms have been implemented in MATLAB and run on a standard laptop (Intel Core i7, 2 cores, 256kB/4MB L2/L3 cache). We used the toolbox YALMIP [93] together with SDPT3 [117] to model and solve the SDP problems, while the LP problems and the minimization problems arising in the cutting plane algorithm have been solved via MATLAB's built-in functions `linprog.m` (for LP), `fmincon.m` (for minimizing non-polynomial functions) and `roots.m` (for finding the critical points of polynomials and, consequently, their minima).

For both models, the payoff parameters and the initial log-asset price are set to

$$x_0 = 0, \quad T = 1, \quad k = \{-0.1, 0, 0.1\},$$

so that we consider different types of moneyness. For the model parameters, we set

$$\sigma = 0.2, \quad r = 0.01$$

for the Black-Scholes model, while for the Jacobi model we consider the model parameters

$$v_0 = 0.04, \;\; \sigma = 0.15, \;\; \kappa = 0.5, \;\; \theta = 0.04, \;\; \rho = -0.5, \;\; v_{\min} = 10^{-4}, \;\; v_{\max} = 0.1, \;\; r = 0.01.$$

In these numerical experiments we also consider the computation of lower bounds in order to better assess the method. For the numerical computation of upper bounds, the optimization algorithms are set up as described in the previous section, while for the lower bounds the algorithms have been adapted to the new setting. Note that when considering the computation of lower bounds, the corresponding SDP formulation cannot be simplified as in the case of the upper bounds (see the formulation (3.35)) since the relation (3.34) cannot be used. The obtained SDP problem is therefore slightly more complicated and involves more variables. Also, when setting up the corresponding CP algorithm the condition (3.37) cannot be included.

The parameters of the optimization algorithms are set as follows. For the CP algorithm (Algorithm 3.1), the tolerance value is set to $tol = -10^{-5}$ and the initial discrete set E^1 consists of 200 equidistant points in $[-2, 2]$. For the SDP approach we solve the

optimization problems for the sequence of polynomials $T_{n-1}(x)^+$ (see (3.29)), where n is the considered polynomial degree in the optimization problem. The vector of moments γ is computed according to the moment formula (3.27), where the matrix G_n is constructed for both models as explained in Chapter 2, Section 2.2.

We compare the obtained bounds with reference prices computed via other techniques. In the Black-Scholes model we compute the reference price using the Black-Scholes formula, see Chapter 1. For the Jacobi model the reference price is computed using the polynomial expansion method developed in [4], and summarized in Chapter 2, Section 2.2. In particular, the auxiliary density $w(x)$ is defined as a Gaussian density with mean $\mu_w = \mathbb{E}[X_T]$ and variance $\sigma_w^2 = \mathbb{E}[X_T^2] - \mathbb{E}[X_T]^2$, and the orthonormal basis consists of generalized Hermite polynomials, defined as

$$H_n(x) = \frac{1}{\sqrt{n!}} \mathcal{H}_n\left(\frac{x-\mu_w}{\sigma_w}\right), \tag{3.42}$$

where $\mathcal{H}_n(x)$ are the standard probabilists' Hermite polynomials, i.e.

$$\mathcal{H}_n(x) = (-1)^n \mathrm{e}^{\frac{x^2}{2}} \frac{d^n}{dx^n} \mathrm{e}^{-\frac{x^2}{2}}.$$

In the case of the European call option, the quantities f_n in the expansion (2.17) are given recursively by

$$\begin{aligned} f_0 &= \mathrm{e}^{-rT+\mu_w} I_0\left(\frac{k-\mu_w}{\sigma_w}; \sigma_w\right) - \mathrm{e}^{-rT+k} \Phi\left(\frac{\mu_w - k}{\sigma_w}\right), \\ f_n &= \mathrm{e}^{-rT+\mu_w} \frac{1}{\sqrt{n!}} \sigma_w I_{n-1}\left(\frac{k-\mu_w}{\sigma_w}; \sigma_w\right), \quad n \geq 1, \end{aligned} \tag{3.43}$$

where the functions $I_n(\mu;\nu)$ are defined recursively by

$$\begin{aligned} I_0(\mu;\nu) &= \mathrm{e}^{\frac{\nu^2}{2}} \Phi(\nu - \mu), \\ I_n(\mu;\nu) &= \mathcal{H}_{n-1}(\mu) \mathrm{e}^{\nu\mu} \phi(\mu) + \nu I_{n-1}(\mu;\nu), \quad n \geq 1. \end{aligned}$$

Here, $\Phi(x)$ denotes the standard Gaussian distribution function, and $\phi(x)$ its density. In order to obtain reference prices, we first compute call option prices by truncating the expansion at $N = 50$ (see (2.18)). Then, put prices are obtained via the put-call parity (1.3), see Chapter 1.

In the following analysis we also consider the mid-price $MP_n(f)$, defined as $MP_n(f) := \frac{UB_n(f)+LB_n(f)}{2}$. The mid-price can be used as an approximation of the option price and, since the bounds converge towards the exact price as n goes to infinity, it satisfies $\lim_{n\to\infty} MP_n(f) = \mathbb{E}[f(X_T)]$. The quality of the mid-price is of practical relevance and we consider it in the construction of a black box algorithm for European option pricing, which we introduce later in the Section 3.1.5. In Table 3.1, for each value of k and for different values of n, we show the obtained lower bounds (LB), the upper bounds (UB), and (in blue) the absolute error of the mid-price $MP_n(f)$ with respect to the reference price. We show the results for both the SDP and the CP approach, in the Black-Scholes model. Similarly in Table 3.2 for the Jacobi model. In Figure 3.2 we show the obtained bounding polynomials of degree $n = 12$ in the Jacobi model, while in Figure 3.1 we show the bounding polynomials obtained in the Black-Scholes model for $n = 12$, multiplied by the probability density function of X_T which is explicitly known in this model[2]. This will give us more insight on the quality of the bounds with respect to the correct integrating measure. Finally, the Figures 3.3 and 3.4 show the Black-Scholes implied volatilities[3] corresponding to the upper bounds, to the lower bounds and to the mid-price, for the Black-Scholes model and the Jacobi model, respectively.

First, we observe that both methods yield tight upper and lower bounds for both models, in line with the numerical results obtained also in [14, 89]. The numerical values in the Tables 3.1-3.2 show in particular that the upper bounds have a slightly better quality than the lower ones, which can be also observed from the implied volatilities pictured in the Figures 3.3-3.4. For $n = 20$, the absolute error of the mid-price is of magnitude 10^{-3} for both models and any type of moneyness. Also, the absolute implied volatility error can be observed to be much lower than 1%. Therefore, considering moments of order 20 already yields a good price approximation.

Regarding the comparison between the two optimization approaches, it can be observed that the CP algorithm performs better for small polynomial degrees ($n = 2, \cdots, 8$), while for higher degrees the two approaches produce almost identical results. This is due to the fact that the SDP approach requires the polynomial approximation of the payoff function, which is poor for n small, while the CP algorithm is directly applied to the

[2]We have plotted the bounding polynomials for $n = 12$, which is the polynomial degree for which the results of the SDP and the CP algorithm start to coincide for both models.

[3]For a given put option price C, the Black-Scholes implied volatility is defined as the volatility parameter that renders the corresponding Black-Scholes price equal to C.

	SDP		
	$k = -0.1$	$k = 0$	$k = 0.1$
n	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]
2	[0.0000, 0.0056, 0.0579]	[0.0099, 0.0174, 0.1041]	[0.1204, 0.0139, 0.1851]
4	[0.0141, 0.0028, 0.0493]	[0.0526, 0.0010, 0.0942]	[0.1121, 0.0052, 0.1550]
6	[0.0202, 0.0029, 0.0429]	[0.0527, 0.0064, 0.0833]	[0.1225, 0.0028, 0.1495]
8	[0.0224, 0.0028, 0.0410]	[0.0614, 0.0021, 0.0833]	[0.1238, 0.0039, 0.1459]
10	[0.0255, 0.0018, 0.0399]	[0.0614, 0.0036, 0.0802]	[0.1284, 0.0019, 0.1453]
12	[0.0260, 0.0022, 0.0387]	[0.0651, 0.0018, 0.0801]	[0.1285, 0.0027, 0.1435]
14	[0.0279, 0.0013, 0.0384]	[0.0654, 0.0024, 0.0786]	[0.1311, 0.0015, 0.1434]
16	[0.0280, 0.0017, 0.0377]	[0.0673, 0.0014, 0.0786]	[0.1312, 0.0020, 0.1424]
18	[0.0292, 0.0011, 0.0376]	[0.0676, 0.0017, 0.0778]	[0.1326, 0.0013, 0.1424]
20	[0.0293, 0.0013, 0.0370]	[0.0682, 0.0014, 0.0777]	[0.1327, 0.0016, 0.1416]

	CP		
	$k = -0.1$	$k = 0$	$k = 0.1$
n	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]
2	[0.0000, 0.0083, 0.0524]	[0.0084, 0.0228, 0.0947]	[0.0959, 0.0078, 0.1659]
4	[0.0146, 0.0026, 0.0493]	[0.0530, 0.0008, 0.0941]	[0.1128, 0.0050, 0.1547]
6	[0.0211, 0.0026, 0.0428]	[0.0532, 0.0064, 0.0828]	[0.1238, 0.0023, 0.1492]
8	[0.0227, 0.0030, 0.0403]	[0.0619, 0.0020, 0.0828]	[0.1248, 0.0035, 0.1457]
10	[0.0260, 0.0016, 0.0399]	[0.0619, 0.0034, 0.0801]	[0.1285, 0.0019, 0.1453]
12	[0.0260, 0.0022, 0.0387]	[0.0651, 0.0019, 0.0800]	[0.1287, 0.0027, 0.1435]
14	[0.0279, 0.0014, 0.0384]	[0.0653, 0.0024, 0.0786]	[0.1311, 0.0015, 0.1434]
16	[0.0281, 0.0017, 0.0376]	[0.0672, 0.0015, 0.0785]	[0.1313, 0.0019, 0.1424]
18	[0.0293, 0.0011, 0.0376]	[0.0674, 0.0018, 0.0777]	[0.1326, 0.0012, 0.1424]
20	[0.0294, 0.0013, 0.0370]	[0.0685, 0.0013, 0.0777]	[0.1328, 0.0016, 0.1417]

Table 3.1 – Lower/upper bounds and mid-price absolute error for put option prices obtained via SDP and CP approach in the Black-Scholes model. The reference prices are 0.0345 *($k = -0.1$),* 0.0744 *($k = 0$) and* 0.1388 *($k = 0.1$).*

piecewise exponential payoff function. As n increases, the quality of the approximation $T_{n-1}(x)^+$ of $f(x)$ improves, and the SDP approach reaches the same accuracy as the CP one. Therefore, for lower polynomial degree n, the CP method is to be preferred, while for n sufficiently large both methods can be in principle considered.

Finally, the Figures 3.1-3.2 give us an idea on how the bounding polynomials look like. Intuitively, the bounding polynomials are expected to be tighter around the mean $\mathbb{E}[X_T]$,

	SDP		
	$k = -0.1$	$k = 0$	$k = 0.1$
n	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]
2	[0.0000, 0.0062, 0.0587]	[0.0099, 0.0161, 0.1051]	[0.1204, 0.0171, 0.1860]
4	[0.0165, 0.0013, 0.0521]	[0.0502, 0.0019, 0.0932]	[0.1061, 0.0075, 0.1511]
6	[0.0190, 0.0043, 0.0435]	[0.0523, 0.0054, 0.0840]	[0.1199, 0.0018, 0.1487]
8	[0.0242, 0.0018, 0.0434]	[0.0583, 0.0033, 0.0822]	[0.1199, 0.0044, 0.1435]
10	[0.0247, 0.0029, 0.0408]	[0.0609, 0.0028, 0.0807]	[0.1243, 0.0022, 0.1434]
12	[0.0274, 0.0015, 0.0408]	[0.0621, 0.0029, 0.0792]	[0.1255, 0.0025, 0.1417]
14	[0.0275, 0.0021, 0.0395]	[0.0643, 0.0019, 0.0790]	[0.1265, 0.0023, 0.1411]
16	[0.0291, 0.0013, 0.0395]	[0.0644, 0.0024, 0.0779]	[0.1282, 0.0016, 0.1407]
18	[0.0292, 0.0016, 0.0388]	[0.0663, 0.0015, 0.0779]	[0.1283, 0.0019, 0.1400]
20	[0.0301, 0.0012, 0.0388]	[0.0663, 0.0018, 0.0772]	[0.1293, 0.0015, 0.1399]

	CP		
	$k = -0.1$	$k = 0$	$k = 0.1$
n	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]	[LB, Abs Err MP, UB]
2	[0.0000, 0.0091, 0.0531]	[0.0025, 0.0246, 0.0955]	[0.0956, 0.0050, 0.1667]
4	[0.0165, 0.0013, 0.0521]	[0.0505, 0.0018, 0.0930]	[0.1064, 0.0075, 0.1507]
6	[0.0194, 0.0041, 0.0435]	[0.0524, 0.0054, 0.0840]	[0.1200, 0.0018, 0.1486]
8	[0.0245, 0.0017, 0.0434]	[0.0589, 0.0030, 0.0822]	[0.1201, 0.0043, 0.1435]
10	[0.0246, 0.0028, 0.0408]	[0.0609, 0.0020, 0.0822]	[0.1251, 0.0018, 0.1435]
12	[0.0274, 0.0016, 0.0406]	[0.0621, 0.0029, 0.0792]	[0.1255, 0.0025, 0.1417]
14	[0.0276, 0.0020, 0.0395]	[0.0643, 0.0019, 0.0790]	[0.1265, 0.0023, 0.1410]
16	[0.0291, 0.0013, 0.0395]	[0.0646, 0.0023, 0.0779]	[0.1281, 0.0017, 0.1407]
18	[0.0293, 0.0016, 0.0388]	[0.0661, 0.0017, 0.0778]	[0.1283, 0.0019, 0.1399]
20	[0.0301, 0.0011, 0.0388]	[0.0661, 0.0019, 0.0772]	[0.1294, 0.0015, 0.1398]

Table 3.2 – Lower/upper bounds and mid-price absolute error for put option prices obtained via SDP and CP approach in the Jacobi model. The reference prices are 0.0356 *($k = -0.1$),* 0.0736 *($k = 0$) and* 0.1361 *($k = 0.1$).*

and then explode far away from that region. This is because the probability mass is concentrate around the mean. This intuition is confirmed by the polynomials shown in the Figure 3.2 where the Jacobi model is considered. When multiplying the bounding polynomial with the probability density function, as we did in the Figure 3.1 for the Black-Scholes case, one can first observe that the explosion of the polynomials is compensated by the faster decay of the density function. The polynomials are therefore tight over the whole real axis in the "correct measure". Second, the same figure shows that the region

Algorithm 3.2 European option pricing in polynomial models based on optimization

Input: Model and payoff parameters, tolerance ϵ
Output: Approximation $Price$ of $\mathbb{E}[f(\mathbf{X})]$

$n = 2$
$Gap = 1$
while $Gap > \epsilon$ **do**
    Set up the optimization problems (3.5) and (3.6)
    Solve them numerically to get a lower bound LB and an upper bound UB
    $Gap = \text{UB} - \text{LB}$
    $n = n + 2$
end while
$Price = (\text{UB} + \text{LB})/2$

where the polynomials cannot completely capture the shape of the payoff function is the one around the "kink", i.e. the irregularity of the payoff function given by the strike value k. Also, one can see that the irregular region can be better approximated from above than from below, generating the quality gap between upper and lower bounds.

3.1.5 A black box algorithm for European option pricing

In real-world applications, one usually needs option pricing algorithms that are able to output an option price, given some model and payoff parameters as input. In other words, the result of running the pricing algorithm must be given by a single number: the price. The technique described in this chapter, as well as the approaches developed in [14, 122, 63, 89, 37], is able to compute a lower bound and an upper bound for the option price of interest. However, it does not give any a priori information on the quality of the bounds or on the price itself. Moreover, even if in theory it can be shown that the bounds converge to the real price as $n \to \infty$, it is not known a priori how to choose the value of n required to get a satisfactory accuracy.

In order to overcome these obstacles, we can design an algorithm which computes the bounds incrementally (by increasing n) and stops after that the gap between them goes below a certain tolerance ϵ. Such a black box algorithm would then require model and payoff parameters as input, and would return a ϵ approximation of the exact price in output, as desired. Assuming that both upper and lower bounds converge to the exact price, such an algorithm can be defined as in Algorithm 3.2 (again, we omit the discounting factor for simplicity).

The Step 5 of Algorithm 3.2 can be performed by employing one of two numerical techniques developed in Section 3.1.3. In order to perform Step 4 and efficiently update the vector γ of moments in the setting of polynomial models, one can use the algorithm `incexpm.m` developed in [83]. This algorithm exploits the block triangular structure of the nested sequence $G_0, G_1, \cdots$ of matrices in the moment formula (3.27) in order to incrementally update γ in an efficient way, as n increases. This particular structure has been already presented in Chapter 2, Section 2.1, and the algorithm `incexpm.m` is also developed in this thesis. We present it in the next chapter.

3.2 American option pricing

Considering the dual problem D_n allows us to extend the methodology to price exotic options, as Asian and barrier options, see [89, 37]. On the other side, the primal optimization problem (3.5) can be modified in order to obtain an optimization problem whose solution yields an upper bound for the American version of the same option. We now explain how to do it.

3.2.1 Polynomial upper bounds via optimization

We consider the problem of computing the price of an American option maturing at time T, with payoff function $f : E \to \mathbb{R}$. The price at time $t = 0$ is given by (see Section 1)

$$\sup_{\tau \in \mathcal{S}_{0,T}} \mathbb{E}[e^{-r\tau} f(S_\tau)], \tag{3.44}$$

where $\mathcal{S}_{0,T}$ is the set of all stopping times in $[0, T]$. In the following, the methodology is developed for polynomial models. In particular, we assume that all the conditional moments of $(\mathbf{X}_t)$ exist and are available.

The core idea is again to consider an optimization problem whose solution represents an upper bound of the price (3.44). In the American case we consider only upper bounds. It

is shown in [25], that the price (3.44) is upper bounded by the solution of the problem

$$\inf\{e^{-rT}\mathbb{E}[s(\mathbf{X}_T)] \mid s \text{ measurable s.t. } e^{-r(T-t)}\mathbb{E}_{\mathbf{x}}[s(\mathbf{X}_T)] \geq f(\mathbf{x}),\ \forall(\mathbf{x},t) \in E \times [0,T]\},$$

(3.45)

where $\mathbb{E}_{\mathbf{x}}$ denotes the expectation assuming the starting value $\mathbf{X}_t = \mathbf{x}$ at time t. Intuitively, the idea is to derive a payoff function s whose corresponding European option price $e^{-r(T-t)}\mathbb{E}_{\mathbf{x}}[s(\mathbf{X}_T)]$ dominates f, at any time $t \in [0,T]$ and for any $\mathbf{x} \in E$. Then, the European price $e^{-rT}\mathbb{E}[s(\mathbf{X}_T)]$ at time $t = 0$ is an upper bound of the American price (3.44). In other words, we represent the American option in (3.44) with a dominating European option. In [91], the solution s of (3.45) is referred to as the *cheapest dominating European option* of the American option with payoff f.

Results in [25, 91] show that (3.45) yields sharp bounds for (3.44). We aim at adapting this approach to the setting of polynomial models. To do that, we propose to restrict the feasible set of (3.45) to the set $\text{Pol}_n(E)$. This yields the problem

$$\inf\{e^{-rT}\mathbb{E}[p(\mathbf{X}_T)] \mid p \in \text{Pol}_n(E) \text{ s.t. } e^{-r(T-t)}\mathbb{E}_{\mathbf{x}}[p(\mathbf{X}_T)] \geq f(\mathbf{x}),\ \forall(\mathbf{x},t) \in E \times [0,T]\},$$

which can be rewritten as

$$\inf_{\vec{p}}\{e^{-rT}\gamma^\top\vec{p} \mid \vec{p} \in \mathbb{R}^N \text{ s.t. } e^{-r(T-t)}H_n(\mathbf{x})e^{G_n(T-t)}\vec{p} \geq f(\mathbf{x}),\ \forall(\mathbf{x},t) \in E \times [0,T]\},$$

(3.46)

after inserting the moment formula (2.6) and

$$\gamma^\top = H_n(\mathbf{X}_0)e^{G_nT}.$$

Solving (3.46) will yield the desired upper bound of (3.44). We note that if we impose the condition $e^{-r(T-t)}H_n(\mathbf{x})e^{G_n(T-t)}\vec{p} \geq f(\mathbf{x})$ only at finite time $t = T$, the problem (3.46) reduces to (3.10), the upper bound of the European version of the option. This gives us more insights on how the American case can be seen as an extension of the European one.

As in the European case, we propose two ways to solve (3.46): a semidefinite programming approach and a cutting plane algorithm. Both are extensions of the algorithms presented in Section 3.1.3.

3.2.2 Numerical algorithms for the optimization problems

SDP approach for American options

We assume that f is piecewise polynomial and E can be partitioned in semialgebraic sets, as in the European case. In order to derive an SDP formulation of (3.46) for a fixed polynomial degree n, we again aim at rewriting the constraint set. Since the function $e^{-r(T-t)}H_n(\mathbf{x})e^{G_n(T-t)}\vec{p} - f(\mathbf{x})$ is not polynomial nor piecewise polynomial in the time variable t, we cannot directly substitute the non-negativity conditions with s.o.s. conditions. In order to overcome this obstacle, we propose to discretize the time interval $[0,T]$ and impose the inequality constraints at each time discretization point. More precisely, we consider a discretization $\{t_0 = 0, t_1, \cdots, t_{\ell-1}, t_\ell = T\}$ of $[0,T]$ and we look at the optimization problem

$$\begin{cases} \inf_{\vec{p}\in\mathbb{R}^N} \quad e^{-rT}\gamma^\top\vec{p}, \quad \text{s. t.} \\ \qquad\qquad e^{-r(T-t_i)}H_n(\mathbf{x})e^{G_n(T-t_i)}\vec{p} \geq f(\mathbf{x}), \ \ \forall \mathbf{x}\in E \text{ and for } i = 0,\cdots,\ell. \end{cases} \tag{3.47}$$

Now, each of the $\ell+1$ inequalities in the constraint set can be treated as in the European case (see Section 3.1.3). Hence, we impose s.o.s. conditions instead of non-negativity conditions for each inequality constraint by splitting E in a suitable way. The resulting problem is an SDP problem.

The obtained problem is more complex and the number of s.o.s constraints is larger, depending on the number ℓ of discretization points. However, we will see in Chapter 3.2.3 that the problem remains numerically solvable for the chosen applications and the obtained upper bounds are sharp. Also, the fact that we discretize the time interval does not influence the quality of the bounds.

CP algorithm for American options

The cutting plane algorithm does not require the infinite linear constraints to be polynomial nor piecewise polynomial in any variable. Consequently, the proposed algorithm for the European case described in Section 3.1.3 can be easily adapted to the American case. More precisely, at iteration l we perform the following steps.

1. Define a finite discrete subset $\tilde{E}^l = \{(\mathbf{x}^{l,1}, t^{l,1}), \cdots, (\mathbf{x}^{l,m}, t^{l,m})\}$ of $E \times [0,T]$ and

impose the inequality constraint in (3.46) only in $\tilde{E}^l$, obtaining the LP problem

$$\min\{e^{-rT}\gamma^\top \vec{p} \mid \vec{p} \in \mathbb{R}^N \text{ so that } \mathbf{H}\vec{p} \geq \mathbf{f}\}, \tag{3.48}$$

where $\mathbf{H} := (e^{-r(T-t^{l,1})}H_n(\mathbf{x}^{l,1})e^{G_n(T-t^{l,1})}|\cdots|e^{-r(T-t^{l,m})}H_n(\mathbf{x}^{l,m})e^{G_n(T-t^{l,m})})^\top \in \mathbb{R}^{m\times N}$ contains the left hand-side of the inequality constraints evaluated in $\tilde{E}^l$, and $\mathbf{f} = (f(\mathbf{x}^{l,1}), \cdots, f(\mathbf{x}^{l,m}))^\top$.

2. Solve the LP problem (3.48) using standard techniques. Denote by $p_l^\star$ the resulting optimal coefficient vector.

3. Find the point $(\mathbf{x}^v, t^v) \in E \times [0,T]$ defined as

$$\text{argmin}_{(\mathbf{x},t)\in E\times[0,T]}(e^{-r(T-t)}H_n(\mathbf{x})e^{G_n(T-t)}p_l^\star - f(\mathbf{x})),$$

where the inequality constraint is violated the most.

4. If $e^{-r(T-t^v)}H_n(\mathbf{x}^v)e^{G_n(T-t^v)}p_l^\star - f(\mathbf{x}^v) < 0$, then insert $(\mathbf{x}^v, t^v)$ in $\tilde{E}^l$, defining a new finite discrete subset $\tilde{E}^{l+1} := \tilde{E}^l \cup (\mathbf{x}^v, t^v)$ and restart from Step 1 with $\tilde{E}^{l+1}$. If $e^{-r(T-t^v)}H_n(\mathbf{x}^v)e^{G_n(T-t^v)}p_l^\star - f(\mathbf{x}^v) \geq 0$ stop the iteration and return $e^{-rT}\gamma^\top p_l^\star$.

Compared to the European case, the CP algorithm does not increase much in complexity. For example, the dimension of the LP problem solved at Step 1 remains unchanged. Only in Step 3 the numerical solution of a more complex problem is required. Here, the minimization problem is now to be performed over a set of dimension $d+1$ (instead of d). Again, in this step we can use standard minimization solvers on smartly split regions to make the algorithm more efficient.

3.2.3 American option pricing in polynomial models

We apply the methodology developed in the Sections 3.2.1 and 3.2.2 to price American put options and American rainbow options. We explain how to set up the optimization problems and we show numerical experiments for concrete asset models. Note that the following numerical experiments have been performed on the same machine as in the European case. Also, the same MATLAB toolboxes and built-in functions have been used.

American put option

In this section we price American put options in the one-dimensional Black-Scholes model. As in the European case, Section 3.1.4, we use the log-asset price version and the payoff function is given by $f(x) = (e^k - e^x)^+$ for a log-strike value k.

We set up the SDP algorithm as follows. First, we approximate $f(x)$ with $T_{2m-1}(x)^+$ and we consider the equivalence

$$p(x) \geq (T_{2m-1}(x))^+, \ \forall x \in \mathbb{R} \iff p(x) \geq T_{2m-1}(x), \ \forall x \in \mathbb{R} \quad \text{and} \quad p(x) \geq 0, \ \forall x \in \mathbb{R}.$$

Then, the optimization problem (3.47) takes the form

$$\begin{cases} \inf_{\vec{p} \in \mathbb{R}^N} & e^{-rT} \gamma^\top \vec{p}, \quad \text{s. t.} \\ & \\ & e^{-r(T-t_i)} H_n(x) e^{G_n(T-t_i)} \vec{p} \geq T_{2m-1}(x), \ \forall x \in \mathbb{R} \text{ and for } i = 0, \cdots, \ell. \\ & e^{-r(T-t_i)} H_n(x) e^{G_n(T-t_i)} \vec{p} \geq 0, \ \forall x \in \mathbb{R} \text{ and for } i = 0, \cdots, \ell. \end{cases}$$

Substituting the non-negativity conditions on each polynomial with s.o.s. conditions yields the final SDP problem. To apply the CP algorithm, we adapt the steps described in Section 3.2.2 to the American put option case and we obtain Algorithm 3.3.

We consider the model and payoff parameters

$$\sigma = 0.15, \quad x_0 = 0, \quad T = 1/12 \quad r = 0.01, \quad k = \{-0.1, 0, 0.1\}.$$

For the CP algorithm, the tolerance value is set to $tol = -10^{-5}$ and the initial discrete set $\tilde{E}^1$ is a two dimensional grid of 100×100 equidistant points in $[-20, 20] \times [0, T]$. In the SDP approach we discretize $[0, T]$ with $\ell = 10$ equidistant points and we consider the approximation $T_{n-1}(x)^+$ of $f(x)$, where n is the considered polynomial degree. The matrix G_n is computed again with respect to the monomial basis. We compute the reference price using a finite difference discretization scheme for the PDE approach (see e.g [36] and Section 1). In particular we use a θ-scheme with parameters $N_s = 40$ (number of subintervals in $s = e^x$), $S_{min} = 0.8125$, $S_{max} = 1.2297$, $\Delta t = 0.0021$, and $\theta = 0.5$. In Table 3.3 we show the obtained upper bounds, together with the corresponding absolute errors obtained via both SDP and CP approaches. Note that in this example we considered only polynomials (moments) of degree at most $n = 10$. This was sufficient to get very

Algorithm 3.3 CP routine for upper bounding American put option prices

Input: Model and payoff parameters, tolerance *tol*, maximal number of iterations maxiter
Output: Upper bound (3.46) of the American option price

1: Construct G_n and γ according to (3.27)
2: Define a random discrete set $\tilde{E}^1$ of $E \times [0,T]$
3: Build $\mathbf{H}$ and $\mathbf{f}$ and add the constraint $\gamma^\top \vec{p} \geq 0$ to the LP (3.48)
4: $l = 1$
5: **while** $l \leq$ maxiter **do**
6: Solve LP (3.48) and get $p_l^\star$
7: Compute

$$(x_1^v, t_1^v) := \text{argmin}_{(x,t)\in(-\infty,k]\times[0,T]} \Big(e^{-r(T-t)} H_n(x) e^{G_n(T-t)} p_l^\star - (e^k - e^x) \Big).$$

8: Compute

$$(x_2^v, t_2^v) := \text{argmin}_{(x,t)\in[k,\infty)\times[0,T]} \Big(e^{-r(T-t)} H_n(x) e^{G_n(T-t)} p_l^\star \Big)$$

9: Define the function $g(x,t) := e^{-r(T-t)} H_n(x) e^{G_n(T-t)} p_l^\star$
10: **if** $g(x_1^v, t_1^v) - f(x_1^v) \geq tol$ and $g(x_2^v, t_2^v) - f(x_2^v) \geq tol$ **then**
11: Break
12: **end if**
13: $\tilde{E}^{l+1} = \tilde{E}^l \cup \{(x_1^v, t_1^v), (x_2^v, t_2^v)\}$
14: Update $\mathbf{H}$ and $\mathbf{f}$
15: $l = l + 1$
16: **end while**
17: Return $e^{-rT} \gamma^\top p_l^\star$

sharp bounds, at least in the same range of accuracy as in the European case. The obtained numerical results show that both algorithmic approaches are able to produce very tight upper bounds, only by using moments of order at most 10 and for every type of moneyness. Moreover, the reference price is matched for $n = 8, 10$ in the case $k = 0.1$.

American put on minimum of two assets

In the second example we price American rainbow options in the two-dimensional Black-Scholes model, described in Section 2.2. Here, we directly model the prices (S_t^1, S_t^2) of the two assets. The payoff function is defined as a put option on the minimum of the two assets, i.e. $f(s_1, s_2) = (K - \min(s_1, s_2))^+$, for a strike value K. Recalling that the state

	SDP					
	$k=-0.1$		$k=0$		$k=0.1$	
n	UB	Abs error	UB	Abs error	UB	Abs error
2	0.0041	0.0039	0.0217	0.0047	0.1155	0.0103
4	0.0006	0.0005	0.0212	0.0042	0.1054	0.0002
6	0.0003	0.0002	0.0188	0.0018	0.1053	0.0001
8	0.0003	0.0001	0.0179	0.0009	0.1052	0.0000
10	0.0002	0.0001	0.0177	0.0007	0.1052	0.0000

	CP					
	$k=-0.1$		$k=0$		$k=0.1$	
n	UB	Abs error	UB	Abs error	UB	Abs error
2	0.0039	0.0037	0.0212	0.0042	0.1094	0.0042
4	0.0006	0.0005	0.0212	0.0042	0.1054	0.0002
6	0.0006	0.0004	0.0188	0.0018	0.1053	0.0001
8	0.0005	0.0003	0.0187	0.0017	0.1052	0.0000
10	0.0003	0.0001	0.0187	0.0017	0.1052	0.0000

Table 3.3 – Upper bounds (with absolute errors) for American put option prices obtained via SDP and CP approach in the one-dimensional Black-Scholes model. The reference prices are 0.0002 *($k=-0.1$),* 0.0170 *($k=0$), and* 0.1052 *($k=0.1$).*

space is $E=\mathbb{R}_+\times\mathbb{R}_+$, we note that the payoff function can be split on E as

$$f(s_1,s_2)=\begin{cases}0, & \text{on} \quad E_1:=\{(s_1,s_2)\in\mathbb{R}^2 \mid K\le s_1,s_2\},\\ (K-s_1), & \text{on} \quad E_2:=\{(s_1,s_2)\in\mathbb{R}^2 \mid 0\le s_1\le s_2\le K\},\\ (K-s_2), & \text{on} \quad E_3:=\{(s_1,s_2)\in\mathbb{R}^2 \mid 0\le s_2\le s_1\le K\}.\end{cases}$$

Also, for any polynomial $p(s_1,s_2)\in\mathrm{Pol}(E)$ we consider the equivalence

$$p(s_1,s_2)\ge f(s_1,s_2),\ \forall(s_1,s_2)\in E \iff \begin{cases}p(s_1,s_2)\ge 0, & \forall(s_1,s_2)\in E,\\ p(s_1,s_2)\ge K-s_1, & \forall(s_1,s_2)\in E,\\ p(s_1,s_2)\ge K-s_2, & \forall(s_1,s_2)\in E.\end{cases} \tag{3.49}$$

In order to simplify both the SDP and the CP algorithm, we impose the three inequalities in (3.49) on the set $\mathbb{R}^2$, instead of E. This produces a strengthening of the original problem, since the polynomial p is required to be positive on a larger set. However, as we

see below in the numerical experiments, this does not pose any limitation on the quality of the upper bounds. For the SDP approach, the optimization problem (3.47) takes the form

$$\begin{cases} \inf_{\vec{p}\in\mathbb{R}^N} & e^{-rT}\gamma^\top\vec{p}, \quad \text{s. t.} \\ & \\ & e^{-r(T-t_i)}H_n(s_1,s_2)e^{G_n(T-t_i)}\vec{p} \geq 0, \ \ \forall (s_1,s_2)\in\mathbb{R}^2 \text{ and for } i=0,\cdots\ell. \\ & e^{-r(T-t_i)}H_n(s_1,s_2)e^{G_n(T-t_i)}\vec{p} \geq K-s_1, \ \ \forall (s_1,s_2)\in\mathbb{R}^2 \text{ and for } i=0,\cdots\ell. \\ & e^{-r(T-t_i)}H_n(s_1,s_2)e^{G_n(T-t_i)}\vec{p} \geq K-s_2, \ \ \forall (s_1,s_2)\in\mathbb{R}^2 \text{ and for } i=0,\cdots\ell. \end{cases}$$

To apply the CP algorithm, we adapt the steps described in Section 3.2.2 and we obtain Algorithm 3.4.

We consider the model and payoff parameters

$$S_0^1=0.5, \ \ S_0^2=0.6, \ \ r=0.01, \ \ \sigma_1=\sigma_2=0.15, \ \ T=1/12, \ \ \rho=0.5, \ \ K=\{0.55,0.65\}.$$

For the CP approach the tolerance value is set to $tol=-10^{-4}$ and the initial discrete set $\tilde{E}^1$ consists of 100 uniformly distributed random points in $[-10,10]\times[-10,10]\times[0,T]$. In the SDP approach we discretize $[0,T]$ with $\ell=10$ equidistant points. The matrix G_n is computed as in Lemma 2.4. We compute the reference price using the Longstaff-Schwartz Monte Carlo approach for American options, originally developed in [94]. In particular, the reference price is computed using 10^5 simulations and by dividing $[0,T]$ in 10 subintervals. In Table 3.4 we show the obtained upper bounds, together with the corresponding absolute errors obtained via both SDP and CP approaches. Again, we consider polynomials of degree at most $n=10$. Here, we emphasize that since the dimension is $d=2$, the actual number of used moments is $N=\frac{(n+1)(n+2)}{2}$, the dimension of $\mathrm{Pol}_n(\mathbb{R}^2)$. For $n=10$ one has $N=66$.

The numerical results show again the effectiveness of the method for both numerical approaches. Indeed, the obtained upper bounds are very tight with an absolute error that reaches the order 10^{-4} for $n=10$ or less.

Algorithm 3.4 CP routine for upper bounding American rainbow option prices

Input: Model and payoff parameters, tolerance tol, maximal number of iterations $maxiter$
Output: Upper bound (3.46) of the American option price

Construct G_n and γ according to (3.27)
Define a random discrete set $\tilde{E}^1$ of $\mathbb{R}^2 \times [0,T]$
Build $\mathbf{H}$ and $\mathbf{f}$ and add the constraint $\gamma^\top \vec{p} \geq 0$ to the LP (3.48)
$l = 1$
while $l \leq maxiter$ **do**
  Solve LP (3.48) and get $p_l^\star$
  Compute

$$(s_1^{v,1}, s_2^{v,1}, t_1^v) := \text{argmin}_{(s_1,s_2,t)\in\mathbb{R}^2\times[0,T]} \Big(e^{-r(T-t)} H_n(s_1,s_2) e^{G_n(T-t)} p_l^\star - (K - s_1) \Big)$$

  Compute

$$(s_1^{v,2}, s_2^{v,2}, t_2^v) := \text{argmin}_{(s_1,s_2,t)\in\mathbb{R}^2\times[0,T]} \Big(e^{-r(T-t)} H_n(s_1,s_2) e^{G_n(T-t)} p_l^\star - (K - s_2) \Big)$$

  Compute

$$(s_1^{v,3}, s_2^{v,3}, t_3^v) := \text{argmin}_{(s_1,s_2,t)\in\mathbb{R}^2\times[0,T]} \Big(e^{-r(T-t)} H_n(s_1,s_2) e^{G_n(T-t)} p_l^\star \Big)$$

  Define the function $g(s_1,s_2,t) := e^{-r(T-t)} H_n(s_1,s_2) e^{G_n(T-t)} p_l^\star$
  if $g(s_1^{v,i}, s_2^{v,i}, t_i^v) - f(s_1^{v,i}, s_2^{v,i}) \geq tol$ for $i = 1,2,3$ **then**
    Break
  end if
  $\tilde{E}^{l+1} = \tilde{E}^l \cup \{(s_1^{v,1}, s_2^{v,1}, t_1^v), (s_1^{v,2}, s_2^{v,2}, t_2^v), (s_1^{v,3}, s_2^{v,3}, t_3^v)\}$
  Update $\mathbf{H}$ and $\mathbf{f}$
  $l = l + 1$
end while
Return $e^{-rT} \gamma^\top p_l^\star$

3.3 Conclusion

In this chapter we have proposed a method to compute bounds for European and American option prices. The method assumes the existence and the availability of all the moments of the underlying asset price process. It applies therefore to polynomial models. The core idea is to set up an optimization problem whose solution yields an upper or a lower bound of the option price of interest. More specifically, for the European case an upper bound is

	SDP			
	$K = 0.55$		$K = 0.65$	
n	UB	Abs error	UB	Abs error
2	0.0531	0.0032	0.1516	0.0018
4	0.0501	0.0002	0.1500	0.0002
6	0.0500	0.0001	0.1500	0.0002
8	0.0500	0.0001	0.1499	0.0001
10	0.0500	0.0001	0.1499	0.0001

	CP			
	$K = 0.55$		$K = 0.65$	
n	UB	Abs error	UB	Abs error
2	0.0548	0.0049	0.1538	0.0040
4	0.0505	0.0006	0.1509	0.0011
6	0.0505	0.0006	0.1503	0.0005
8	0.0502	0.0003	0.1501	0.0003
10	0.0500	0.0001	0.1501	0.0003

Table 3.4 – Upper bounds (with absolute errors) for American rainbow option prices obtained via SDP and CP approach in the 2d Black-Scholes model. The reference prices are 0.0499 *with* 95% *confidence interval* $(0.0499, 0.0500)$ *($K = 0.55$), and* 0.1498 *with* 95% *confidence interval* $(0.1497, 0.1499)$ *($K = 0.65$).*

obtained by minimizing $e^{-rT}\mathbb{E}[p(\mathbf{X}_T)]$ over the set of all upper bounding polynomials of the payoff function f, i.e. $p(\mathbf{x}) \geq f(\mathbf{x})$ for all $\mathbf{x}$ in the state space E, see (3.5). For the American case, instead, we aim at minimizing $e^{-rT}\mathbb{E}[p(\mathbf{X}_T)]$ over the set of all polynomials p that satisfy $e^{-r(T-t)}\mathbb{E}_{\mathbf{x}}[p(\mathbf{X}_T)] \geq f(\mathbf{x})$ for all $(\mathbf{x}, t) \in E \times [0, T]$, see (3.46). This can be interpreted as finding the cheapest dominating European option of the American option with payoff f. For both the European and the American case we have proposed two algorithmic techniques to compute the numerical solution of the optimization problems. The first one is based on semidefinite programming, while the second one follows the cutting plane technique. For the European case, we have obtained convergence results for the one-dimensional case and for different classes of payoff functions and state spaces. The numerical experiments that have been performed for different types of models showed that the methodology yields tight bounds and, in the case of European option, the mid-price can be used as a price approximation. Moreover, the CP algorithm performs better than the SDP one for moments of low degree. This indicates that our extension to non

piecewise polynomial payoff functions combined with the CP algorithm is valid. The overall method is therefore effective. Finally, for the European case we have explained how to efficiently design a black box algorithm able to take model and payoff parameters as input, and return the option price in output. This algorithm is based on an efficient computation of the moment sequence which is directly connected to the next chapter.

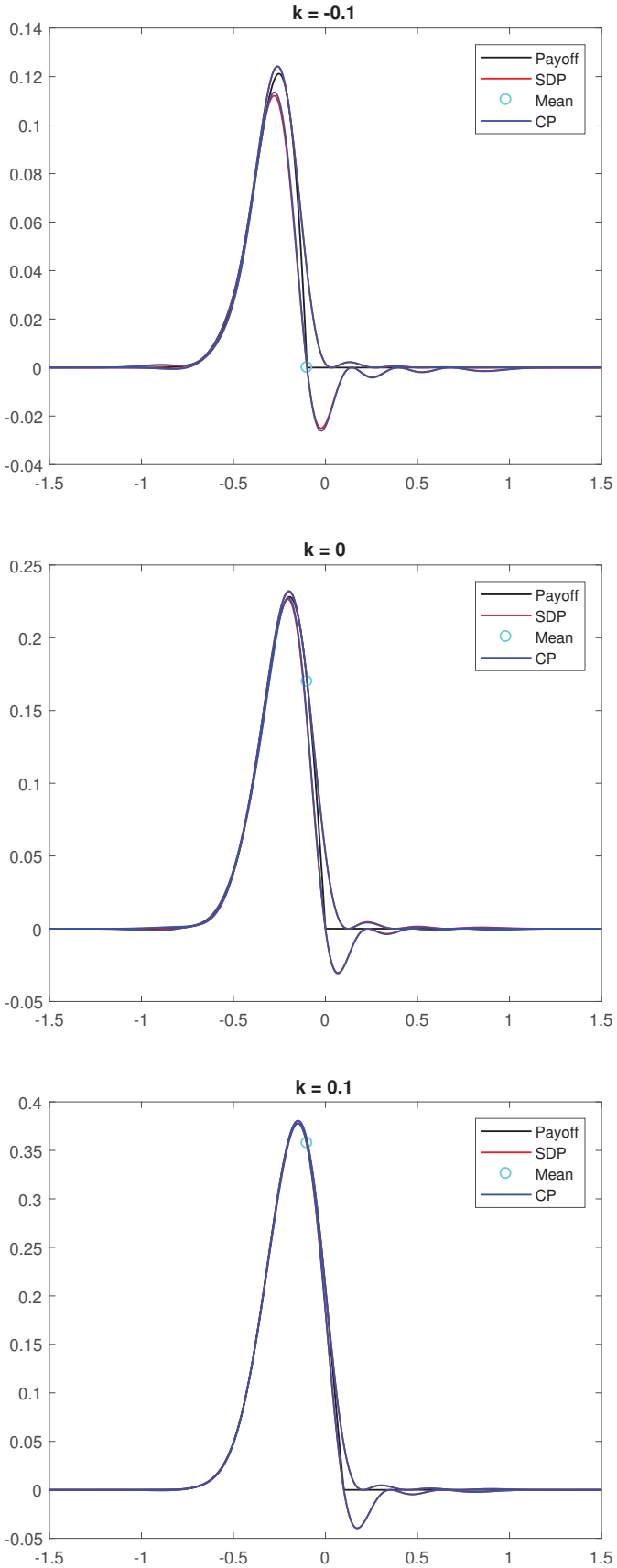

Figure 3.1 – Bounding polynomials of degree $n = 12$ multiplied by the density function for the European put option payoff in the Black-Scholes model, for different log-strike values k.

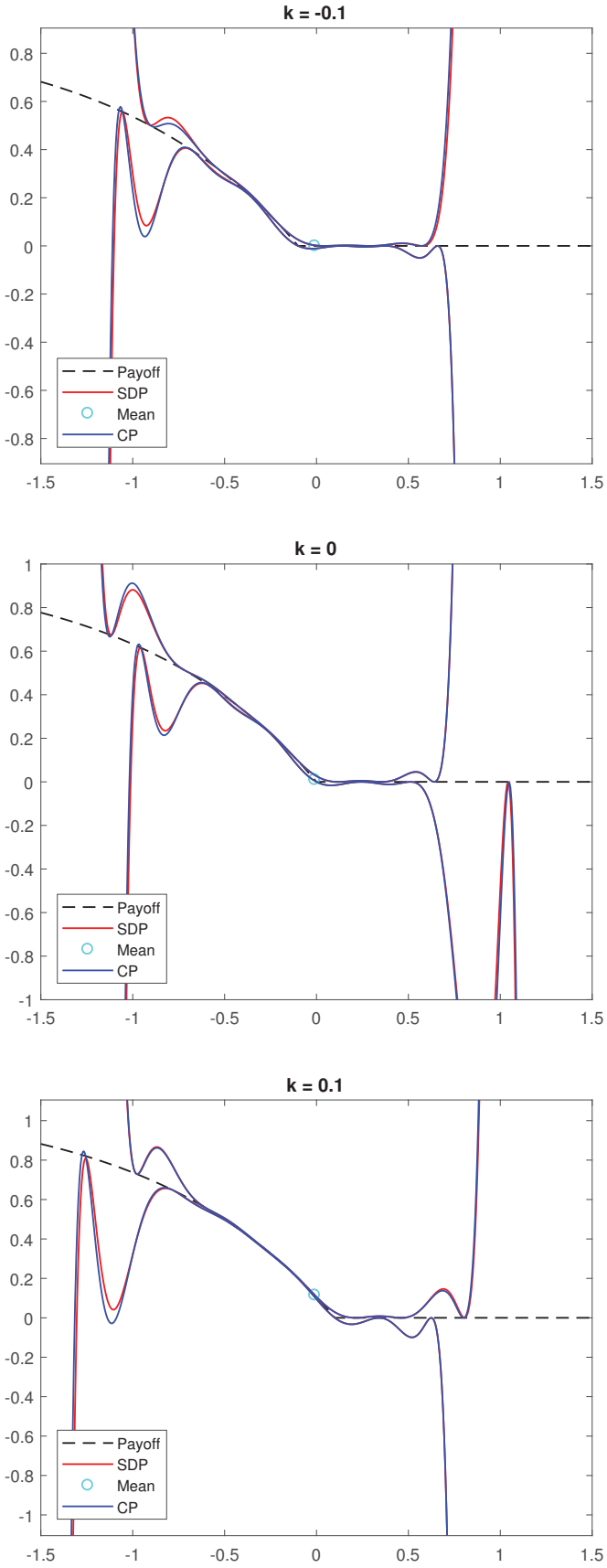

Figure 3.2 – Bounding polynomials of degree $n = 12$ for the European put option payoff in the Jacobi model, for different log-strike values k.

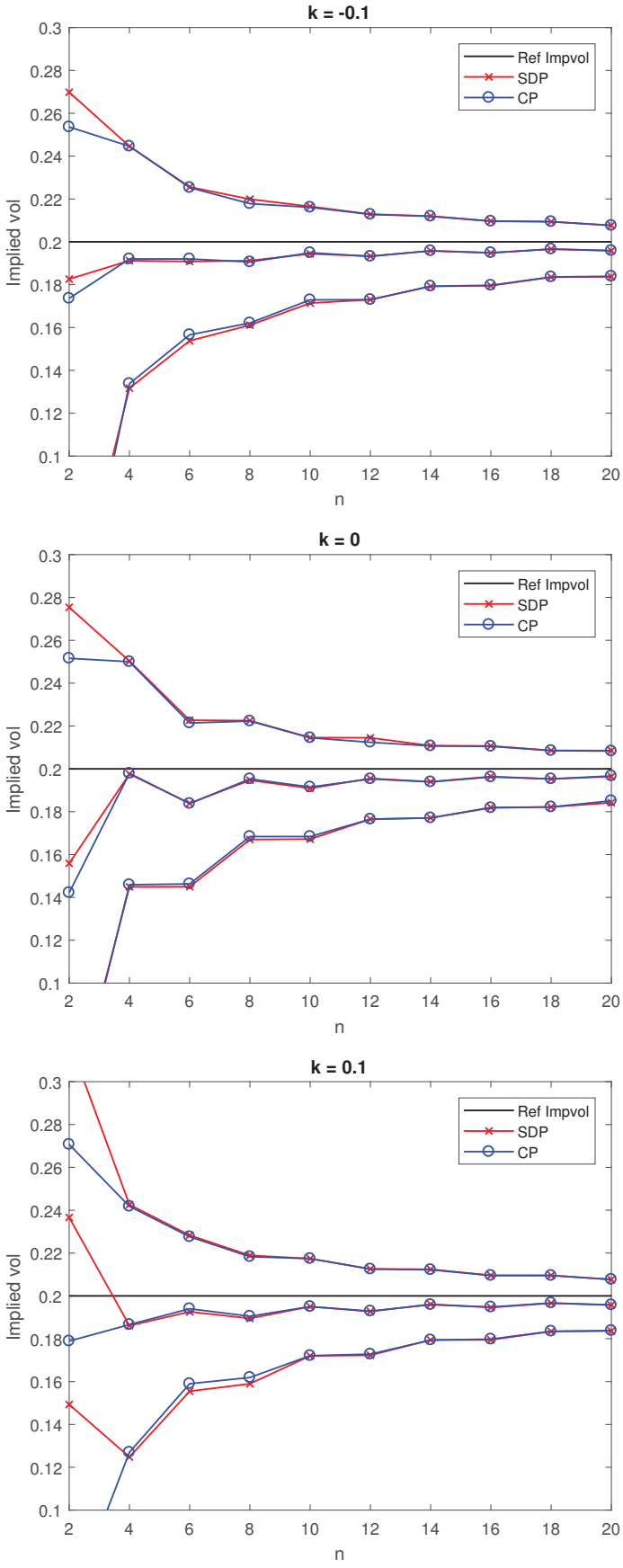

Figure 3.3 – Implied volatilities of the upper bounds, lower bounds and mid-price for different values of n (x-axis) in the Black-Scholes model. For all choices of the log-strike k the reference implied volatility is 0.2.

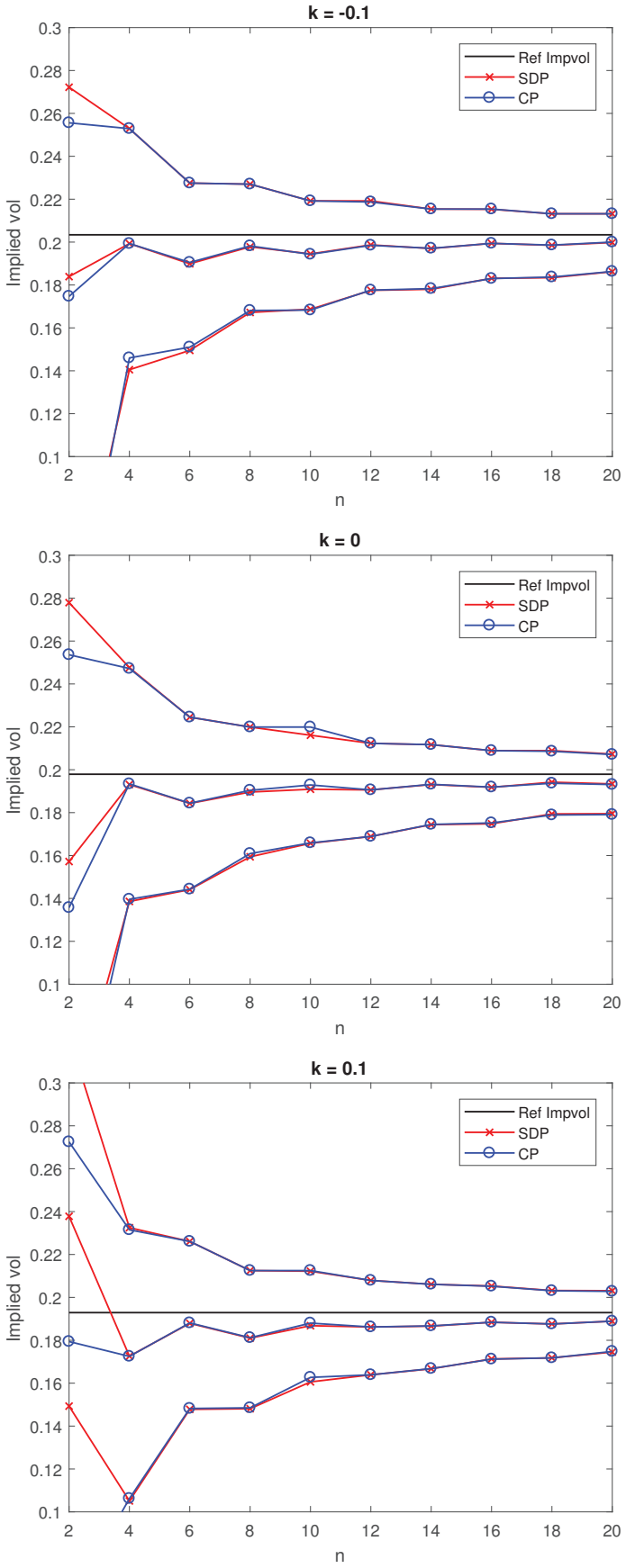

Figure 3.4 – Implied volatilities of the upper bounds, lower bounds and mid-price for different values of n (x-axis) in the Jacobi model. The reference implied volatilities are 0.2034 *(for* $k = -0.1$*),* 0.1979 *(for* $k = 0$*) and* 0.1929 *(for* $k = 0.1$*).*

4 Efficient incremental computation of the moment sequence

In this chapter we develop an algorithm for the incremental computation of block triangular matrix exponentials. More precisely, we study the problem of computing the matrix exponential of a block triangular matrix in a peculiar way: block column by block column, from left to right. Since the moments of polynomial jump-diffusions are given in closed form via the moment formula (2.6), and the sequence $G_0, G_1, G_2, \ldots$ of matrix representations of the generator $\mathcal{G}$ restricted to the polynomial spaces $\mathrm{Pol}_0(E), \mathrm{Pol}_1(E), \mathrm{Pol}_2(E), \ldots$ forms a nested sequence of block upper triangular matrices (see Chapter 2), this algorithm allows for an efficient incremental computation of the moment sequence. In other words, we aim to efficiently compute the moments of order $0, 1, 2, 3, \ldots.$ until a dynamically evaluated criterion tells us to stop. The need for such an evaluation scheme arises in the context of option pricing in polynomial models. An example is given by the Algorithm 3.2, where in Step 4 such an algorithm can be used to update the vector of moments γ. Our algorithm is based on scaling and squaring. By carefully reusing certain intermediate quantities from one step to the next, we can efficiently compute the required sequence of matrix exponentials.

The rest of this chapter is organized as follows. In Section 4.1 we motivate the development of our algorithm and we give a precise problem formulation. In Section 4.2 we provide a detailed description of the algorithm. Finally, in Section 4.3 we present numerical experiments, some of which treat the option pricing problem in polynomial models. This chapter is mostly based on the paper [83].

4.1 Motivation and problem formulation

Some option pricing techniques require an incremental computation of the moments of the underlying asset price $\mathbf{X}_T$, in the sense that moments of $\mathbf{X}_T$ of order $0, 1, 2, 3, \dots$ are to be computed sequentially, one after the other, until a dynamically evaluated stopping criterion allows to stop. In particular, increasing the order n of the computed moments allows for a better approximation of the option price, however, the value of n required to attain a desired accuracy is usually not known a priori. One of these algorithms is illustrated in Algorithm 3.2, in the Chapter 3 of this thesis. There, updating the vector γ of moments allows to incrementally compute the bounds of the option price and the algorithm can stop as soon as the bounds are close enough to each other, in an ϵ-neighborhood of the exact price. An other algorithm that requires an efficient incremental computation of the moment sequence arises when considering the pricing technique based on polynomial expansions, presented in Section 2.3 of Chapter 2. There, it is possible to design a heuristic algorithm to select the truncation value N in (2.18) based on the absolute value of the summands. We present this algorithm later in Section 4.3.

When considering polynomial jump-diffusions, the moment formula (2.6) implies that computing the sequence of moments boils down to evaluating quantities of the form $H_n(\mathbf{X}_0)e^{G_n{}^T\vec{p}}$, for $n = 0, 1, 2, \dots$, and for p belonging to the nested sequence of polynomial spaces $\mathrm{Pol}_0(E) \subseteq \mathrm{Pol}_1(E) \subseteq \mathrm{Pol}_2(E) \dots$. As mentioned in Chapter 2, Section 2.1, the matrix sequence $G_0, G_1, G_2, \dots$ forms a nested sequence of block triangular matrices. Therefore, we are required to incrementally compute the nested block triangular matrix exponentials

$$\exp(G_0),\ \exp(G_1),\ \exp(G_2),\ \dots$$

step by step. We develop an algorithm able to perform this task efficiently.

We now give the general problem formulation. Consider a sequence of block upper triangular matrices $G_0, G_1, G_2, \dots$ of the form

$$G_n = \begin{bmatrix} G_{0,0} & G_{0,1} & \cdots & G_{0,n} \\ & G_{1,1} & \cdots & G_{1,n} \\ & & \ddots & \vdots \\ & & & G_{n,n} \end{bmatrix} \in \mathbb{R}^{d_n \times d_n}, \tag{4.1}$$

where all diagonal blocks $G_{n,n}$ are square. In other words, the matrix G_i arises from G_{i-1} by appending a block column (and adjusting the size). We aim at computing the sequence of matrix exponentials

$$\exp(G_0),\ \exp(G_1),\ \exp(G_2),\ \ldots\ . \tag{4.2}$$

One could, of course, simply compute each of the exponentials (4.2) individually using standard techniques (see [96] for an overview). However, the sequence of matrix exponentials (4.2) inherits the nested structure from the matrices G_n in (4.1), i.e., $\exp(G_{n-1})$ is a leading principle submatrix of $\exp(G_n)$. In effect only the last block column of $\exp(G_n)$ needs to be computed and the goal of this chapter is to explain how this can be achieved in a numerically safe manner.

In the special case where the spectra of the diagonal blocks $G_{n,n}$ are separated, Parlett's method [102] yields – in principle – an efficient computational scheme: Compute $F_{0,0} := \exp(G_{0,0})$ and $F_{1,1} := \exp(G_{1,1})$ separately, then the missing (1,2) block of $\exp(G_1)$ is given as the unique solution X to the Sylvester equation

$$G_{0,0}X - XG_{1,1} = F_{0,0}G_{0,1} - G_{0,1}F_{1,1}.$$

Continuing in this manner all the off-diagonal blocks required to compute (4.2) could be obtained from solving Sylvester equations. However, it is well known (see Chapter 9 in [67]) that Parlett's method is numerically safe only when the spectra of the diagonal blocks are well separated, in the sense that all involved Sylvester equations are well conditioned. Since we consider the block structure as fixed, imposing such a condition would severely limit the scope of applications; it is certainly not met by the application in option pricing for polynomial models we discuss below. For instance, one can easily see that the spectra of the first three diagonal blocks of the matrix G_2 arising in the Jacobi model, explicitly shown in the Equation (2.15), are not separated. Indeed, the diagonal blocks $G_{0,0}, G_{1,1}$ and $G_{2,2}$ have the common eigenvalue 0, and the blocks $G_{1,1}$ and $G_{2,2}$ share the eigenvalue $-\kappa$. Therefore, Parlett's method would not be suitable to compute the sequence (4.2) for this particular case.

Exponentials of block triangular matrices have also been studied in other contexts. For two-by-two block triangular matrices, Dieci and Papini study conditioning issues in [34],

and a discussion on the choice of scaling parameters for using Padé approximants to exponential function in [33]. In the case where the matrix is also block-Toeplitz, a fast exponentiation algorithm is developed in [16].

4.2 Incremental scaling and squaring

Since the set of conformally partitioned block triangular matrices forms an algebra, and $\exp(G_n)$ is a polynomial in G_n, the matrix $\exp(G_n)$ has the same block upper triangular structure as G_n, that is,

$$\exp(G_n) = \begin{bmatrix} \exp(G_{0,0}) & * & \cdots & * \\ & \exp(G_{1,1}) & \ddots & \vdots \\ & & \ddots & * \\ & & & \exp(G_{n,n}) \end{bmatrix} \in \mathbb{R}^{d_n \times d_n}.$$

As outlined in the introduction, we aim at computing $\exp(G_n)$ block column by block column, from left to right. Our algorithm is based on the scaling and squaring methodology, which we briefly summarize next.

4.2.1 Summary of the scaling and squaring method

The scaling and squaring method uses a rational function to approximate the exponential function, and typically involves three steps. Denote by $r_{k,m}(z) = \frac{p_{k,m}(z)}{q_{k,m}(z)}$ the (k,m)-Padé approximant to the exponential function, meaning that the numerator is a polynomial of degree k, and the denominator is a polynomial of degree m. These Padé approximants are very accurate close to the origin, and in a first step the input matrix G is therefore scaled by a power of two, so that $\|2^{-s}G\|$ is small enough to guarantee an accurate approximation $r_{k,m}(2^{-s}G) \approx \exp(2^{-s}G)$.

The second step consists of evaluating the rational approximation $r_{k,m}(2^{-s}G)$, and, finally, an approximation to $\exp(G)$ is obtained in a third step by repeatedly squaring the result, i.e.,

$$\exp(G) \approx r_{k,m}(2^{-s}G)^{2^s}.$$

Different choices of the scaling parameter s, and of the approximation degrees k and m

yield methods of different characteristics. The choice of these parameters is critical for the approximation quality, and for the computational efficiency, see [67, Chapter 10].

In what follows we describe techniques that allow for an incremental evaluation of the matrix exponential of the block triangular matrix (4.1), using scaling and squaring. These techniques can be used with any choice for the actual underlying scaling and squaring method, defined through the parameters s, k, and m.

4.2.2 Tools for the incremental computation of exponentials

Before explaining the algorithm, we first introduce some notation that is used throughout. The matrix G_n from (4.1) can be written as

$$G_n = \left[\begin{array}{ccc|c} G_{0,0} & \cdots & G_{0,n-1} & G_{0,n} \\ & \ddots & \vdots & \vdots \\ & & G_{n-1,n-1} & G_{n-1,n} \\ \hline & & & G_{n,n} \end{array}\right] =: \begin{bmatrix} G_{n-1} & g_n \\ 0 & G_{n,n} \end{bmatrix} \tag{4.3}$$

where $G_{n-1} \in \mathbb{R}^{d_{n-1} \times d_{n-1}}$, $G_{n,n} \in \mathbb{R}^{b_n \times b_n}$, so that $g_n \in \mathbb{R}^{d_{n-1} \times b_n}$. Let s be the scaling parameter, and $r = \frac{p}{q}$ the rational function used in the approximation (for simplicity we will often omit the indices k and m). We denote the scaled matrix by $\tilde{G}_n := 2^{-s} G_n$ and we partition it as in (4.3).

The starting point of the algorithm consists in computing the Padé approximant of the exponential $\exp(G_0) = \exp(G_{0,0})$, using a scaling and squaring method. Then, the sequence of matrix exponentials (4.2) is incrementally computed by reusing at each step previously obtained quantities. So more generally, assume that $\exp(G_{n-1})$ has been approximated by using a scaling and squaring method. The three main computational steps for obtaining the Padé approximant of $\exp(G_n)$ are

1. evaluating the polynomials $p(\tilde{G}_n)$, $q(\tilde{G}_n)$,
2. evaluating $p(\tilde{G}_n)^{-1} q(\tilde{G}_n)$, and
3. repeatedly squaring it.

We now discuss each of these steps separately, noting the quantities to keep at every

iteration.

Evaluating $p(\tilde{G}_n)$, $q(\tilde{G}_n)$ from $p(\tilde{G}_{n-1})$, $q(\tilde{G}_{n-1})$

Similarly to (4.3), we start by writing $P_n := p(\tilde{G}_n)$ and $Q_n := q(\tilde{G}_n)$ as

$$P_n = \begin{bmatrix} P_{n-1} & p_n \\ 0 & P_{n,n} \end{bmatrix}, \quad Q_n = \begin{bmatrix} Q_{n-1} & q_n \\ 0 & Q_{n,n} \end{bmatrix}.$$

In order to evaluate P_n, we first need to compute monomials of $\tilde{G}_n$, which for $l = 1, \ldots, k$, can be written as

$$\tilde{G}_n^l = \begin{bmatrix} \tilde{G}_{n-1}^l & \sum_{j=0}^{l-1} \tilde{G}_{n-1}^j \tilde{g}_n \tilde{G}_{n,n}^{l-j-1} \\ & \tilde{G}_{n,n}^l \end{bmatrix}.$$

Denote by $X_l := \sum_{j=0}^{l-1} \tilde{G}_{n-1}^j \tilde{g}_n \tilde{G}_{n,n}^{l-j-1}$ the upper off diagonal block of $\tilde{G}_n^l$, then we have the relation

$$X_l = \tilde{G}_{n-1} X_{l-1} + \tilde{g}_n \tilde{G}_{n,n}^{l-1}, \quad \text{for } l = 2, \cdots, k,$$

with $X_1 := \tilde{g}_n$, so that all the monomials $\tilde{G}_n^l$, $l = 1, \ldots, k$, can be computed in $\mathcal{O}(b_n^3 + d_{n-1} b_n^2 + d_{n-1}^2 b_n)$. Let $p(z) = \sum_{l=0}^{k} \alpha_l z^l$ be the numerator polynomial of r, then we have that

$$P_n = \begin{bmatrix} P_{n-1} & \sum_{l=0}^{k} \alpha_l X_l \\ & p(\tilde{G}_{n,n}) \end{bmatrix}, \tag{4.4}$$

which can be assembled in $\mathcal{O}(b_n^2 + d_{n-1} b_n)$, since only the last block column needs to be computed. The complete evaluation of P_n is summarized in Algorithm 4.1.

Similarly, one computes Q_n from Q_{n-1}, using again the matrices X_l.

Evaluating $Q_n^{-1} P_n$

With the matrices P_n, Q_n at hand, we now need to compute the rational approximation $Q_n^{-1} P_n$. We assume that Q_n is well conditioned, in particular non-singular, which is ensured by the choice of the scaling parameter and of the Padé approximation, see,

Algorithm 4.1 Evaluation of P_n, using P_{n-1}

Input: $G_{n-1}, G_{n,n}, g_n, P_{n-1}$, Padé coefficients $\alpha_l, l = 0, \cdots, k$.
Output: P_n.

1: $\tilde{g}_n \leftarrow 2^{-s} g_n$, $\tilde{G}_{n,n} \leftarrow 2^{-s} G_{n,n}$, $\tilde{G}_{n-1} \leftarrow 2^{-s} G_{n-1}$
2: $X_1 \leftarrow \tilde{g}_n$
3: **for** $l = 2, 3, \cdots, k$ **do**
4: Compute $\tilde{G}_{n,n}^l$
5: $X_l = \tilde{G}_{n-1} X_{l-1} + \tilde{g}_n \tilde{G}_{n,n}^{l-1}$
6: **end for**
7: $X_0 \leftarrow \mathbf{0}_{d_{n-1} \times b_n}$
8: Compute off diagonal block of P_n: $\sum_{l=0}^{k} \alpha_l X_l$.
9: Compute $p(\tilde{G}_{n,n}) = \sum_{l=0}^{k} \alpha_l \tilde{G}_{n,n}^l$
10: Assemble P_n as in (4.4)

e.g., [68]. We focus on the computational cost. For simplicity, we introduce the notation

$$\tilde{F}_n = \begin{bmatrix} \tilde{F}_{0,0} & \cdots & \tilde{F}_{0,n} \\ & \ddots & \vdots \\ & & \tilde{F}_{n,n} \end{bmatrix} := Q_n^{-1} P_n, \quad F_n = \begin{bmatrix} F_{0,0} & \cdots & F_{0,n} \\ & \ddots & \vdots \\ & & F_{n,n} \end{bmatrix} := \tilde{F}_n^{2^s},$$

and we see that

$$\begin{aligned} \tilde{F}_n = Q_n^{-1} P_n &= \begin{bmatrix} Q_{n-1}^{-1} & -Q_{n-1}^{-1} q_n Q_{n,n}^{-1} \\ 0 & Q_{n,n}^{-1} \end{bmatrix} \begin{bmatrix} P_{n-1} & p_n \\ 0 & P_{n,n} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{F}_{n-1} & Q_{n-1}^{-1}(p_n - q_n Q_{n,n}^{-1} P_{n,n}) \\ 0 & Q_{n,n}^{-1} P_{n,n} \end{bmatrix}. \end{aligned} \tag{4.5}$$

To solve the linear system $Q_{n,n}^{-1} P_{n,n}$ we compute an LU decomposition with partial pivoting for $Q_{n,n}$, requiring $\mathcal{O}(b_n^3)$ operations. This LU decomposition is saved for future use, and hence we may assume that we have available the LU decompositions for all diagonal blocks from previous computations:

$$\Pi_l Q_{l,l} = L_l U_l, \quad l = 0, \ldots, n-1. \tag{4.6}$$

Here, $\Pi_l \in \mathbb{R}^{b_l \times b_l}$, $l = 0, \ldots, n-1$ are permutation matrices; $L_l \in \mathbb{R}^{b_l \times b_l}$, $l = 0, \ldots, n-1$ are lower triangular matrices and $U_l \in \mathbb{R}^{b_l \times b_l}$, $l = 0, \ldots, n-1$ are upper triangular matrices.

Set $Y_n := p_n - q_n Q_{n,n}^{-1} P_{n,n} \in \mathbb{R}^{d_{n-1} \times b_n}$, and partition it as

$$Y_n = \begin{bmatrix} Y_{0,n} \\ \vdots \\ Y_{n-1,n} \end{bmatrix}.$$

Then we compute $Q_{n-1}^{-1} Y_n$ by block backward substitution, using the decompositions of the diagonal blocks. The total number of operations for this computation is hence $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2)$, so that the number of operations for computing $\tilde{F}_n$ is $\mathcal{O}(b_n^3 + d_{n-1}^2 b_n + d_{n-1} b_n^2)$. Algorithm 4.2 describes the complete procedure to compute $\tilde{F}_n$.

Algorithm 4.2 Evaluation of $\tilde{F}_n = Q_n^{-1} P_n$

Input: Q_n, P_n and quantities (4.6)
Output: $\tilde{F}_n = Q_n^{-1} P_n$ and LU decomposition of $Q_{n,n}$.

1: Compute $\Pi_n Q_{n,n} = L_n U_n$ and keep it for future use (4.6)
2: Compute $\tilde{F}_{n,n} := Q_{n,n}^{-1} P_{n,n}$
3: $Y_n = p_n - q_n Q_{n,n}^{-1} P_{n,n}$
4: $\tilde{F}_{n-1,n} = U_{n-1}^{-1} L_{n-1}^{-1} \Pi_{n-1} Y_{n-1,n}$
5: **for** $l = n-2, n-3, \cdots, 0$ **do**
6: $\quad \tilde{F}_{l,n} = U_l^{-1} L_l^{-1} \Pi_l (Y_{l,n} - \sum_{j=l+1}^{n-1} Q_{l,j} \tilde{F}_{j,n})$
7: **end for**
8: Assemble $\tilde{F}_n$ as in (4.5)

The squaring phase

Having computed $\tilde{F}_n$, which we write as

$$\tilde{F}_n = \begin{bmatrix} \tilde{F}_{n-1} & \tilde{f}_n \\ & \tilde{F}_{n,n} \end{bmatrix},$$

we now need to compute s repeated squares of that matrix, i.e.,

$$\tilde{F}_n^{2^l} = \begin{bmatrix} \tilde{F}_{n-1}^{2^l} & \sum_{j=0}^{l-1} \tilde{F}_{n-1}^{2^{l-1+j}} \tilde{f}_n \tilde{F}_{n,n}^{2^j} \\ & \tilde{F}_{n,n}^{2^l} \end{bmatrix}, \quad l = 1, \ldots, s, \tag{4.7}$$

so that $F_n = \tilde{F}_n^{2^s}$. Setting $Z_l := \sum_{j=0}^{l-1} \tilde{F}_{n-1}^{2^{l-1+j}} \tilde{f}_j \tilde{F}_{n,n}^{2^j}$, we have the recurrence

$$Z_l = \tilde{F}_{n-1}^{2^{l-1}} Z_{l-1} + Z_{l-1} \tilde{F}_{n,n}^{2^{l-1}},$$

with $Z_0 := \tilde{f}_n$. Hence, if we have stored the intermediate squares from the computation of F_{n-1}, i.e.,

$$\tilde{F}_{n-1}^{2^l}, \quad l = 1, \ldots, s \tag{4.8}$$

we can compute all the quantities Z_l, $l = 1, \ldots, s$ in $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2)$, so that the total cost for computing F_n (and the intermediate squares of $\tilde{F}_n$) is $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2 + b_n^3)$. Again, we summarize the squaring phase in the following algorithm.

Algorithm 4.3 Evaluation of $F_n = \tilde{F}_n^{2^s}$

Input: $\tilde{F}_{n-1}, \tilde{f}_n, \tilde{F}_{n,n}$, quantities (4.8).
Output: F_n and updated intermediates.
1: $Z_0 \leftarrow \tilde{f}_n$
2: **for** $l = 1, 2, \cdots, s$ **do**
3: Compute $\tilde{F}_{n,n}^{2^l}$
4: $Z_l = \tilde{F}_{n-1}^{2^{l-1}} Z_{l-1} + Z_{l-1} \tilde{F}_{n,n}^{2^{l-1}}$
5: Assemble $\tilde{F}_n^{2^l}$ as in (4.7) and save it
6: **end for**
7: $F_n \leftarrow \tilde{F}_n^{2^s}$

4.2.3 Overall algorithm

Using the techniques from the previous section, we now give a concise description of the overall algorithm. We assume that the quantities listed in equations (4.6) and (4.8) are stored in memory, with a space requirement of $\mathcal{O}(d_{n-1}^2)$.

In view of this, we assume that F_{n-1} and the aforementioned intermediate quantities have been computed. Algorithm 4.4 describes the overall procedure to compute F_n, and to update the intermediates; we continue to use the notation introduced in (4.3).

Algorithm 4.4 Computation of $F_n \approx \exp(G_n)$, using F_{n-1}

Input: Block column g_n, diagonal block $G_{n,n}$, quantities (4.6), and (4.8).
Output: F_n, and updated intermediates.
1: Extend P_{n-1} to P_n using Algorithm 4.1, and form analogously Q_n
2: Compute $\tilde{F}_n$ using Algorithm 4.2
3: Evaluate $F_n = \tilde{F}_n^{2^s}$ using Algorithm 4.3

As explained in the previous section, the number of operations for each step in Algorithm 4.4 is $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2 + b_n^3)$, using the notation at the beginning of Section 4.2.2.

If F_n were simply computed from scratch, without the use of the intermediates, the number of operations for scaling and squaring would be $\mathcal{O}((d_{n-1} + b_n)^3)$. In the typical situation where $d_{n-1} \gg b_n$, the dominant term in the latter complexity bound is d_{n-1}^3, which is absent from the complexity bound of Algorithm 4.4.

In order to solve our original problem, the computation of the sequence $\exp(G_0)$, $\exp(G_1)$, $\exp(G_2)$, ..., we use Algorithm 4.4 repeatedly; the resulting procedure is shown in Algorithm 4.5.

Algorithm 4.5 Approximation of $\exp(G_0), \exp(G_1), \ldots$

Input: Padé approximation parameters k, m, and s
Output: $F_0 \approx \exp(G_0)$, $F_1 \approx \exp(G_1), \ldots$

```
1: Compute F_0 using scaling and squaring, store intermediates for Algorithm 4.4
2: for n = 1, 2, ... do
3:    Compute F_n from F_{n-1} using Algorithm 4.4
4:    if termination criterion is satisfied then
5:       return
6:    end if
7: end for
```

We now derive a complexity bound for the number of operations spent in Algorithm 4.5. For simplicity of notation we consider the case where all diagonal blocks are of equal size, i.e., $b_k \equiv b \in \mathbb{N}$, so that $d_k = (k+1)b$. At iteration k the number of operations spent within Algorithm 4.4 is thus $\mathcal{O}(k^2 b^3)$. Assume that the termination criterion used in Algorithm 4.5 effects to stop the procedure after the computation of F_n. The overall complexity bound for the number of operations until termination is $\mathcal{O}(\sum_{k=0}^{n} k^2 b^3) = \mathcal{O}(n^3 b^3)$, which matches the complexity bound of applying scaling and squaring only to $G_n \in \mathbb{R}^{(n+1)b \times (n+1)b}$, which is also $\mathcal{O}((nb)^3)$.

In summary the number of operations needed to compute F_n by Algorithm 4.5 is asymptotically the same as applying the same scaling and squaring setting *only* to compute $\exp(G_n)$, while Algorithm 4.5 incrementally reveals *all* exponentials $\exp(G_0), \ldots, \exp(G_n)$ in the course of the iteration, satisfying our requirements outlined in the introduction.

4.2.4 Adaptive scaling

In Algorithms 4.4 and 4.5 we have assumed that the scaling power s is given as input parameter, and that it is fixed throughout the computation of $\exp(G_0), \ldots, \exp(G_n)$.

This is in contrast to what is usually intented in the scaling and squaring method, see Section 4.2.1. On the one hand s must be sufficiently large so that $r_{k,m}(2^{-s}G_l) \approx \exp(2^{-s}G_l)$, for $0 \leq l \leq n$. If, on the other hand, s is chosen *too large*, then the evaluation of $r_{k,m}(2^{-s}G_l)$ may become inaccurate, due to *overscaling*. So if s is fixed, and the norms $\|G_l\|$ grow with increasing l, as one would normally expect, an accurate approximation cannot be guaranteed for all l.

Most scaling and squaring designs hence choose s in dependence of the norm of the input matrix [96, 58, 68]. For example, in the algorithm of Higham described in [68], it is the smallest integer satisfying

$$\|2^{-s}G_l\|_1 \leq \theta \approx 5.37.... \tag{4.9}$$

In order to combine our incremental evaluation techniques with this scaling and squaring design, the scaling power s must thus be chosen dynamically in the course of the evaluation. Assume that s satisfies the criterion (4.9) at step $l-1$, but not at step l. We then simply discard all accumulated data structures from Algorithm 4.4, increase s to match the bound (4.9) for G_l, and start Algorithm 4.5 anew with the *repartitioned* input matrix

$$G_n = \left[\begin{array}{ccc|c|c|c} G_{0,0} & \cdots & G_{0,l} & G_{0,l+1} & \cdots & G_{0,n} \\ & \ddots & \vdots & \vdots & & \vdots \\ & & G_{l,l} & G_{l,l+1} & \cdots & G_{l,n} \\ \hline & & & G_{l+1,l+1} & \cdots & G_{l+1,n} \\ & & & & \ddots & \vdots \\ & & & & & G_{n,n} \end{array}\right] = \underbrace{\left[\begin{array}{c|c|c|c} \hat{G}_{0,0} & \hat{G}_{0,1} & \cdots & \hat{G}_{0,n-l} \\ \hline & \hat{G}_{1,1} & \cdots & \hat{G}_{1,n-l} \\ & & \ddots & \vdots \\ & & & \hat{G}_{n-l,n-l} \end{array}\right]}_{=:\hat{G}_{n-l}}. \tag{4.10}$$

The procedure is summarized in Algorithm 4.6.

It turns out that the computational overhead induced by this restarting procedure is quite modest. In the notation introduced for the complexity discussion in Section 4.2.3, the number of operations for computing $\exp(G_n)$ by Higham's scaling and squaring method is $\mathcal{O}(\log(\|G_n\|_1)(nb)^3)$. Since there are at most $\log(\|G_n\|_1)$ restarts in Algorithm 4.6, the total number of operations for incrementally computing all exponentials $\exp(G_0), \ldots, \exp(G_n)$ can be bounded by a function in $\mathcal{O}(\log(\|G_n\|_1)^2(nb)^3)$. We assess the actual performance of Algorithm 4.6 in Section 4.3.

In the application from option pricing in polynomial models it turns out that the norms

Algorithm 4.6 Approximation of $\exp(G_0), \exp(G_1), \ldots$ with adaptive scaling

Input: Padé approximation parameters k, m, norm bound θ.
Output: $F_0 \approx \exp(G_0)$, $F_1 \approx \exp(G_1), \ldots$

```
s ← max{0, log(‖G_0‖_1)}
Compute F_0 using scaling and squaring, store intermediates for Algorithm 4.4
for l = 1, 2, ... do
    if ‖G_l‖_1 > θ then
        Repartition G_n = Ĝ_{n−l} as in (4.10)
        Restart algorithm with Ĝ_{n−l}.
    end if
    Compute F_l from F_{l−1} using Algorithm 4.4
    if termination criterion is satisfied then
        return
    end if
end for
```

of the matrices G_l do not grow dramatically and quite accurate approximations to all the matrix exponentials can be computed even if the scaling factor is fixed (see Section 4.3.2).

4.3 Numerical experiments

We have implemented the algorithms described in this chapter in MATLAB and compared them with Higham's scaling and squaring method from [68], which typically employs a diagonal Padé approximation of degree 13 and is referred to as "`expm`" in the following. The implementation of our algorithms for block triangular matrices, Algorithm 4.5 (fixed scaling parameter), and Algorithm 4.6 (adaptive scaling parameter), is based on the same scaling and squaring design and are referred to as "`incexpm`' in the following. All experiments were run on a standard laptop (Intel Core i5, 2 cores, 256kB/4MB L2/L3 cache) using a single computational thread.

4.3.1 Random block triangular matrices

We first assess run time and accuracy on a randomly generated block upper triangular matrix $G_n \in \mathbb{R}^{2491\times 2491}$. There are 46 diagonal blocks, of size varying between 20 and 80. The matrix is generated to have a spectrum contained in the interval $[-80, -0.5]$, and a well conditioned eigenbasis X ($\kappa_2(X) \approx 100$).

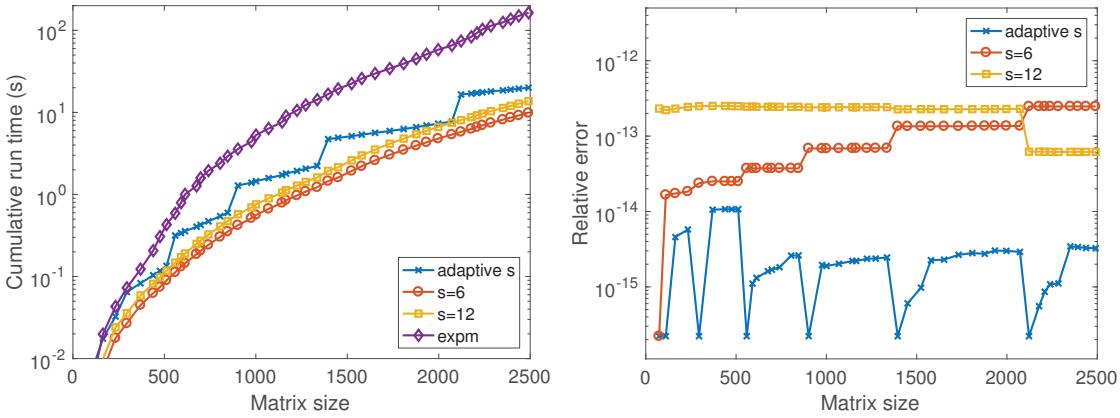

Figure 4.1 – Comparison of `incexpm` *and* `expm` *for a random block triangular matrix.* Left: *Cumulative run time for computing the leading portions.* Right: *Relative error of* `incexpm` *w.r.t.* `expm`*.*

Figure 4.1 (left) shows the wall clock time for the incremental computation of all the leading exponentials. Specifically, given $0 \leq l \leq n$, each data point shows the time vs. $d_l = b_0 + \cdots + b_l$ needed for computing the $l+1$ matrix exponentials $\exp(G_0), \exp(G_1), \ldots, \exp(G_l)$ when using

- `expm` (by simply applying it to each matrix separately);
- `incexpm` with the adaptive scaling strategy from Algorithm 4.6;
- `incexpm` with fixed scaling power 6 (scaling used by `expm` for G_0);
- `incexpm` with fixed scaling power 12 (scaling used by `expm` for G_n).

As expected, `incexpm` is much faster than naively applying `expm` to each matrix separately; the total times for $l = n$ are also displayed in Table 4.1. For reference we remark that the run time of MATLAB's `expm` applied only the final matrix G_n is 13.65s, which is very close to the run time of `incexpm` with scaling parameter set to 12 (see Section 4.2.3 for a discussion of the asymptotic complexity). Indeed, a closer look at the runtime profile of `incexpm` reveals that the computational overhead induced by the more complicated data structures is largely compensated in the squaring phase by taking advantage of the block triangular matrix structure, from which MATLAB's `expm` does not profit automatically. It is also interesting to note that the run time of the adaptive scaling strategy is roughly only twice the run time for running the algorithm with a fixed scaling parameter 6, despite its worse asymptotic complexity.

Algorithm	Time (s)	Rel. error
`expm`	163.60	
`incexpm` (adaptive)	20.01	3.27e-15
`incexpm` ($s = 6$)	9.85	2.48e-13
`incexpm` ($s = 12$)	13.70	6.17e-14

Table 4.1 – Run time and relative error attained by `expm` *and* `incexpm` *on a random block triangular matrix of size* 2491*.*

The accuracy of the approximations obtained by `incexpm` is shown on the right in Figure 4.1. We assume `expm` as a reference, and measure the relative distance between these two approximations, i.e.,

$$\frac{\|\texttt{expm}(G_l) - \texttt{incexpm}(G_l)\|_F}{\|\texttt{expm}(G_l)\|_F},$$

at each iteration l (quantities smaller than the machine precision are set to u in Figure 4.1, for plotting purpose). One notes that the approximations of the adaptive strategy remain close to `expm` throughout the sequence of computations. An observed drop of the error down to u for this strategy corresponds to a restart in Algorithm 4.6; the approximation at this step is *exactly* the same as the one of `expm`. Even for the fixed scaling parameters 6 and 12, the obtained approximations are quite accurate.

4.3.2 Application to option pricing in polynomial models

We now apply `incexpm` in the framework of polynomial models to price European options. First, we remind that `incexpm` allows us to sequentially compute the moment sequence of polynomial jump-diffusions by means of the moment formula (2.6), see Section 4.1. Moreover, we recall that the matrix G_n representing the action of the generator $\mathcal{G}$ to the basis elements of $\mathrm{Pol}_n(E)$ exhibits the block triangular structure as in (4.1), where its size d_n is given by $d_n := N(n, d)$, the dimension of $\mathrm{Pol}_n(\mathbb{R}^d)$, and the sizes of the $n + 1$ square diagonal blocks are

$$1, d, \binom{1+d}{2}, \ldots, \binom{n+d-1}{n},$$

as highlighted in Chapter 2.

As discussed in Section 4.2, a norm estimate for G_n is instrumental for choosing a priori

the scaling parameter in the scaling and squaring method. In the following we provide such an estimate for the Jacobi model and for the Heston model. We consider these two models in the numerical experiments below.

Lemma 4.1. *Let G_n be the matrix representation of the generator $\mathcal{G}$ associated to the Jacobi model (see definition in Chapter 2) with respect to the monomial basis* (2.13) *of* $\mathrm{Pol}_n(E)$. *Define*

$$\alpha := \frac{\sigma(1 + v_{\min}v_{\max} + v_{\max} + v_{\min})}{2(\sqrt{v_{\max}} - \sqrt{v_{\min}})^2}.$$

Then the matrix 1-norm of G_n is bounded by

$$n(r + \kappa + \kappa\theta - \sigma\alpha) + \frac{1}{2}n^2(1 + |\rho|\alpha + 2\sigma\alpha). \tag{4.11}$$

Proof. As already mentioned in Chapter 2, the action of the generator $\mathcal{G}$ on a basis element $x^p v^q$ yields

$$\begin{aligned}\mathcal{G}x^p v^q =& x^{p-2}v^{q+1}p\frac{p-1}{2} - x^{p-1}v^{q+1}p\left(\frac{1}{2} + \frac{q\rho\sigma}{S}\right) + x^{p-1}v^q p\left(r + q\rho\sigma\frac{v_{\max} + v_{\min}}{S}\right)\\ &- x^{p-1}v^{q-1}\frac{pq\rho\sigma v_{\max}v_{\min}}{S} - x^p v^q q\left(\kappa + \frac{q-1}{2}\frac{\sigma^2}{S}\right)\\ &- x^p v^{q-2}q\frac{q-1}{2}\frac{\sigma^2 v_{\max}v_{\min}}{S} + x^p v^{q-1}q\left(\kappa\theta + \frac{q-1}{2}\sigma^2\frac{v_{\max} + v_{\min}}{S}\right),\end{aligned}$$

where $S := (\sqrt{v_{\max}} - \sqrt{v_{\min}})^2$. For the matrix 1-norm of G_n, one needs to determine the values of $(p, q) \in \mathcal{M} := \{(p, q) \in \mathbb{N}_0 \times \mathbb{N}_0 | p + q \leq n\}$ for which the 1-norm of the coordinate vector of $\mathcal{G}x^p v^q$ becomes maximal. First, we recall that the model parameters $\kappa, \theta, \sigma, r, v_{\min}, v_{\max}$ and S are nonnegative. Then, considering that p and q are also nonnegative, an upper bound of the 1-norm of the coordinate vector of $\mathcal{G}x^p v^q$ can be obtained by replacing ρ by $|\rho|$ as follows:

$$\begin{aligned}&p\frac{p-1}{2} + p\left(\frac{1}{2} + \frac{q|\rho|\sigma}{S}\right) + p\left(r + q|\rho|\sigma\frac{v_{\max} + v_{\min}}{S}\right) + \frac{pq|\rho|\sigma v_{\max}v_{\min}}{S}\\ &\quad + q\left(\kappa + \frac{q-1}{2}\frac{\sigma^2}{S}\right) + q\frac{q-1}{2}\frac{\sigma^2 v_{\max}v_{\min}}{S} + q\left(\kappa\theta + \frac{q-1}{2}\sigma^2\frac{v_{\max} + v_{\min}}{S}\right)\\ =& pr + q\kappa(\theta + 1) + \frac{1}{2}p^2 + 2pq|\rho|\alpha + q(q-1)\sigma\alpha\\ \leq& n(r + \kappa + \kappa\theta) + \frac{1}{2}n^2 + 2pq|\rho|\alpha + n(n-1)\sigma\alpha.\end{aligned}$$

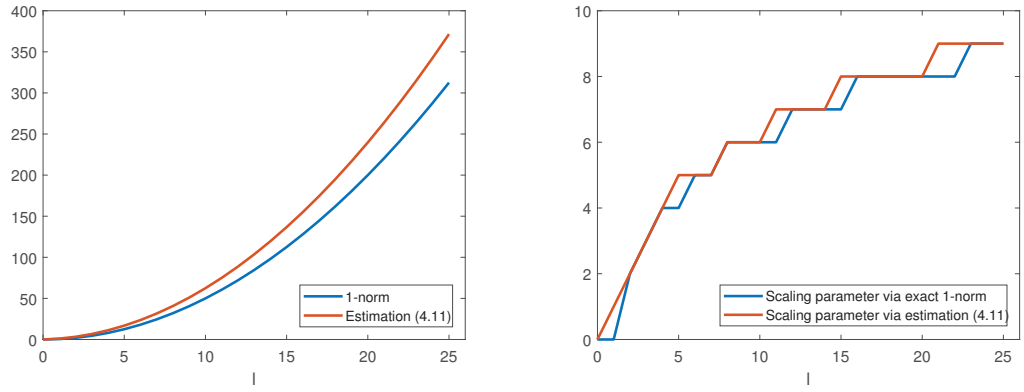

Figure 4.2 – Left: 1-norm and estimation (4.11) *of* G_l, $l = 0, 1, \cdots$. *Right: Corresponding scaling parameters.*

This completes the proof, noting that the maximum of pq on $\mathcal{M}$ is bounded by $n^2/4$ over $\mathcal{M}$. □

The result of Lemma 4.1 predicts that the norm of G_n grows, in general, quadratically. This prediction is confirmed numerically for parameter settings of practical relevance. Consider for example the set of model parameters

$$\kappa = 0.5, \quad \theta = 0.04, \quad \sigma = 0.15, \quad \rho = -0.5, \quad v_{\min} = 0.01, \quad v_{\max} = 1, \quad r = 0.$$

Figure 4.2 shows (left) the 1-norm of the matrices $G_l, l = 0, 1, \cdots$ and the corresponding estimation (4.11). On the right side, one can see the scaling parameter we would choose for all different values of l.

The following lemma extends the result of Lemma 4.1 to the Heston model.

Lemma 4.2. *Let* G_n *be the matrix representation of the generator* $\mathcal{G}$ *associated to the Heston model (see definition in Chapter 2) with respect to the monomial basis* (2.13) *of* $\mathrm{Pol}_n(E)$. *Then the matrix 1-norm of* G_n *is bounded by*

$$n(r + \kappa + \kappa\theta - \frac{\sigma^2}{2}) + \frac{1}{2}n^2(1 + |\rho|\frac{\sigma}{2} + \sigma^2).$$

Proof. The action of the generator $\mathcal{G}$ on a basis element $x^p v^q$ yields (see (2.12))

$$\begin{aligned}\mathcal{G}x^p v^q =& x^p v^{q-1}\Big(\kappa\theta q + \frac{1}{2}\sigma^2 q(q-1)\Big) + x^{p-1}v^q(rp + \rho\sigma qp) + \\ & x^{p-2}v^{q+1}\Big(\frac{1}{2}p(p-1)\Big) + x^p v^q(-kq) + x^{p-1}v^{q+1}\Big(-\frac{1}{2}p\Big).\end{aligned}$$

As done in the proof of Lemma 4.1, an upper bound of the 1-norm of G_n is obtained by considering the non-negativity of the model parameters and by replacing ρ by $|\rho|$. The required bound is then given by

$$\begin{aligned}&\kappa\theta q + \frac{1}{2}\sigma^2 q(q-1) + rp + |\rho|\sigma qp + \frac{1}{2}p(p-1) + kq + \frac{1}{2}p \\ \leq& \kappa\theta n + \frac{1}{2}\sigma^2 n(n-1) + rn + |\rho|\sigma pq + \frac{1}{2}n(n-1) + \kappa n + \frac{1}{2}n \\ \leq& n(r + \kappa + \kappa\theta - \frac{\sigma^2}{2}) + \frac{1}{2}n^2(1 + |\rho|\frac{\sigma}{2} + \sigma^2),\end{aligned}$$

where the last inequality follows the same reasoning as in Lemma 4.1. □

We now present the first option pricing technique that requires the incremental computation of the moment sequence. It is based on the polynomial expansion technique presented in Chapter 2, Section 2.3. We consider the pricing of European call options in the Jacobi model. The polynomial expansion method takes the particular form described in Chapter 3, Section 3.1.4. In particular the price is given by

$$\sum_{n\geq 0} f_n \ell_n, \tag{4.12}$$

where the coefficients f_n are computed recursively as in (3.43) and the coefficients ℓ_n are given by

$$\ell_n = H_n(X_0, V_0)e^{G_n T}\vec{h}_n, \tag{4.13}$$

for $\vec{h}_n$ containing the coordinates of the generalized Hermite polynomials (3.42) with respect to the monomial basis (2.13). Truncating the sum (4.12) after a finite number of terms allows us to obtain an approximation of the option price. However, in practice it is not known a priori how to choose the truncation level N in order to get a satisfactory accuracy. This issue can be solved by designing a heuristic algorithm to selecting N based on the absolute value of the summands. This procedure requires a sequential moment computation and it is presented in Algorithm 4.7, which makes use of `incexpm`,

Algorithm 4.7 Option pricing for the European call option in the Jacobi model

Input: Model and payoff parameters, tolerance ϵ
Output: Approximate option price
1: $n = 0$
2: Compute ℓ_0, f_0; set Price $= \ell_0 f_0$.
3: **while** $|\ell_n f_n| > \epsilon \cdot$ Price **do**
4: $\quad n = n + 1$
5: $\quad$ Compute $\exp(G_n T)$ using Algorithm 4.4.
6: $\quad$ Compute Hermite moment ℓ_n using (4.13).
7: $\quad$ Compute Fourier coefficient f_n as in (3.43).
8: $\quad$ Price $=$ Price $+ \ell_n f_n$;
9: **end while**

Algorithm	Time (s)	Rel. price error
`expm`	42.97	1.840e-03
`incexpm` (adaptive)	5.84	1.840e-03
`incexpm` ($s = 7$)	5.60	1.840e-03

Table 4.2 – Total run time and option price errors for the Jacobi model for $n = 61$.

Algorithm 4.5, for computing the required moments incrementally.

We now show results for computing option prices using Algorithm 4.7 for the set of parameters

$$v_0 = 0.04, \quad x_0 = 0, \quad \sigma_w = 0.5, \quad \mu_w = 0, \quad \kappa = 0.5, \quad \theta = 0.04, \quad \sigma = 0.15,$$
$$\rho = -0.5, \quad v_{\min} = 0.01, \quad v_{\max} = 1, \quad r = 0, \quad T = 1/4, \quad k = \log(1.1).$$

We use the tolerance $\epsilon = 10^{-3}$ for stopping Algorithm 4.7.

We explore the use of different algorithms for the computation of the matrix exponentials in line 5 of Algorithm 4.7: `incexpm` with adaptive scaling, `incexpm` with fixed scaling parameter $s = 7$ (corresponding to the upper bound from Lemma 4.1 for $n = 60$), and `expm`. Similar to Figure 4.1, the observed cumulative run times and errors are shown in Figure 4.3. Again, `incexpm` is observed to be significantly faster than `expm` (except for small matrix sizes) while delivering the same level of accuracy. Both `incexpm` run times are also close to the run time of MATLAB's `expm` applied only to the final matrix $G_n T$ (4.64s).

Table 4.2 displays the impact of the different algorithms on the overall Algorithm 4.7, in

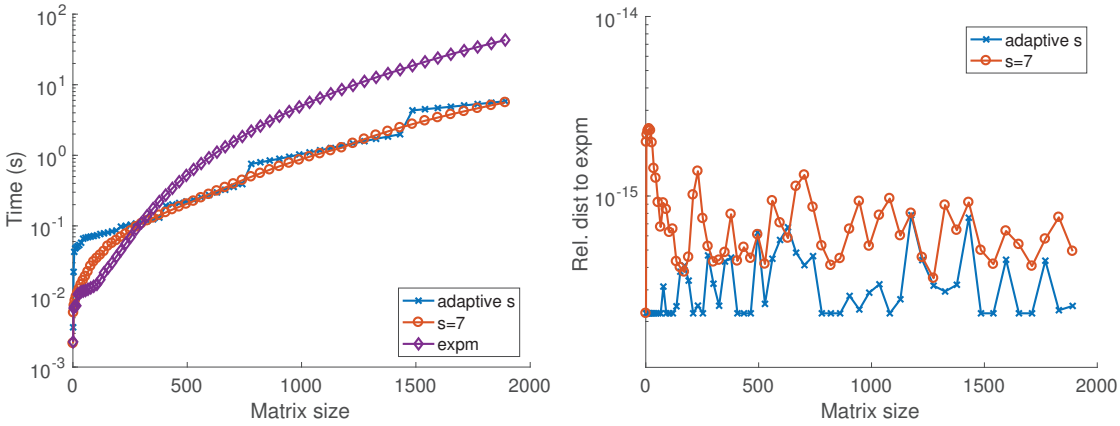

Figure 4.3 – Comparison of `incexpm` *and* `expm` *for the block upper triangular matrices arising in the context of the Jacobi model in Algorithm 4.7.* Left: *Cumulative run time for computing the leading portions.* Right: *Relative error of* `incexpm` *w.r.t.* `expm`*.*

terms of execution time and accuracy. Concerning accuracy, we computed the relative error with respect to a reference option price computed by considering a truncation order $N = 100$. It can be observed that there is no difference in accuracy for the three algorithms.

We now consider a second option pricing algorithm that requires an incremental computation of the moment sequence: the black box algorithm for European option pricing developed in Chapter 3, Section 3.1.5, based on the computation of lower and upper bounds of the option price. In particular we compute the price of a European call option with payoff parameters $k = -0.1$ and $T = 1/12$ in the Heston model by running the Algorithm 3.2. We set the tolerance for the gap to $\epsilon = 10^{-5}$ and we consider the model parameters

$$v_0 = 0.01, \quad x_0 = 0, \quad \kappa = 0.5, \quad \theta = 0.04, \quad \sigma = 0.15, \quad \rho = 0.5, \quad r = 0.$$

The solution of the optimization problems (3.5) and (3.6) (Step 5 of the algorithm) are computed using the SDP approach explained in Chapter 3, Section 3.1.3. We employ `incexpm` in Step 4 to update the vector of moments. In particular, we combine it with the adaptive scaling strategy and with fixed scaling parameter $s = 6$ (obtained via the Lemma 4.2 with $n = 40$). We compare it with `expm`. We show the impact of the different algorithms on the overall Algorithm 3.2 in Table 4.3. We list the total run times to compute the sequence of matrix exponentials and the relative price errors computed with

Algorithm	Time (s)	Rel. price error
`expm`	3.09	5.421e-04
`incexpm` (adaptive)	0.78	5.421e-04
`incexpm` ($s = 6$)	1.06	5.421e-04

Table 4.3 – Total run time and option price errors for the Heston model for $n = 40$.

respect to a reference price obtained via the Fourier pricing technique as in [66] and reviewed in Chapter 1.

The algorithm always stops after reaching maximal moment order $n = 40$. Again, we see that `incexpm` outperforms `expm` in terms of total run time by keeping the same accuracy, for both choices of the scaling parameter. It is worth it to mention that, compared to the previous example, there is a notable difference. The evaluation of the stopping criterion requires the numerical solution of the two SDPs, which completely dominates the time needed for the computation of the matrix exponentials. Therefore, while in the Jacobi model example `incexpm` has a big impact on the overall pricing Algorithm 4.7 in terms of total run time, in this last example the dominating cost of the Algorithm 3.2 is concentrated in Step 5.

We conclude with a short remark about a computational detail of `incexpm`.

Remark 4.3. *While the particular structure of the diagonal blocks is taken into account automatically by `expm` and `incexpm` when computing the LU decompositions of the diagonal blocks, it is not so easy to benefit from the sparsity. Starting from sparse matrix arithmetic, the matrix quickly becomes denser during the evaluation of the initial rational approximation, and in particular during the squaring phase. In all our numerical experiments we used a dense matrix representation throughout.*

4.4 Conclusion

We have presented techniques for scaling and squaring algorithms that allow for the incremental computation of block triangular matrix exponentials. We combined these techniques with an adaptive scaling strategy that allows for both fast and accurate computation of each matrix exponential in this sequence (Algorithm 4.6).

The developed algorithm `incexpm` has been effectively employed for European option

pricing in polynomial models. In this framework, we have considered two pricing algorithms that require the incremental computation of the moment sequence. Numerical experiments have shown that the running time can be reduced by using `incexpm` with both the adaptive scaling strategy and with a fixed scaling parameter, while maintaining the same accuracy. Our algorithm is therefore effective. Lastly, we have seen that the fixed scaling parameter can be determined through the estimation techniques in Lemmas 4.1 and 4.2, which can be in principle extended to any polynomial model.

5 A complexity reduction technique for high-dimensional option pricing

In this chapter we present a complexity reduction technique for high-dimensional option pricing. Our approach is based on the work by Gass et al. [46], who propose a complexity reduction technique for parametric option pricing based on Chebyshev interpolation. There, the idea consists of using the classical Chebyshev interpolation on the space of model and payoff parameters to increase the efficiency in computing option prices, while maintaining the required accuracy. As the number of treated parameters increases, however, this method is affected by the curse of dimensionality. We extend this approach to treat parameter spaces of high dimensions by exploiting low-rank structures. The core idea of our method is to express the tensorized interpolation in the tensor train format and to develop an efficient way, based on tensor completion, to approximate the interpolation coefficients.

Our method is designed to tackle the high dimensionality of parameter spaces in the general option pricing problem. Given an arbitrary option pricing algorithm for a certain stochastic model, this method can be applied to reduce its complexity and increase the efficiency. As a consequence, moment-based option pricing techniques, as the one developed in Chapter 3, and polynomial models are a special case to which it can be applied.

Generally speaking, treating high dimensionality is one of the main challenges in the development of computational methods for solving problems arising in finance. We continue this chapter by reviewing high dimensionality in finance, Section 5.1. Then, in Section 5.2 we give an overview of the different steps of our method, and we provide a

detailed explanation in Section 5.3. Finally, we show its effectiveness in Section 5.4 where we present two numerical experiments. This chapter is mostly based on the preprint [51].

5.1 Introduction to high dimensionality in finance

Financial problems are, by their nature, multi- and high-dimensional, because a large number of risk factors contribute to the prices of each financial asset. Moreover, the banking, insurance and hedge fund industry draws on investments in large portfolios. The interdependencies of both the risk factors and the assets make basic computational tasks such as model calibration, pricing, and hedging as well as more global tasks such as uncertainty quantification, risk assessment and capital reserve calculation computationally extremely challenging, see for instance [10].

Automatic and high-speed trading challenge the computational methods in that the results need to be available fast and with minimal storage requirement. Moreover, we observe rising regulatory requirements. On the one hand, more realistic modeling demands more prudent considerations, which leads to rising computational complexity. On the other hand, the availability of requested performance characteristics is expected to be delivered within shorter periods of time. This poses a high challenge for traditional approaches, which typically suffer from low convergence rates in higher dimensions, see for instance [22, 32].

For the reasons explained above, the development of efficient computational methods for high-dimensional problems in finance is an utmost active field of research in both academia and industry. For example, further developments of the Monte Carlo method have been very successfully applied to financial problems; we refer to [90, 49] for the quasi Monte Carlo method and to [48] for the multilevel Monte Carlo method. Besides stochastic integration, deterministic numerical integration has been exploited using sparse grid techniques, see [56, 71, 11]. Also PDE methods have been extended to multivariate problems in finance. For instance using operator splitting methods as in [76], principal component analysis and expansions as in [103], and wavelet compression techniques proposed in [95, 69, 70].

Exploiting the particular structure of a problem, *complexity reduction techniques* exhibit great potential to save run-time and storage capacity while maintaining the required

accuracy. One way to reduce the complexity of a certain approach is to look at the parameter dependency of the problem. Let us consider the example of option pricing. The price of an option can be seen as a function of model and payoff parameters and this parameter dependency can be exploited as follows. First, the price is computed in a certain set of parameters during an *offline phase*. Then, the procedure *learns* from the computed prices to get insights on the structure of the function "parameters $\mapsto$ Price". Finally, this knowledge is exploited to compute (an approximation of) the prices for a *new* arbitrary set of parameters during an *online* phase. The idea is to make the procedure work a lot during the offline phase, which is the most expensive part of the approach, in order to collect as much information as possible. Similar approaches are used in the field of machine learning, where complex models are *trained* during an offline phase using available data, see e.g. [92] for a machine learning approach to option pricing.

The big advantage of such an approach is that the offline phase can be performed at any time, and once finished, the information can be stored and used whenever needed during the online phase to efficiently compute option prices. Such a structure is very convenient, for example, when model calibration has to be performed. Indeed, this task requires a lot of price evaluations and this can be efficiently done in the offline-online framework described above. In this chapter we propose a complexity reduction technique for computing the price of options that depend on a large set of parameters. The idea is to use the direct interpolation of multivariate functions which enables us to exploit parameter dependency and to build an offline-online procedure.

5.2 Introduction to the method

Our starting point is the tensorized Chebyshev interpolation of conditional expectations in the parameter and state space, as introduced in [46]. Having observed for a large set of applications that these functions are highly regular, admitting sensitivities of high order or even being analytic, and that the domain of interest can be restricted to a hyperrectangular, Chebyshev interpolation is a promising choice: Its convergence is superalgebraic for multivariate analytic functions, its implementation is numerically stable, and the coefficients are simply given by a linear transformation of the function values at the nodal points. In this work we exploit this favorable structure further for high dimensionality. In passing, we point out that, while we choose Chebyshev interpolation for the reasons listed in this paragraph, the technique presented in this chapter extends

to other tensorized interpolation techniques.

The basis of our approach is the following. In an *offline phase*, the price as function of parameters $\mathbf{p} \in [-1,1]^d$, $\mathbf{p} \mapsto \mathsf{Price}^{\mathbf{p}}$ is evaluated at selected parameter samples $\mathbf{p}$ to prepare an approximation by tensorized Chebyshev polynomials $T_{j_1,\dots,j_d}$ with precomputed Fourier coefficients $c_{j_1,\dots,j_d}$, as follows,

$$\mathsf{Price}^{\mathbf{p}} \approx \sum_{j_1=0}^{n_1} \dots \sum_{j_d=0}^{n_d} c_{j_1,\dots,j_d} T_{j_1,\dots,j_d}(\mathbf{p}). \tag{5.1}$$

To evaluate the function in the *online phase*, only the multivariate polynomials on the right-hand side need to be evaluated. However, implementing (5.1) in a straightforward manner exposes the method to the curse of dimensionality in both the offline and the online phase: In the offline phase, the prices need to be evaluated on a tensorized grid of Chebyshev nodes, amounting to $\mathcal{O}(n^d)$ parameter samples when n nodes are required for each parameter. This is computationally costly, especially if the underlying pricing method is already computationally demanding. In the online phase alike, $\mathcal{O}(n^d)$ operations are needed for evaluating the approximating multivariate polynomial. Even for a number as low as $n = 3$, corresponding to quadratic polynomials, a problem with $d = 20$ parameters becomes infeasible.

One approach to breaking the curse of dimensionality that has already proven effective in a number of areas is to exploit low-rank structures of high-dimensional tensors; see [55, 59, 78] and the references therein. These techniques reduce, sometimes dramatically, memory requirements and the cost of operating with tensors. In the context of parametric PDEs, low-rank tensor structures have been successfully exploited in, e.g., [6, 9, 79, 86, 112]. As option prices are characterized as solutions of parabolic PDEs, this gives hope that low-rank structures can be exploited in finance as well. The following questions arise:

Can we detect low-rank structures for the problem of form (5.1)*?* Existing theoretical studies only provide partial answers to this question, either not reflecting the observed effectiveness of low-rank techniques or being limited to rather specific function classes; see [29, 59, 107] for examples. We therefore approach the question from an experimental perspective and analyze examples of different nature and different dimensionality in Section 5.4. The results clearly indicate an approximate low-rank structure of the tensor $\mathcal{P}$ containing the prices evaluated at the nodes of the tensorized Chebyshev grid. In

the specific case of the interpolation of American option prices in the Heston model in five parameters we can explicitly compare the full tensor $\mathcal{P}$ with the one resulting from low-rank approximation. We perform this comparison in Section 5.4.1, which confirms the low-rank structure of $\mathcal{P}$. In Section 5.4.2 we consider prices of basket options in the Black-Scholes model with up to 25 underlyings and interpolate in the initial values of the underlyings. Although the resulting full tensor $\mathcal{P}$ is too large to be explicitly computed and compared with, we provide a structural analysis that explains why $\mathcal{P}$ is expected to exhibit low-rank structure.

How can we exploit low-rank structures for the problem of form (5.1)*?* Expressing the problem in a tensor format reveals that exploiting the tensor structure itself (even without low-rank structure) leads to a considerable efficiency gain in both the offline and the online phase. Next, we explore existing low-rank tensor techniques. In order to efficiently exploit these techniques for problem (5.1), we need to introduce several new components resulting in the new method. We detail these steps below.

In order to construct the interpolation coefficients $c_{j_1,\dots,j_d}$ in the offline phase, it is first required to compute or approximate all values of the tensor $\mathcal{P}$, containing the prices in the tensorized Chebyshev grid. Evaluating $\mathcal{P}$ explicitly is too costly for larger d, especially when the underlying pricing procedure is computationally expensive. Instead we only compute part of the entries of $\mathcal{P}$ and then need to deal with an incomplete tensor. This leads us to the following first step:

1. We start by computing the prices for a small portion of the Chebyshev grid points only. Then, we adapt a *completion algorithm* (in Section 5.3.4) which allows us to approximate the tensor of prices for the complete Chebyshev grid by fitting tensors of pre-specified low rank to the provided data points. As it is not reasonable to assume a priori knowledge of low-rank structure, the completion procedure needs to be combined with an *adaptive rank and sampling strategy*. Specifically, we repeat the process of adding new samples and increasing the pre-specified rank until an adequate stopping criterion is fulfilled. This completion algorithm is designed to work with tensors built and stored in the tensor train (TT) format.

With the low-rank approximation of the tensor $\mathcal{P}$ in the TT format at hand, we can then approximate efficiently the Fourier coefficients $c_{j_1,\dots,j_d}$. This is the last step of the offline phase:

2. The computation of the tensor $\mathcal{C}$, containing the Fourier coefficients $c_{j_1,\dots,j_d}$, is computed by a sequence of d tensor-matrix multiplications. The particular structure of the involved matrices facilitates the use of the fast Fourier transform, leading to a complexity of $\mathcal{O}(dnr^2 \log(n))$, where r is determined by the ranks of $\mathcal{P}$. This step is explained in Section 5.3.5.

Suppose now that, in the online phase, we want to compute the interpolated price (5.1) for a new set of parameter samples. Given the tensor $\mathcal{C}$ in the TT format, the evaluation of (5.1) for a price $\mathbf{p}$ is performed efficiently as follows:

3. First, each of the Chebyshev polynomials involved in the tensorized Chebyshev basis is evaluated in $\mathbf{p}$. It turns out that (5.1) can be viewed as inner product between $\mathcal{C}$ and a rank-one tensor. Thanks to the TT format, the complexity of computing this inner product is $\mathcal{O}(dnr^2)$; see Section 5.3.3. As long as r is reasonably small, this compares favorably with the $\mathcal{O}(n^d)$ operations needed by the standard approach.

In Section 5.4, we test the performance of the new method for two different option pricing problems, the interpolation of

- American option prices in the Heston model in $d = 5$ parameters, and of
- prices of basket options in the Black-Scholes model in up to $d = 25$ underlyings.

At comparable accuracy, the interpolation in American option prices reveals a promising gain in efficiency when compared to an ADI-based PDE solver. The efficiency gain for the basket option prices is shown in comparison to a Monte Carlo simulation with variance reduction.

5.3 TT format and tensor completion for Chebyshev interpolation

This section describes the methodology proposed in this work. We start with recalling the tensorized Chebyshev interpolation method from [46]. After defining the concept of low-rank approximation and the TT format [101], we present and extend the tensor

completion approach from [112]. Finally, we explain how to combine these algorithms in order to efficiently price parametric options for a large number of parameters.

5.3.1 Chebyshev interpolation for parametric option pricing

We consider an option price that depends on a vector of d parameters $\mathbf{p}$ contained in $[-1,1]^d$; general hyperrectangular parameter domains can be addressed by a suitable affine transformation. The basic idea developed in [46] consists of using the tensorized Chebyshev interpolation in the parameters (model and payoff parameters) to increase the efficiency of computing option prices, while maintaining satisfactory accuracy. Writing $\mathsf{Price}^{\mathbf{p}}$ for the price evaluated in $\mathbf{p}$, the Chebyshev interpolation of order $\overline{\mathbf{n}} := (n_1, \ldots, n_d)$ (with $n_i \in \mathbb{N}_0$) aims at interpolating $\mathsf{Price}^{\mathbf{p}}$ in the d-dimensional tensorized Chebyshev grid defined as

$$q_{k_1,\ldots,k_d} := (q_{k_1}, \ldots, q_{k_d}), \quad \text{for} \quad k_i = 0, \ldots, n_i \ \text{ and } \ i = 1, \ldots, d, \tag{5.2}$$

where

$$q_{k_i} := \cos\left(\pi \frac{k_i}{n_i}\right)$$

are the *Chebyshev nodes of the second kind*, by means of basis functions that are constructed from *Chebyshev polynomials of the first kind* as

$$T_{j_1,\ldots,j_d}(\mathbf{p}) := \prod_{i=1}^{d} T_{j_i}(p_i), \quad T_{j_i}(p_i) = \cos(j_i \arccos(p_i)), \tag{5.3}$$

for $j_\ell = 0, \ldots, n_\ell$, with $\ell = 1, \ldots, d$. By extending standard results on the one-dimensional Chebyshev interpolation (see e.g. [116]), one can show (see also Lemma 5.1 below) that the interpolating polynomial that solves the tensorized Chebyshev interpolation problem is given by

$$I_{\overline{\mathbf{n}}}(\mathsf{Price}^{(\cdot)})(\mathbf{p}) = \sum_{j_1=0}^{n_1} \cdots \sum_{j_d=0}^{n_d} c_{j_1,\ldots,j_d} T_{j_1,\ldots,j_d}(\mathbf{p}), \tag{5.4}$$

where the coefficients $c_{j_1,\ldots,j_d}$ are defined as

$$c_{j_1,\ldots,j_d} = \left(\prod_{i=1}^{d} \frac{2^{\mathbb{1}_{n_i > j_i > 0}}}{n_i}\right) \sum_{k_1=0}^{n_1}{}'' \cdots \sum_{k_d=0}^{n_d}{}'' \mathcal{P}(k_1, \ldots, k_d) \prod_{i=1}^{d} \cos\left(j_i \pi \frac{k_i}{n_i}\right). \tag{5.5}$$

The symbol $\sum''$ in (5.5) indicates that the first and the last summand are halved, and the tensor $\mathcal{P}$ contains the prices on the tensorized Chebyshev grid:

$$\mathcal{P}(k_1, \ldots, k_d) = \mathsf{Price}^{q_{k_1,\ldots,k_d}}, \tag{5.6}$$

for $k_i = 0, \ldots, n_i$ and $i = 1, \ldots, d$. A convergence analysis of the tensorized Chebyshev interpolation in the setting of option pricing is given in [46].

In the following lemma we rigorously show that the interpolation problem is indeed solved by the interpolating polynomial $I_{\overline{\mathbf{n}}}(\mathsf{Price}^{(\cdot)})(\mathbf{p})$.

Lemma 5.1. *Let $q_{\alpha_1,\ldots,\alpha_d}$ be an arbitrary Chebyshev node that belongs to the tensorized grid* (5.2)*. Then,*

$$I_{\overline{\mathbf{n}}}(\mathsf{Price}^{(\cdot)})(q_{\alpha_1,\ldots,\alpha_d}) = \mathsf{Price}^{q_{\alpha_1,\ldots,\alpha_d}}.$$

Proof. For the sake of clarity, we first show the statement for the one-dimensional case $d = 1$. In this case, the interpolating polynomial evaluated in the Chebyshev node q_α takes the form

$$\begin{aligned} I_n(\mathsf{Price}^{(\cdot)})(q_\alpha) = \sum_{j=0}^{n} c_j T_j(q_\alpha) = \sum_{j=0}^{n} \left(\frac{2^{\mathbb{1}_{n>j>0}}}{n} \sum_{k=0}^{n}{}'' \mathcal{P}(k) \cos\left(j\pi \frac{k}{n}\right) \right) T_j(q_\alpha) = \\ \frac{2}{n} \sum_{j=0}^{n}{}'' \left(\sum_{k=0}^{n}{}'' \mathcal{P}(k) T_j(q_k) \right) T_j(q_\alpha) = \frac{2}{n} \sum_{k=0}^{n}{}'' \mathcal{P}(k) \left(\sum_{j=0}^{n}{}'' T_j(q_k) T_j(q_\alpha) \right). \end{aligned} \tag{5.7}$$

It follows straightforward from the definition of T_j and q_k that the equality

$$T_j(q_k) = T_k(q_j) \tag{5.8}$$

holds for all $k, j = 0, \ldots, n$. Moreover, the Chebyshev polynomials satisfy the following discrete orthogonality condition (see e.g. [47])

$$\sum_{j=0}^{n}{}'' T_k(q_j) T_\alpha(q_j) = \delta_{k,\alpha} \begin{cases} n & \text{for } \alpha = 0 \text{ or } \alpha = n \\ \frac{n}{2} & 1 \le \alpha \le n-1 \end{cases}. \tag{5.9}$$

By using the Properties (5.8) and (5.9), the Equation (5.7) can be further simplified

obtaining

$$\frac{2}{n}\sum_{k=0}^{n}{}''\,\mathcal{P}(k)\Big(\sum_{j=0}^{n}{}''\,T_j(q_k)T_j(q_\alpha)\Big)=\frac{2}{n}\sum_{k=0}^{n}{}''\,\mathcal{P}(k)\Big(\sum_{j=0}^{n}{}''\,T_k(q_j)T_\alpha(q_j)\Big)=\mathcal{P}(\alpha)=\mathsf{Price}^{q_\alpha},$$

which concludes the proof for the case $d=1$. For an arbitrary $d>1$, the Property (5.8) takes the form

$$T_{j_1,\dots,j_d}(q_{k_1,\dots,k_d})=\prod_{i=1}^{d}T_{j_i}(q_{k_i})=\prod_{i=1}^{d}T_{k_i}(q_{j_i})=T_{k_1,\dots,k_d}(q_{j_1,\dots,j_d}),\tag{5.10}$$

for all $k_i, j_i=0,\dots,n_i$, while the Property (5.9) becomes

$$\sum_{j_1=0}^{n_1}{}''\cdots\sum_{j_d=0}^{n_d}{}''\Big(\prod_{i=1}^{d}T_{k_i}(q_{j_i})\prod_{\ell=1}^{d}T_{\alpha_\ell}(q_{j_\ell})\Big)=\begin{cases}\prod_{i=1}^{d}\frac{n_i}{2^{\mathbb{1}_{n_i>k_i>0}}} & \text{if } \alpha_i=k_i \text{ for all } i\\ 0 & \text{otherwise}\end{cases}.\tag{5.11}$$

By exploiting (5.10) and (5.11) the statement can be shown for an arbitrary dimension d by rewriting $I_{\overline{\mathbf{n}}}(\mathsf{Price}^{(\cdot)})(q_{\alpha_1,\dots,\alpha_d})$ as

$$\begin{aligned}
&\sum_{j_1=0}^{n_1}\cdots\sum_{j_d=0}^{n_d}c_{j_1,\dots,j_d}T_{j_1,\dots,j_d}(q_{\alpha_1,\dots,\alpha_d})=\\
&\sum_{j_1=0}^{n_1}\cdots\sum_{j_d=0}^{n_d}\Bigg(\Bigg(\prod_{i=1}^{d}\frac{2^{\mathbb{1}_{n_i>j_i>0}}}{n_i}\Bigg)\sum_{k_1=0}^{n_1}{}''\cdots\sum_{k_d=0}^{n_d}{}''\,\mathcal{P}(\mathbf{k})\prod_{i=1}^{d}\cos\Big(j_i\pi\frac{k_i}{n_i}\Big)\Bigg)T_{j_1,\dots,j_d}(q_{\alpha_1,\dots,\alpha_d})=\\
&\frac{2^d}{n_1\cdots n_d}\sum_{k_1=0}^{n_1}{}''\cdots\sum_{k_d=0}^{n_d}{}''\,\mathcal{P}(\mathbf{k})\sum_{j_1=0}^{n_1}{}''\cdots\sum_{j_d=0}^{n_d}{}''\Big(\prod_{i=1}^{d}\cos\Big(j_i\pi\frac{k_i}{n_i}\Big)\prod_{\ell=1}^{d}T_{j_\ell}(q_{\alpha_\ell})\Big)=\\
&\frac{2^d}{n_1\cdots n_d}\sum_{k_1=0}^{n_1}{}''\cdots\sum_{k_d=0}^{n_d}{}''\,\mathcal{P}(\mathbf{k})\sum_{j_1=0}^{n_1}{}''\cdots\sum_{j_d=0}^{n_d}{}''\Big(\prod_{i=1}^{d}T_{k_i}(q_{j_i})\prod_{\ell=1}^{d}T_{\alpha_\ell}(q_{j_\ell})\Big)=\mathcal{P}(\alpha_1\dots,\alpha_d)=\\
&\mathsf{Price}^{q_{\alpha_1,\dots,\alpha_d}},
\end{aligned}$$

where we used the notation $\mathbf{k}:=(k_1,\dots,k_d)$ throughout. This concludes the proof. □

The tensor $\mathcal{P}$ in equation (5.5) is of order d and size $(n_1+1)\times\cdots\times(n_d+1)$. The interpolation procedure first requires to compute each entry of this tensor with the reference method. This becomes expensive when the interpolation order and the dimension d increase. We will use tensor completion to lower this cost.

Remark 5.2 (Choice of interpolation order)**.** *In our numerical experiments, the interpolation order* $\overline{\mathbf{n}}$ *is chosen a priori for simplicity. However, this choice can be made adaptively as explained in [64] for the case* $d = 3$*.*

5.3.2 Low-rank matrix approximation

In this section we start introducing the notion of low-rank approximation. We begin by focusing on matrices and we consider tensors in the next section. Let $A \in \mathbb{R}^{m\times n}$ be a matrix and assume that $m \geq n$. Its singular value decomposition (SVD) is given by

$$A = U\Sigma V^T, \quad \text{with} \quad \Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & 0 & \end{bmatrix} \in \mathbb{R}^{m\times n}, \tag{5.12}$$

for two orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$, and where the scalars $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are the *singular values* of A. The number r of positive (nonzero) singular values is equal to the *rank* of A which we denote by $\operatorname{rank}(A)$. The matrices U and V can be partitioned in columns as

$$U = \left[u_1 \middle| \ldots \middle| u_n \middle| u_{n+1} \middle| \ldots \middle| u_m\right], \quad V = \left[v_1 \middle| \ldots \middle| v_n\right],$$

and the vectors $u_1, \ldots, u_n \in \mathbb{R}^m$ are called *left singular vectors*. Similarly, the vectors $v_1, \ldots, v_n \in \mathbb{R}^n$ are called *right singular vectors*. If the rank of A is strictly smaller than n, then the singular values $\sigma_{r+1}, \ldots, \sigma_n$ are equal to 0. In this case, the matrix A can be decomposed according to the so-called *reduced SVD*

$$A = U_r\Sigma_r V_r^T,$$

where

$$U_r = \left[u_1 \middle| \ldots \middle| u_r\right] \in \mathbb{R}^{m\times r}, \quad V_r = \left[v_1 \middle| \ldots \middle| v_r\right] \in \mathbb{R}^{n\times r}, \quad \Sigma_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r).$$

If r is much smaller than m and n we say that A has *low rank*.

Let $A \in \mathbb{R}^{m\times n}$ be an arbitrary matrix. We aim at approximating A with a matrix of a

certain prescribed rank $r \leq \operatorname{rank}(A)$. We consider the SVD (5.12) and we define the *rank r-truncation* of A as the matrix

$$\mathcal{T}_r(A) := U_r \Sigma_r V_r^T, \tag{5.13}$$

where

$$U_r := \left[u_1 \middle| \dots \middle| u_r\right], \quad \Sigma_r := \operatorname{diag}(\sigma_1, \dots, \sigma_r), \quad V_r := \left[v_1 \middle| \dots \middle| v_r\right].$$

The matrix $\mathcal{T}_r(A)$ has rank r and represents a *rank-r approximation* of A. The following theorem (see e.g. [73]) states that this is the best rank-r approximation in the Frobenius norm and in the 2-norm, given by

$$\|A\|_F = \sqrt{\operatorname{Tr}(A^T A)} = \sqrt{\sigma_1^2 + \dots + \sigma_n^2}, \quad \|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2 = \sigma_1$$

respectively.

Theorem 5.3. *Consider the rank r-truncation $\mathcal{T}_r(A)$ defined in (5.13) for $A \in \mathbb{R}^{m \times n}$ with $0 \leq r \leq n \leq m$. Then*

$$\min\left\{\|A - B\| : B \in \mathbb{R}^{m \times n}, B \text{ has rank at most } r\right\} = \|A - \mathcal{T}_r(A)\|$$

holds for both the Frobenius norm and the 2-norm.

By construction, the error $\|A - \mathcal{T}_r(A)\|$ is explicitly given by

$$\|A - \mathcal{T}_r(A)\|_2 = \sigma_{r+1}, \quad \|A - \mathcal{T}_r(A)\|_F = \sqrt{\sigma_{r+1}^2 + \dots + \sigma_n^2}.$$

If there exists $r \ll m, n$ such that the singular values $\sigma_{r+1}, \dots, \sigma_n$ are small, the truncation $\mathcal{T}_r(A)$ yields a good rank-r approximation for A and we say that A *has a low-rank structure*. This allows to reduce the storage complexity. In fact, storing the low-rank factorization (5.13) requires $r(m+n)$ units of storage, while storing A in full format requires mn units of storage. If $r \ll m, n$ this represents a big storage reduction. Also, the matrix operations can be performed more efficiently. For example, the matrix-vector multiplication $\mathcal{T}_r(A)x$, for a vector $x \in \mathbb{R}^n$, has a computational cost $\mathcal{O}(r(m+n))$, while performing Ax in full format costs $\mathcal{O}(nm)$ operations. Again, if $r \ll m, n$ this leads to a complexity reduction of the operation.

The concept of (matrix)-decomposition and low-rank approximation can be extended

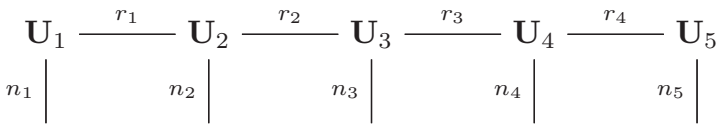

Figure 5.1 – Tensor network diagram of the TT decomposition of a tensor of order $d = 5$.

further to tensors, which are d-dimensional arrays. More precisely, a tensor of order d and size $n_1 \times n_d \times \cdots \times n_d$ is a d-dimensional array with entries

$$\mathcal{X}(i_1, i_2, \cdots, i_d), \quad i_\mu \in \{1, \ldots, n_\mu\} \text{ for } \mu = 1, \ldots, d.$$

Matrices are tensors of order 2. While the rank of a matrix is uniquely defined and the SVD allows to compute the best low-rank approximations, the situation is more complex with tensors. In fact, they can be decomposed in different ways, leading to different definitions of tensor-rank and to different *tensor decompositions*. Examples are the Canonical Polyadic Decomposition and the Tucker decomposition, see e.g. [82]. An other tensor decomposition is the tensor train (TT) decomposition, introduced in [101]. In this chapter, we work with tensors in the TT format and their corresponding low-rank approximations, which we present next.

5.3.3 TT format

For recalling the TT format introduced in [101], we consider a general tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ of order d. The tensor $\mathcal{X}$ is in the *TT decomposition* if every entry $\mathcal{X}(i_1, i_2, \cdots, i_d)$ can be expressed as

$$\mathcal{X}(i_1, i_2, \cdots, i_d) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_{d-1}=1}^{r_{d-1}} \mathbf{U}_1(1, i_1, k_1)\mathbf{U}_2(k_1, i_2, k_2) \cdots \mathbf{U}_d(k_{d-1}, i_d, 1), \quad (5.14)$$

for some third order tensors $\mathbf{U}_\mu$ of size $r_{\mu-1} \times n_\mu \times r_\mu$, for $\mu = 1, \cdots, d$, the so-called *TT cores* of $\mathcal{X}$. Tensors in the TT format can be represented by a tensor network diagram [100]. For example, Figure 5.1 illustrates a tensor of order 5 in the TT format. As shown in [101], the integer tuple $(r_0, r_1, \cdots, r_d)$, where we formally set $r_0 = r_d = 1$, is related to the (matrix-)ranks of the so-called *unfoldings* of $\mathcal{X}$, which we define next. For each

$\mu = 1, \ldots, d-1$, the entries of $\mathcal{X}$ can be rearranged into a matrix

$$X^{<\mu>} \in \mathbb{R}^{(n_1 n_2 \cdots n_\mu) \times (n_{\mu+1} \cdots n_d)},$$

which is called the *μth unfolding* of $\mathcal{X}$. For this purpose, the first μ indices of $\mathcal{X}$ are merged into the row index and the last $n - \mu$ indices into a column index. Formally, the index map is given by

$$\iota : \mathbb{N}^d \to \mathbb{N} \times \mathbb{N}, \quad \iota(i_1, \ldots, i_d) = (i_{\text{row}}, i_{\text{col}}),$$
$$i_{\text{row}} = 1 + \sum_{\ell=1}^{\mu} (i_\ell - 1) \prod_{k=1}^{\ell-1} n_k, \quad i_{\text{col}} = 1 + \sum_{\ell=\mu+1}^{d} (i_\ell - 1) \prod_{k=\mu+1}^{\ell-1} n_k.$$

The μth unfolding of $\mathcal{X}$ can be computed in MATLAB with the `reshape` command as

$$X^{<\mu>} = \texttt{reshape}\Big(\mathcal{X}, \Big[\prod_{i=1}^{\mu} n_i, \prod_{i=\mu+1}^{d} n_i\Big]\Big).$$

The Theorem 2.1 in [101] shows that the tuple $(r_0, r_1, \cdots, r_d)$ can be chosen to be[1]

$$(r_0, r_1, \cdots, r_d) = (1, \text{rank}(X^{<1>}), \cdots, \text{rank}(X^{<d-1>}), 1). \tag{5.15}$$

We refer to (5.15) as the *TT ranks of* $\mathcal{X}$ and we denote it by $\text{rank}_{\text{TT}}(\mathcal{X})$. The proof of [101, Theorem 2.1] is constructive and gives an algorithm ([101, Algorithm 1]) to compute the TT decomposition (5.14) with TT ranks (5.15) of a general tensor $\mathcal{X}$. This algorithm is referred to as *TT-SVD* and it uses a sequence of $d-1$ SVDs of some auxiliary matrices to compute the TT cores of $\mathcal{X}$. We give the pseudo-code of the TT-SVD in Algorithm 5.1. Note that in the algorithm we write the SVD (5.12) in the form $A = U\tilde{V}$ where $U \in \mathbb{R}^{m \times r}$ and $\tilde{V} := \Sigma V^T \in \mathbb{R}^{r \times n}$, for $r = \text{rank}(A)$.

We now proceed as in the case of matrices by defining the concept of low rank and low-rank approximation of tensors in the TT format. We say that the tensor $\mathcal{X}$ has *low rank* if the TT ranks are much smaller than $n_1, \ldots, n_d$. A *low-rank approximation* of $\mathcal{X}$ in the TT format can be constructed by modifying the TT-SVD. In particular, if one truncates each SVD (Step 4 and Step 9) in Algorithm 5.1 as in (5.13), then the algorithm yields an approximation $\tilde{\mathcal{X}}$ of $\mathcal{X}$ which has lower TT ranks. Assuming that

[1] This is the "best" choice for $(r_0, r_1, \ldots, r_d)$ in the sense that if (5.14) holds for some $r_1, \cdots, r_{d-1}$, then necessarily $r_\mu \geq \text{rank}(X^{<\mu>})$, for $\mu = 1, \ldots, d-1$.

Algorithm 5.1 TT-SVD

Input: Tensor $\mathcal{X}$
Output: TT cores $\mathbf{U}_\mu$ for $\mu = 1, \dots, d$

$n \leftarrow n_1 n_2 \dots n_d$
$n_R \leftarrow n/n_1$
$X_1 \leftarrow \texttt{reshape}(\mathcal{X}, [n_1, n_R])$
Compute the SVD $X_1 = U\tilde{V}$ and set $r_1 \leftarrow \operatorname{rank}(X_1)$
$\mathbf{U}_1 \leftarrow U$
for $\mu = 2, \dots, d-1$ **do**
  $n_R \leftarrow n_R/n_\mu$
  $\tilde{V}_{\mu-1} \leftarrow \texttt{reshape}(\tilde{V}, [r_{\mu-1} n_\mu, n_R])$
  Compute the SVD $\tilde{V}_{\mu-1} = U\tilde{V}$ and set $r_\mu \leftarrow \operatorname{rank}(\tilde{V}_{\mu-1})$
  $\mathbf{U}_\mu \leftarrow \texttt{reshape}(U, [r_{\mu-1}, n_\mu, r_\mu])$
end for
$\mathbf{U}_d \leftarrow \tilde{V}$

$\operatorname{rank}_{\mathrm{TT}}(\tilde{\mathcal{X}}) = (1, \tilde{r_1}, \dots, \tilde{r}_{d-1}, 1)$, the approximation error is (see [101, Theorem 2.2])

$$\|\mathcal{X} - \tilde{\mathcal{X}}\|_F \leq \sqrt{\epsilon_1^2 + \dots + \epsilon_{d-1}^2},$$

where the ϵ-terms are given by the rank $\tilde{r}_\mu$-truncations of the unfoldings of $\mathcal{X}$, i.e.

$$\epsilon_\mu^2 := \|X^{<\mu>} - \mathcal{T}_{\tilde{r}_\mu}(X^{<\mu>})\|_F^2.$$

While the Theorem 5.3 states that the truncated SVD yields the best rank-r approximation in the case of matrices, the following result states that the TT-SVD yields a quasi-optimal approximation, see [101, Corollary 2.4].

Corollary 5.4. *Let $\mathcal{X}_{best}$ denotes the best approximation of $\mathcal{X}$ with TT ranks bounded by $(1, \tilde{r}_1, \dots, \tilde{r}_{d-1}, 1)$, i.e.*

$$\mathcal{X}_{best} := \operatorname{argmin}_{\mathcal{Y} \in \mathbb{R}^{n_1 \times \dots \times n_d}} \Big\{ \|\mathcal{X} - \mathcal{Y}\|_F : \mathit{rank}_{TT}(\mathcal{Y}) \leq (1, \tilde{r}_1, \dots, \tilde{r}_{d-1}, 1) \Big\}.$$

Then, the tensor approximation $\tilde{\mathcal{X}}$ satisfies

$$\|\mathcal{X} - \tilde{\mathcal{X}}\|_F \leq \sqrt{d-1} \|\mathcal{X} - \mathcal{X}_{best}\|_F.$$

As in the case of matrices, if the singular values of the unfoldings of $\mathcal{X}$ exhibit a fast decay, $\tilde{\mathcal{X}}$ is a good approximation and if its TT ranks are small, we say that the tensor $\mathcal{X}$ has a *low-rank structure*. Moreover, if the TT ranks are small, storing the TT cores instead of the full tensor $\mathcal{X}$ yields a significant memory reduction: from $\mathcal{O}(n^d)$ to $\mathcal{O}(dnr^2)$, where $r = \max\{r_0, \ldots, r_d\}$ and $n = \max\{n_1, \ldots, n_d\}$.

Some operations can be effected quite cheaply in the TT format for tensors of low TT ranks. Let us first consider the inner product of two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ defined as

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \langle \mathrm{vec}(\mathcal{X}), \mathrm{vec}(\mathcal{Y}) \rangle = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \mathcal{X}(i_1, \ldots, i_d)\mathcal{Y}(i_1, \ldots, i_d), \tag{5.16}$$

where $\mathrm{vec}(\cdot)$ stacks the entries of a tensor into a long vector. The corresponding tensor network diagram when $\mathcal{X}$ and $\mathcal{Y}$ are both in the TT decomposition is shown in Figure 5.2. It can be seen that the summations in (5.16) become contractions between the TT cores of $\mathcal{X}$ and $\mathcal{Y}$. By carrying out these contractions of cores from the left to right, the cost of evaluating the inner product reduces from $\mathcal{O}(n^d)$ to $\mathcal{O}(dnr^3)$, where r denotes the maximum of all involved TT ranks. This procedure is described in [101, Algorithm 4].

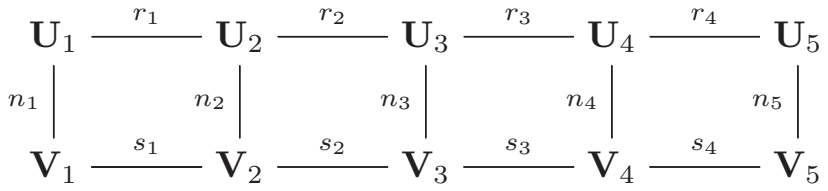

Figure 5.2 – Inner product of two tensors of order $d = 5$ in the TT decomposition.

The *mode-μ matrix multiplication* between a tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a matrix $M \in \mathbb{R}^{m \times n_\mu}$ results in a tensor $\mathcal{Z} \in \mathbb{R}^{n_1 \times \cdots n_{\mu-1} \times m \times n_{\mu+1} \cdots \times n_d}$ defined by

$$\mathcal{Z}(i_1, \cdots, i_{\mu-1}, j, i_{\mu+1} \cdots, i_d) = \sum_{i_\mu=1}^{n_\mu} \mathcal{X}(i_1, \cdots, i_d)M(j, i_\mu), \quad j = 1, \ldots, m.$$

We will denote this operation by $\mathcal{Z} = \mathcal{X} \times_\mu M$. If $\mathcal{X}$ is in the TT decomposition (5.14) then it is straightforward to obtain a TT decomposition for $\mathcal{Z}$, by performing a mode-2 matrix multiplication of $\mathbf{U}_\mu$ with M. Indeed, the core tensors $\mathbf{U}_1, \ldots \mathbf{U}_{\mu-1}, \mathbf{U}_{\mu+1}, \ldots, \mathbf{U}_d$ remain unchanged and the μ-th core tensor of $\mathcal{Z}$ is obtained by computing $M\mathbf{U}_\mu^{(2)}$, where

$\mathbf{U}_\mu^{(2)} \in \mathbb{R}^{n_\mu \times r_{\mu-1} r_\mu}$ is the 2-*mode* matricization of $\mathbf{U}_\mu$, see e.g. [59]. This has a cost of $\mathcal{O}(nmr^2)$ while computing $\mathcal{Z} = \mathcal{X} \times_\mu M$ in full format requires $\mathcal{O}(mn^d)$ operations. Again, we obtain a significant complexity reduction by exploiting the TT format.

5.3.4 Completion algorithm

The goal of completion algorithms is to reconstruct a given data set from a small fraction of its entries. As this is clearly an ill-posed task, one needs to additionally impose some regularization, such as smoothness conditions. In this work, we impose low TT ranks on the tensor $\mathcal{P}$ containing the prices and reconstruct $\mathcal{P}$ using the completion algorithm proposed in [112].

In the following, we briefly summarize the approach from [112]. Let $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ denote the original data tensor for which only the entries in a (small) *training set* $\Omega \subset \{1, \ldots, n_1\} \times \cdots \times \{1, \ldots, n_d\}$ are known. When aiming at fitting a tensor of fixed (low) TT ranks $\mathbf{r} = (r_0, \ldots, r_d)$ to this data, completion takes the form of the constrained optimization problem

$$\begin{aligned} \min_{\mathcal{X}} \quad & \frac{1}{2} ||P_\Omega \mathcal{X} - P_\Omega \mathcal{A}||^2 \\ \text{subject to} \quad & \mathcal{X} \in \mathcal{M}_\mathbf{r} := \{\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d} \mid \operatorname{rank}_{\mathrm{TT}} = \mathbf{r}\}, \end{aligned} \tag{5.17}$$

where $P_\Omega \mathcal{X}$ denotes the orthogonal projection onto Ω and $\|\cdot\|$ is the norm induced by the inner product (5.16). It is known that $\mathcal{M}_\mathbf{r}$ is a smooth embedded submanifold of $\mathbb{R}^{n_1 \times \cdots \times n_d}$ (see e.g. [72]), which enables one to apply Riemannian optimization techniques to (5.17). Specifically, in [112] it is proposed to employ a Riemannian conjugate gradient (CG) method (see Algorithm 1 in [112]). This method produces iterates that stay on the manifold and, in turn, can be stored and manipulated efficiently in the TT format. In the following we provide the basic outline of the Riemannian CG.

Riemannian CG

We start by introducing the Euclidean (nonlinear) CG algorithm. Later on, we explain how to design a Riemannian version of it, obtaining the basic outline of the algorithm developed in [112] that we use to solve (5.17).

The Euclidean CG algorithm is a line-search method that aims at numerically solving an optimization problem of the form

$$\min_{x\in\mathbb{R}^n} f(x),$$

for an objective function $f : \mathbb{R}^n \to \mathbb{R}$, which is assumed to be differentiable. The idea is to create a sequence of iterates $\{x_0, x_1, \ldots\}$ that converges to a local minimizer of f. In the case of the Euclidean CG algorithm, see e.g. [61], these iterates are defined as (for a random starting vector x_0)

$$x_{k+1} := x_k + \alpha_k \eta_k, \quad k = 0, \ldots, \tag{5.18}$$

where α_k is a stepsize, and η_k is the *search direction* given by

$$\eta_k := \begin{cases} -\nabla f(x_k), & \text{for } k = 0, \\ -\nabla f(x_k) + \beta_k \eta_{k-1}, & \text{for } k = 1, \ldots, \end{cases} \tag{5.19}$$

for some parameter β_k. The search directions $\{\eta_k\}_{k\in\mathbb{N}_0}$ are referred to as the *conjugate directions*. Ideally, the stepsize α_k is chosen such that it minimizes $f(x_{k+1})$, i.e.

$$\alpha_k := \text{argmin}_\alpha f(x_k + \alpha\eta_k).$$

The parameter β_k can be chosen in different ways, yielding different versions of the Euclidean CG algorithm. One way to define β_k is according to the *Fletcher-Reeves* rule, see [45], given by

$$\beta_k = \frac{\nabla f(x_k)^\top \nabla f(x_k)}{\nabla f(x_{k-1})^\top \nabla f(x_{k-1})}. \tag{5.20}$$

In order to solve the problem (5.17), we need to adapt the CG algorithm to the Riemannian setting. In particular, the problem is now of the form

$$\min_{x\in\mathcal{M}} f(x),$$

where we consider an arbitrary Riemannian submanifold $\mathcal{M}$, and $f : \mathcal{M} \to \mathbb{R}$. In order to derive a Riemannian version of the CG algorithm, the following points are considered, see e.g. [109].

- The Euclidean gradient is replaced by the *Riemannian gradient*, which we denote

Algorithm 5.2 Riemannian CG for tensor completion

Input: Initial guess $\mathcal{X}_0 \in \mathcal{M}_{\mathbf{r}}$.
$\xi_0 \leftarrow \text{grad} f(\mathcal{X}_0)$ % compute Riemannian gradient
$\eta_0 \leftarrow -\xi_0$ % first step is steepest descent
$\alpha_0 \leftarrow \text{argmin}_\alpha f(\mathcal{X}_0 + \alpha\eta_0)$ % step size by linearized line search
$\mathcal{X}_1 \leftarrow R(\mathcal{X}_0, \alpha_0\eta_0)$ % obtain next iterate by retraction
for $k = 1, 2, \ldots$ **do**
  $\xi_k \leftarrow \text{grad} f(\mathcal{X}_k)$ % compute Riemannian gradient
  $\eta_k \leftarrow -\xi_k + \beta_k T_{\mathcal{X}_{k-1} \to \mathcal{X}_k} \eta_{k-1}$ % conjugate direction by updating rule
  $\alpha_k \leftarrow \text{argmin}_\alpha f(\mathcal{X}_k + \alpha\eta_k)$ % step size by linearized search
  $\mathcal{X}_{k+1} \leftarrow R(\mathcal{X}_k, \alpha_k\eta_k)$ % obtain next iterate by retraction
end for

by $\text{grad} f(x)$. At each iteration k, the negative Riemannian gradient points into the direction of the steepest descent within the *tangent space* of $\mathcal{M}$ at the point x_k.

- At each iteration, we need to make sure that the new iterate x_k stays on the manifold $\mathcal{M}$. In general, after performing the update step (5.18), the new iterate x_k does not lie on $\mathcal{M}$. In the Riemannian CG, the new iterate is mapped back to the manifold by a *retraction* operator, which we denote by R. In [113, Figure 2.3], the concept of retraction is well illustrated.

- For $k \geq 1$, the new conjugate direction η_k in (5.19) is a linear combination of the current gradient and of the previous direction η_{k-1}. Since the vectors belong to different tangent spaces, the operation is not defined a priori in the Riemannian case. Therefore, in order to perform this operation we first need to transport η_{k-1} to the tangent space at the current point x_k. This is done by a *vector transport*, which we denote by $T_{x_{k-1} \to x_k}$. This concept is illustrated in [113, Figure 2.4].

- Finally, the parameter β_k can be computed as in (5.20) where the Euclidean gradient is replaced by the Riemannian gradient and the inner product is substituted by the *Riemannian metric*.

Considering the above points for (5.17) yields the Riemannian CG for the tensor completion problem summarized in the Algorithm 5.2, which corresponds to [112, Algorithm 1]. In [112] it is explained how to define the Riemannian gradient, the retraction operator, and the vector transport operator for the manifold $\mathcal{M}_{\mathbf{r}}$ of tensors of fixed TT ranks. Also, it is shown how to perform every step of the Riemannian CG algorithm 5.2 in the TT

format. Moreover, the complexity of the algorithm is discussed and it is shown that one iteration requires $\mathcal{O}(dnr^3 + d|\Omega|r^2)$ operations, where $|\Omega|$ denotes the cardinality of Ω.

Regarding the stopping criterion of the Riemannian CG, we design it to attain a level of accuracy warranted by the data and the chosen TT ranks. Following [112], we choose a *test set* Ω_C of, say, 100 additional parameter samples not in the training set Ω. At each iteration k, we measure the errors on the training and the test set:

$$\epsilon_\Omega(\mathcal{X}_k) := \frac{\|P_\Omega \mathcal{A} - P_\Omega \mathcal{X}_k\|}{\|P_\Omega \mathcal{A}\|}, \quad \epsilon_{\Omega_C}(\mathcal{X}_k) := \frac{\|P_{\Omega_C} \mathcal{A} - P_{\Omega_C} \mathcal{X}_k\|}{\|P_{\Omega_C} \mathcal{A}\|}.$$

The algorithm is stopped once these errors stagnate, that is,

$$\frac{|\epsilon_\Omega(\mathcal{X}_k) - \epsilon_\Omega(\mathcal{X}_{k+1})|}{|\epsilon_\Omega(\mathcal{X}_k)|} < \delta \quad \text{and} \quad \frac{|\epsilon_{\Omega_C}(\mathcal{X}_k) - \epsilon_{\Omega_C}(\mathcal{X}_{k+1})|}{|\epsilon_{\Omega_C}(\mathcal{X}_k)|} < \delta, \tag{5.21}$$

holds for some small $\delta > 0$.

Adaptive rank and adaptive sampling strategy

To set up the optimization problem (5.17), two issues remain to be discussed: The choice of the TT ranks $\mathbf{r}$ and a suitable training set Ω. For our application, these are not known a priori and thus need to be chosen adaptively.

Concerning the choice of TT ranks, we follow the adaptive strategy proposed in [112]. We start by solving (5.17) for the smallest sensible choice of TT ranks, $\mathbf{r} = (1, \dots, 1)$. Most likely, this choice will not suffice to obtain satisfactory accuracy and the error on the test set will be relatively large. To decrease it, the obtained solution is used as starting value for Riemannian CG applied again to (5.17), but this time with the increased TT ranks $\mathbf{r} = (1, 2, 1, \dots, 1)$ as discussed in [112]. See also [118] for a greedy rank update procedure in the context of matrix completion. The described procedure is repeated by increasing cyclically every TT rank r_μ. The overall algorithm stops as soon as increasing any of the TT ranks does not improve the test set error anymore or the maximal possible rank $r_{\max}$ is reached; see Algorithm 5.3.

For the adaptive choice of the sampling set Ω, which has not been addressed in [112], we present two different strategies. The core idea is to gradually increase the size of Ω in order to improve the approximation of the tensor. Both strategies are also combined with

Algorithm 5.3 Adaptive rank strategy

Input: Data on sampling/test sets Ω, Ω_C, max rank $r_{\max}$, acceptance parameter $\rho \geq 0$
Output: Completed tensor $\mathcal{X}$ with adaptively chosen TT ranks $\mathbf{r}$, $r_\mu \leq r_{\max}$.

1: $\mathcal{X}$ random tensor having TT ranks $\mathbf{r} = (1, \ldots, 1)$
2: $\mathcal{X} \leftarrow$ result of Riemannian CG using starting guess $\mathcal{X}$
3: $\mathsf{locked} = 0$
4: $\mu = 1$
5: **while** $\mathsf{locked} < d - 1$ & $\max_\nu r_\nu < r_{\max}$ **do**
6: $\quad \mathcal{X}_{\text{new}} \leftarrow$ increase μth rank of $\mathcal{X}$ to $(r_0, \ldots, r_{\mu-1}, r_\mu + 1, r_{\mu+1}, \ldots, r_d)$
7: $\quad \mathcal{X}_{\text{new}} \leftarrow$ result of Riemannian CG using starting guess $\mathcal{X}_{\text{new}}$
8: $\quad$ **if** $(\epsilon_{\Omega_C}(\mathcal{X}_{\text{new}}) - \epsilon_{\Omega_C}(\mathcal{X})) > -\rho$ **then**
9: $\quad\quad \mathsf{locked} \leftarrow \mathsf{locked} + 1$ $\quad$ % revert step
10: $\quad$ **else**
11: $\quad\quad \mathsf{locked} \leftarrow 0$, $\mathcal{X} \leftarrow \mathcal{X}_{\text{new}}$ $\quad$ % accept step
12: $\quad$ **end if**
13: $\quad \mu \leftarrow 1 + (\mu \mod d - 1)$
14: **end while**

Algorithm 5.3 and they differ only in the measurement of the error.

The steps of the first adaptive sampling strategy are as follows.

1. Start with a sample set Ω of small size and a test set Ω_C of a certain prescribed size $|\Omega_C|$. Run Algorithm 5.3.

2. Measure the relative error on the test set Ω_C and stop if the stopping criterion is satisfied. If not satisfied, add the test set Ω_C to the sample set Ω and create a new test set of size $|\Omega_C|$. In our applications, this corresponds to computing new option prices on the Chebyshev grid using the reference method.

3. Run again Algorithm 5.3 from line 2 to the end, by using a rank $\mathbf{r} = (1, \ldots, 1)$ approximation of the result from the previous step as initial guess for the CG algorithm.

4. Repeat 1-3 until a maximal sampling percentage is reached or an a priori chosen stopping criterion is satisfied.

The pseudo-code in Algorithm 5.4 summarizes this first strategy.

The second adaptive sampling strategy that we propose is designed in a similar way. The

Algorithm 5.4 Adaptive sampling strategy 1

Input: Initial sampled data $P_\Omega \mathcal{A}$, maximal rank $r_{\max}$ for rank adaptivity, maximal allowed size percentage p (of Ω)

Output: Completed tensor $\mathcal{X}$ of TT ranks $\mathbf{r}$, $r_\mu \leq r_{\max}$

1: Create test set ${\Omega_C}^{\text{new}}$ such that $\Omega \cap {\Omega_C}^{\text{new}} = \emptyset$
2: Run Algorithm 5.3 with $\Omega, {\Omega_C}^{\text{new}}$ and get completed tensor $\mathcal{X}_c$.
3: $\text{err}_{\text{new}} \leftarrow \epsilon_{{\Omega_C}^{\text{new}}}(\mathcal{X}_c)$
4:
5: **while** $|\Omega|/\text{size}(\mathcal{A}) < p$ **do**
6: $\quad \text{err}_{\text{old}} \leftarrow \text{err}_{\text{new}}$
7: $\quad \tilde{\mathcal{X}} \leftarrow$ rank $(1, \ldots, 1)$ approximation of $\mathcal{X}_c$
8: $\quad {\Omega_C}^{\text{old}} \leftarrow {\Omega_C}^{\text{new}}$
9: $\quad$ Create new test set ${\Omega_C}^{\text{new}}$ such that ${\Omega_C}^{\text{new}} \cap {\Omega_C}^{\text{old}} = \emptyset$
10: $\quad \Omega \leftarrow \Omega \cup {\Omega_C}^{\text{old}}$
11: $\quad$ Run Algorithm 5.3 (from line 2 to end) with $\Omega, {\Omega_C}^{\text{new}}$ and $\tilde{\mathcal{X}}$ as starting guess. Get completed tensor $\mathcal{X}_c$ out of it.
12: $\quad \text{err}_{\text{new}} \leftarrow \epsilon_{{\Omega_C}^{\text{new}}}(\mathcal{X}_c)$
13: $\quad$ **if** stopping criterion satisfied **then**
14: $\quad\quad$ Break
15: $\quad$ **end if**
16: **end while**
17:
18: $\mathcal{X} \leftarrow \mathcal{X}_c$

only difference is that the error is measured on an a priori defined fixed set Γ and not on Ω_C, which changes at each step. Therefore, this strategy follows the same steps as the first one, with the only difference that in Step 2 we measure the error on the set Γ, which has been previously defined. The algorithm summarizing this second strategy can be obtained by replacing line 3 and line 12 in Algorithm 5.4 with

$$\text{err}_{\text{new}} \leftarrow \epsilon_\Gamma(\mathcal{X}_c).$$

The stopping criterion of line 13 can be also defined in different ways. We choose to stop the algorithm if one of the following criteria is satisfied:

1. if $\text{err}_{\text{new}} < \text{tol}$, where tol is a prescribed tolerance;
2. if $|\text{err}_{\text{new}} - \text{err}_{\text{old}}| < \text{tol}'$, where tol' is a prescribed tolerance;
3. if $\exists \mu$ such that $r_\mu(\mathcal{X}_c) == r_{\max}$.

The first criterion allows us to stop as soon as the error goes below a certain level, the second stops the algorithm whenever the error stagnates, and the last one when the TT ranks have reached the maximal allowed rank at least in one mode μ.

We test the new adaptive sampling strategies on a problem with known solution in the next section.

Numerical test for adaptive sampling strategies

We consider the problem of Chapter 5.4.2 in [112] and we apply our adaptive sampling strategies to it in order to compare them and to investigate their advantages and disadvantages. We expect a similar performance of both strategies in terms of accuracy and compression. In this numerical example, as well as in the rest of the chapter, we choose $\|\cdot\|$ to be the 2-norm and $\delta = 10^{-4}$ in (5.21). The problem consists of discretizing the function

$$f : [0,1]^4 \to \mathbb{R}, \quad f(\mathbf{x}) = \exp(-\|\mathbf{x}\|)$$

using $n = 20$ equally spaced discretization points on $[0,1]$ in each mode. We aim at reconstructing the tensor containing the function values in the grid. In Algorithm 5.4 we set the maximum rank to $\mathbf{r}_{\max} = (1,7,7,7,1)$ and we start with an initial sampling set Ω satisfying $|\Omega|/n^4 = 0.01$. Moreover, we set the acceptance parameter ρ of Algorithm 5.3 to $\rho = 10^{-4}$. In order to analyze the behavior of the error, we do not impose any stopping criterion, but we let our adaptive sampling strategies run until $|\Omega|/\text{size}(\mathcal{A}) > 0.25$. The size of each Ω_C is set to 2000 and $|\Gamma| = 3000$ for the second strategy. Figures 5.3 and 5.4 show the results for the two different strategies.

First, we observe that both strategies eventually reach the same accuracy and the same final TT ranks, which makes both of them valid. We observe an oscillatory behavior in Figure 5.3. This non-smooth decay can be expected since in each step the error is measured on a different test set Ω_C. We observe that the amplitude of the oscillations becomes smaller as $|\Omega|$ increases. This indicates an error stagnation over the whole tensor which cannot be improved by enlarging Ω_C further. On the other hand, the error in the second strategy behaves almost monotonically and stagnates much earlier than in the previous case. This is due to the fact that we measure it on the fixed set Γ. In practice, the earlier error stagnation of the second strategy is preferable as it triggers the stopping criterion 2. However, the second strategy has the disadvantage of the initial additional

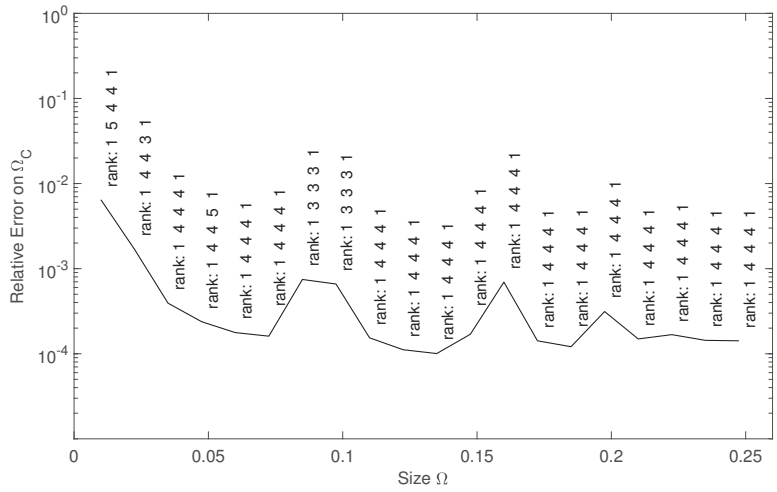

Figure 5.3 – Relative error on varying test sets Ω_C for different sampling set sizes in adaptive sampling strategy 1.

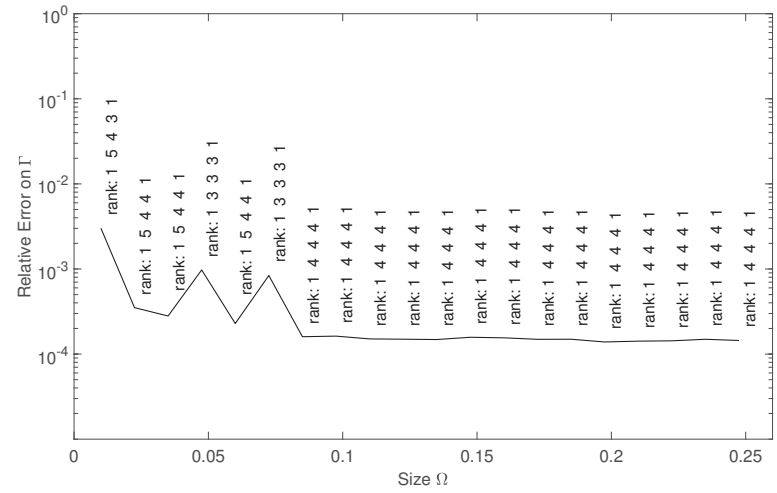

Figure 5.4 – Relative error on set Γ for different sampling set sizes in adaptive sampling strategy 2.

cost of evaluating the tensor in the set Γ. In our numerical experiments in Section 5.4 we choose the first strategy, which turned out to be more favorable since the stopping criterion 1 was triggered.

5.3.5 Combined methodology

We are now in the position to combine the concepts and the algorithms in order to develop an efficient procedure for high-dimensional tensorized Chebyshev interpolation.

We would like to price options that depend on a vector $\mathbf{p} = (p_1, \cdots, p_d)$ of d varying parameters. It is reasonable to assume that every combination of parameters $\mathbf{p}$ belongs to a compact hyper-rectangular $[\underline{p}_1, \overline{p}_1] \times [\underline{p}_2, \overline{p}_2] \times \cdots \times [\underline{p}_d, \overline{p}_d]$. For example, if time-to-maturity T belongs to the set of varying parameters, we can assume that $T \in [0.05, 2]$; similarly for the other payoff or model parameters. The combined methodology consists of two phases: offline phase and online phase, as already introduced in [46].

Offline phase - Computation of $\mathcal{P}$

The offline phase starts by performing following operations:

1. Fix an interpolation order $\overline{\mathbf{n}} = (n_1, \ldots, n_d)$ and compute the entries of the tensor $\mathcal{P}$ (as defined in (5.6)) from an a priori chosen subset Ω of Chebyshev nodes, using the reference pricing technique.

2. Apply tensor completion with adaptive sampling strategy (Algorithm 5.4) in order to get a low-rank approximation of the tensor $\mathcal{P}$ in the TT format.

For simplicity, we denote the obtained low-rank approximation of $\mathcal{P}$ again by $\mathcal{P}$[2]. In the last step of the offline phase we construct the interpolation coefficients, defined in (5.5). We denote the tensor of coefficients by $\mathcal{C} \in \mathbb{R}^{(n_1+1)\times(n_2+1)\times\cdots\times(n_d+1)}$. Its entries are therefore given by (adjusting the ordering according to the Sections 5.3.1 and 5.3.3)

$$\mathcal{C}(i_1, i_2, \cdots, i_d) = c_{i_1-1, i_2-1, \cdots, i_d-1}, \tag{5.22}$$

for $i_j = 1, \cdots, n_j + 1$ and $j = 1, \cdots, d$. The tensor $\mathcal{C}$ can be efficiently computed in the TT format, as explained in the following subsection.

[2]Note that in Section 5.3.1 we let the indices of the entries of $\mathcal{P}$ start from 0. From now on, we let them start from 1 in order to be consistent with the new notation introduced in Section 5.3.2

Offline phase - Efficient computation of $\mathcal{C}$

In order to explain the algorithm we first consider the simple case $d = 1$. In this case $\mathcal{P}$ and $\mathcal{C}$ are in $\mathbb{R}^{(n_1+1)\times 1}$, where n_1 is the chosen interpolation order. The entries of $\mathcal{C}$ are given by

$$\mathcal{C}(j+1) = \frac{2^{\mathbb{1}_{n_1>j>0}}}{n_1} \sum_{k=0}^{n_1}{}'' \mathcal{P}(k+1) \cos\left(j\pi \frac{k}{n_1}\right), \quad j = 0, \cdots, n_1.$$

By defining a matrix $F_{n_1} \in \mathbb{R}^{(n_1+1)\times(n_1+1)}$ as

$$F_{n_1} := \frac{2}{n_1} \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \cdots & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & \cos(\frac{\pi}{n_1}) & \cdots & \cos(\frac{\pi(n_1-1)}{n_1}) & \frac{1}{2}\cos(\pi) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{2} & \cos(\frac{\pi(n_1-1)}{n_1}) & \cdots & \cos(\frac{\pi(n_1-1)^2}{n_1}) & \frac{1}{2}\cos(\pi(n_1-1)) \\ \frac{1}{4} & \frac{1}{2}\cos(\pi) & \cdots & \frac{1}{2}\cos(\pi(n_1-1)) & \frac{1}{4}\cos(\pi n_1) \end{pmatrix},$$

the whole vector $\mathcal{C}$ of coefficients can be computed via the matrix-vector multiplication

$$\mathcal{C} = F_{n_1}\mathcal{P}. \tag{5.23}$$

For a general dimension $d > 1$, the same reasoning can be applied and the tensor $\mathcal{C}$ of interpolation coefficients can be computed by sub-sequentially multiplying $\mathcal{P}$ with F_{n_i} $(i = 1, \cdots, d)$ via the mode-μ multiplication, defined in Section 5.3.3. The final procedure for an efficient computation of $\mathcal{C}$ is given in Algorithm 5.5.

Algorithm 5.5 Efficient computation of $\mathcal{C}$

Input: Tensor $\mathcal{P}$ in the TT format containing option prices in the Chebyshev grid
Output: Tensor $\mathcal{C}$ as defined in (5.22), in the TT format

1: Compute F_{n_1} as in (5.23).
2: $\mathcal{C} \leftarrow \mathcal{P} \times_1 F_{n_1}$
3: **for** $m = 2, \ldots, d$ **do**
4: Compute F_{n_m}
5: $\mathcal{C} \leftarrow \mathcal{C} \times_m F_{n_m}$.
6: **end for**

Note that if $n_1 = \cdots = n_d =: n$, Algorithm 5.5 can be further simplified by computing the

matrix F_n only once. Following the cost of the mode-μ multiplication illustrated in Section 5.3.3, one can easily see that Algorithm 5.5 requires $\mathcal{O}(dn^2r^2)$ operations. Moreover, the cost of the operation (5.23) can be further reduced by applying a Fast-Fourier-Transform (FFT) based algorithm. As a consequence, also the cost of each mode multiplication in the Algorithm 5.5 can be further reduced, from $\mathcal{O}(r^2n^2)$ to $\mathcal{O}(r^2n\log(n))$. In summary, running the whole Algorithm 5.5 and computing $\mathcal{C}$ can be performed in $\mathcal{O}(dnr^2\log(n))$ operations. In the following remark we give a detailed explanation on how to employ a FFT based algorithm to perform (5.23).

Remark 5.5. *We recall that the discrete Fourier transform (DFT) of an arbitrary sequence of complex numbers $x_0, \dots, x_{n-1}$ is defined as the sequence $f_0, \cdots, f_{n-1}$ where*

$$f_j := \sum_{k=0}^{n-1} x_k e^{-i2\pi j\frac{k}{n}}, \quad j = 0, \dots, n-1.$$

A Fast-Fourier-Transform is an algorithm that computes the DFT of a sequence in $\mathcal{O}(n\log(n))$ operations, see e.g. [119]. We explain how to compute the coefficients $\mathcal{C}$ via a FFT based algorithm. For consistency with the definition of DFT, we consider the notation

$$c_j = \frac{2^{\mathbb{1}_{n>j>0}}}{n}\sum_{k=0}^{n}{}'' \,\mathsf{Price}^{q_k}\cos\Big(j\pi\frac{k}{n}\Big), \quad j = 0, \dots, n \tag{5.24}$$

for the interpolation coefficients.

- *We define the vector*

$$\tilde{p} := [\tilde{p}_0, \cdots, \tilde{p}_{2n-1}] := [\mathsf{Price}^{q_0}, \mathsf{Price}^{q_1}, \dots, \mathsf{Price}^{q_n}, \mathsf{Price}^{q_{n-1}}, \dots, \mathsf{Price}^{q_1}].$$

- *We compute the DFT of $\tilde{p}$; we take its real part, and we divide each number of the sequence by n, resulting in the sequence*

$$\tilde{c}_j = \frac{1}{n}\sum_{k=0}^{2n-1} \tilde{p}_k\cos\Big(j\pi\frac{k}{n}\Big), \quad j = 0, \dots, 2n-1. \tag{5.25}$$

- *By exploiting the relation $\tilde{p}_k = \tilde{p}_{2n-k}$, for all $k = 1, \cdots, 2n-1$, and the fact that*

$\cos\left(\frac{\pi jk}{n}\right) = \cos\left(\frac{\pi j(2n-k)}{n}\right)$, *we can rewrite* (5.25) *as*

$$\tilde{c}_j = \frac{2}{n}\sum_{k=0}^{n}{}'' \tilde{p}_k \cos\left(j\pi\frac{k}{n}\right), \quad j = 0, \dots, 2n-1.$$

- *Finally, the interpolation coefficients* (5.24) *are given by*

$$c_0 = \frac{\tilde{c}_0}{2}, \quad c_n = \frac{\tilde{c}_n}{2},$$
$$c_j = \tilde{c}_j, \quad \textit{for } j = 1, \dots, n-1.$$

The offline phase can be finally completed by performing the step

3. Construct the tensor $\mathcal{C}$ as explained in Algorithm 5.5.

Online phase

Once we have stored $\mathcal{C}$ in the TT format, we can use it to compute every option price via interpolation during the online phase. For any particular choice of parameters $\mathbf{p}$, we first perform the step

4. Evaluate the Chebyshev tensor basis (5.3) in $\mathbf{p}$.

This step returns a tensor $\mathcal{T}_{\mathbf{p}} \in \mathbb{R}^{(n_1+1)\times(n_2+1)\times\cdots\times(n_d+1)}$ of TT ranks $(1, \cdots, 1)$, that we store in the TT format. The interpolated price, defined in (5.4), can now be rewritten as the inner product

$$I_{\overline{\mathbf{n}}}(\mathsf{Price}^{(\cdot)})(\mathbf{p}) = \langle \mathcal{C}, \mathcal{T}_{\mathbf{p}} \rangle. \tag{5.26}$$

The final step of our combined methodology is then defined as

5. Compute the interpolated price (5.26) in the TT format as in (5.16).

If we consider a fixed interpolation order n in each dimension and if the TT ranks of $\mathcal{P}$ and $\mathcal{C}$ are approximately r, then the total cost for performing both Step 3 and Step 5 is given by $\mathcal{O}(dnr^2 + dnr^2\log(n))$. These two steps are represented via a tensor network

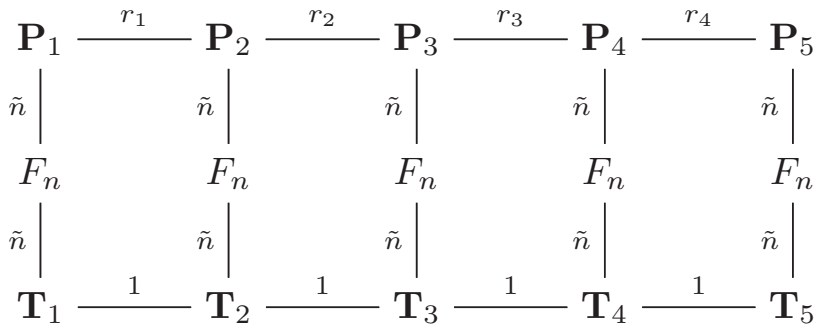

Figure 5.5 – Tensor network diagram representing the whole interpolation procedure as in (5.4) *and* (5.5)*, for $d = 5$ in the TT format. Note that $\tilde{n} := n + 1$.*

diagram in Figure 5.5 (for $d = 5$), where we denoted by $\mathbf{P}_i$ the core tensors of $\mathcal{P}$ and by $\mathbf{T}_i$ the ones of $\mathcal{T}_{\mathbf{p}}$.

Finally, we summarize our complete methodology in Algorithm 5.6.

Algorithm 5.6 Combined methodology for Chebyshev interpolation in parametric option pricing

Input: Interpolation order $\overline{\mathbf{n}}$, subset Ω of total Chebyshev nodes, set Π of parameters $\mathbf{p}$ for which we want to compute option prices
Output: Interpolated option prices for parameters $\mathbf{p} \in \Pi$

```
% Offline phase
Compute option prices using reference method in the subset Ω of Chebyshev points
Construct 𝒫 using tensor completion in the TT format (Algorithm 5.4)
Construct tensor 𝒞 as in Section 5.3.5

% Computation of option prices - Online phase
for p ∈ Π do
    Evaluate the Chebyshev tensor basis 𝒯_p
    Compute interpolated price (5.26)
end for
```

In the next section we see how this combined methodology performs on concrete examples.

5.4 Financial applications and numerical experiments

Putting the new approach to test, we implement the method described in Section 5.3 for two different types of applications. In the first one, we tackle computational intense option pricing methods in a parametric model. We treat option prices as functions in the parameter space which consists of model and option parameters. We then approximate the price function by Chebyshev interpolation in the parameter space. This approach has been successfully tested in cases where the parameter space is low-dimensional. In various applications, several varying parameters are of interest. If the interpolation is even efficient in the full parameter space, it can be interpreted as a new pricing methodology. Here, we combine Chebyshev interpolation and low-rank approximation to cope with higher dimensionality in the parameter space. Already for pricing single asset options, it is promising to tackle medium and high-dimensional parameters spaces in this approach. As a generic example, we choose to approximate American put option prices in the Heston model with the varying parameters K, ρ, σ, κ and θ. It turns out that the computational complexity reduces significantly in this case.

As second type of application we examine the interpolation of basket option prices in the d-variate Black-Scholes model as function of the initial asset prices. This is a prototypical example for the computation of generalized conditional moments of high-dimensional Markov processes.

All algorithms have been implemented in MATLAB and run on a standard laptop (Intel Core i7, 2 cores, 256kB/4MB L2/L3 cache). In order to deal with tensors, we used the toolboxes [101] by Oseledets and [7, 8], while for the completion algorithm we used the TT completion toolbox described in [84, 85, 112, 113]. Note that in this toolbox the most expensive steps have been implemented in C using the `Mex`-function capabilities of MATLAB.

5.4.1 Pricing American options in Heston's model

We consider pricing single asset American put options in the Heston model. In contrast to Chapter 4, in this example we use the asset price formulation of the model, and not the log-asset price formulation as presented in Section 2.2. The price dynamics of the

asset price under the risk neutral measure are given by

$$dS_t = rS_t dt + \sqrt{v_t} S_t dW_t^1,$$

where the square of the volatility v_t is modeled by the square root process

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t^2.$$

Here, the two Brownian motions (W_t^1) and (W_t^2) are correlated with correlation parameter ρ, mean-reversion rate $\kappa > 0$, long-term mean $\theta > 0$, volatility of the variance $\sigma > 0$ and, finally, fixed and deterministic continuously compounding interest rate r.

We recall that the price of an American put option at time $t < T$, maturing at T, with initial underlying price $s \geq 0$ and initial volatility $v \geq 0$ is given by (see Section 1)

$$\mathsf{Price} = \sup_{\tau \in \mathcal{S}_{t,T}} \mathbb{E}[e^{-r(\tau - t)} f(S_\tau) | S_t = s, v_t = v], \tag{5.27}$$

where $\mathcal{S}_{t,T}$ is the set of all stopping times in $[t, T]$, and f is the payoff function of the European put option, i.e. $f(x) = (K - x)^+$, for a strike price K.

As introduced in Chapter 1, the price (5.27) of the American option satisfies the partial differential complementarity problem (PDCP)

$$\begin{cases} \partial_t \mathsf{Price} + \mathcal{G}\mathsf{Price} - r\mathsf{Price} \geq 0 \\ \mathsf{Price} \geq f \\ (\mathsf{Price} - f)(\partial_t \mathsf{Price} + \mathcal{G}\mathsf{Price} - r\mathsf{Price}) = 0, \end{cases} \tag{5.28}$$

where the generator $\mathcal{G}$ in the asset formulation of the Heston model is given by

$$\mathcal{G}g(s, v) = \frac{1}{2} s^2 v \partial_{ss}^2 g + \rho\sigma s v \partial_{sv}^2 g + \frac{1}{2}\sigma^2 v \partial_{vv}^2 g + rs\partial_s g + \kappa(\theta - v)\partial_v g.$$

The problem (5.28) has been well studied in the literature and different pricing algorithms have been developed so far. In our example we consider, as reference method for our combined methodology, the pricing algorithm explained in [60]. More precisely, the authors propose different schemes for the time and the spatial discretization and we consider the Hundsdorfer Verwer - Ikonen Toivanen (HV-IT) scheme, explained at page 219 of [60].

Solving the discretized PDCP yields an approximate price for all values of S_0, v_0 and T in each grid point of the pre-specified domain. For many applications we would like to have the solution at hand for other parameters (as well). In calibration, for instance, we observe S_0 and r, and one could estimate v_0 from historical stock price data. Then the calibration problem reduces to fitting the parameters $(K, \rho, \sigma, \kappa, \theta)$ to the observed option price data. To do so one needs to solve an optimization problem where prices need to be computed for large sets of parameters $(K, \rho, \sigma, \kappa, \theta, T)$. Since the price for different maturities can be obtained by rescaling κ and σ, effectively we need the prices for combinations of the parameters K, ρ, σ, κ and θ. This motivates the following set up, where we fix the model and payoff parameters

$$S_0 = 2, \quad v_0 = 0.0175, \quad r = 0.1, \quad T = 0.25,$$

and we let vary the five parameters

$$(K, \rho, \sigma, \kappa, \theta) \in [2; 4] \times [-1; 1] \times [0.2; 0.5] \times [1; 2] \times [0.05; 0.2]$$

in their corresponding domain.

In order to compute the reference prices we consider 50 equidistant spatial grid points in both directions s and v with $s_{\min} = 0, s_{\max} = 5, v_{\min} = 0, v_{\max} = 1$, 40 time steps and the Crank-Nicholson time stepping scheme.

We start by performing the offline phase of Algorithm 5.6. We consider an interpolation order $n_1 = \cdots = n_5 =: n = 10$ in each direction and we construct the tensor $\mathcal{P}$ by tensor completion as explained in Section 5.3.4. We apply the first adaptive sampling strategy as in Algorithm 5.4. We choose the completion parameters as

$$\rho = 0, \quad tol = 10^{-3}, \quad tol' = 10^{-8}, \quad r_{\max} = 10, \quad |\Omega| = 805, \quad |\Omega_C| = 805, \quad p = 0.2.$$

For this particular example, we were also able to explicitly construct the full tensor (in more than 1 hour and 40 minutes!). In Table 5.1 we show the size of the final set Ω (first column), the relative error of the completed tensor on the last Ω_c^{new} (second column), the relative error between the obtained completed tensor and the full one (third column), the runtime of the completion, Algorithm 5.4, in seconds (fourth column), the TT ranks of $\mathcal{P}$ (fifth column), the storage needed to save $\mathcal{P}$ in the TT format, denoted by store(TT) and measured in bytes (sixth column) and finally, the storage

needed to save the full tensor, denoted by store(full) and again measured in bytes. MATLAB requires 8 bytes to store a floating-point number of type double, which gives us the formula store(full)$= 8 \cdot (n+1)^d$ for the storage of the full tensor and store(TT)$= 8 \cdot (n+1)(r_1 r_2 + \cdots + r_{d-2} r_{d-1}) + 8 \cdot (n+1)(r_1 + r_{d-1})$ for the storage of the tensor in the TT format, see Section 5.3.3

Table 5.1 shows that a sample set of 5% is sufficient for the algorithm to reach the prescribed accuracy. Furthermore, the relative error of the completed tensor $\mathcal{P}$ in the 2-norm over the last test sample parameter space Ω_c^{new} and the relative error over the full tensor, i.e. over all Chebyshev nodes, is only in the 6th digit. This is one order of magnitude smaller than the relative error on the full $\mathcal{P}$. This is a good indication that the approach can be extended to more complex cases, where the computation of the full tensor $\mathcal{P}$ is not feasible any more (see Section 5.4.2). The completion time was about 6 minutes. Finally, the rank properties together with its storage reduction of a factor of 115 confirm the low-rank structure of the problem.

final $\lvert\Omega\rvert$	rel err on last Ω_c^{new}	rel err on full $\mathcal{P}$	completion time (s)
8050 (5 %)	$2.56 \cdot 10^{-5}$	$2.75 \cdot 10^{-5}$	366.12

$\text{rank}_{\text{TT}}(\mathcal{P})$	store(TT) (bytes)	store(full) (bytes)
$(1, 5, 8, 6, 5, 1)$	11264	1288408

Table 5.1 – Completion results on $\mathcal{P}$ for the parametric American put option pricing problem in the Heston model.

For constructing the tensor $\mathcal{C}$ (last step of the offline phase) we applied Algorithm 5.5 and the computation time was 0.0037 seconds, which is negligible compared to the completion time. Hence, almost all the computation time in the offline phase is spent in the construction of the tensor $\mathcal{P}$.

Next, we compute American put option prices for the online phase in both ways using our methodology and the reference algorithm. We compute 243 prices with random model parameters uniformly drawn from the reference set $[2;4] \times [-1;1] \times [0.2;0.5] \times [1;2] \times [0.05;0.2]$. We measure the maximal absolute error over the computed options prices, i.e. we report the quantity

$$\max(|P_{\text{Int}} - P_{\text{Ref}}|),$$

where P_{Int} is a vector containing all interpolated prices for the different choices of model

parameters; analogously is P_{Ref} for the reference method. In Table 5.2 we also report the computation time for computing one single option price for both methods. One can notice that the online phase of the interpolation compared to the reference method accelerates the procedure by a factor of 75. The accuracy of the reference method is reported in part C of Figure 1 in [60] for one specific parameter set to be of the order 10^{-3} in the maximum norm. The interpolation error is one order smaller, making the new procedure at least as accurate as the reference method. Therefore, we can conclude that the methodology strongly outperforms the reference method in the online phase while keeping the same accuracy.

time reference method (s)	time interpolation (s)	max abs error
$3.65 \cdot 10^{-2}$	$4.89 \cdot 10^{-4}$	$1.95 \cdot 10^{-4}$

Table 5.2 – Results on American put option pricing via combined methodology and reference method.

5.4.2 Basket options in the multivariate Black-Scholes model

We now consider the d-variate Black-Scholes model with d assets $S^1, \cdots, S^d$. The risk neutral dynamics and other relevant properties of the model are specified in Chapter 2, Section 2.2. We apply the new methodology in order to price basket options with payoff function $f : \mathbb{R}^d \to \mathbb{R}$ defined as

$$f(\mathbf{s}) := \left(\sum_{n=1}^{d} w_n s_n - K \right)^+,$$

where K is the strike and $(w_1, \cdots, w_d)$ is a vector of weights satisfying $\sum_{n=1}^{d} w_n = 1$. The price at time $t = 0$ of the basket option with maturity T is, as introduced in Chapter 1, given by

$$\mathsf{Price} = e^{-rT} \mathbb{E}[f(\mathbf{S}_T)]. \tag{5.29}$$

From now on, we consider the parameters r, σ_i $(i = 1, \cdots, d)$ and the correlation matrix Σ to be fixed, and we let the vector $\mathbf{S}_0 \in \mathbb{R}^d$ of initial asset prices be the varying parameter. The reference pricing algorithm will be of Monte Carlo (MC) type combined with a variance reduction technique. In particular, we use the control variates method presented

in [50], where the control variate is given by

$$Y =: \left(\exp\Big(\sum_{i=1}^{d} \omega_i \log(S_T^i) \Big) - K \right)^+.$$

Since the only varying parameter is the vector of initial asset prices, it is very convenient to split the Monte Carlo simulation in two parts in order to make the completion more efficient. More precisely, in a pre-computation phase (Algorithm 5.7) we simulate a certain number of realizations (e.g. 10^4) of

$$\exp\Big((r - \frac{\sigma_i^2}{2})T + \sigma_i W_T^i \Big), \quad \text{for } i = 1, \cdots, d,$$

and in a second moment we multiply the vector $\mathbf{S}_0$ (for all required parameter combinations) with all the realizations and we compute the Monte Carlo price by applying the chosen variance reduction technique (Algorithm 5.8). In order to generate the correlated random variables W_T^i, we use the Cholesky factorization of the correlation matrix, which is then multiplied by a vector of independently generated standard normal distributed random variates. Note that $\circ$ in Algorithm 5.8 represents the Hadamard (component-wise) product between vectors.

Algorithm 5.7 Simulation of correlated geometric Brownian motions

Input: Model and payoff parameters σ, Σ, T, r; number of simulations NumberSim.
Output: Matrix $M \in \mathbb{R}^{\text{NumberSim} \times d}$ containing simulated random variables.

1: $L \leftarrow$ Cholesky factor of Σ
2: $M \leftarrow \text{zeros}(\text{NumberSim}, d)$
3: **for** iSim $= 1 :$ NumberSim **do**
4: $\quad \epsilon \leftarrow$ Generate a vector of d independent standard normal variates
5: $\quad x \leftarrow L\epsilon$
6: $\quad$ **for** iStock $= 1 : d$ **do**
7: $\quad\quad M(\text{iSim}, \text{iStock}) \leftarrow \exp((r - \frac{\sigma(\text{iStock})^2}{2})T + \sigma(\text{iStock})x(\text{iStock})\sqrt{T})$
8: $\quad$ **end for**
9: **end for**

Algorithm 5.7 is executed at the beginning of the whole procedure and Algorithm 5.8 whenever needed in later stages. The advantage of splitting the MC algorithm is twofold. Firstly, it supports a considerable gain in efficiency in the performance of the completion algorithm: When we adaptively increment the sampling set Ω (which consists of sampling Chebyshev nodes in $\mathbf{S}_0$) in Algorithm 5.4, we need to compute new prices in the Chebyshev

Algorithm 5.8 Computation of basket options using MC with control variate technique

Input: Matrix M from Algorithm 5.7, $\mathbf{S}_0$, strike K, vector of weights ω, r, T
Output: Basket option price (5.29)

payoff $\leftarrow$ zeros(NumberSim, 1)
control $\leftarrow$ zeros(NumberSim, 1)
for iSim $= 1 :$ NumberSim **do**
  $R \leftarrow$ iSim-th row of M
  $S \leftarrow S_0 \circ R^\top$
  payoff(iSim) $\leftarrow (\sum_{i=1}^{d} \omega_i S_i - K)^+$
  control(iSim) $\leftarrow (\exp(\sum_{i=1}^{d} \omega_i \log(S_i)) - K)^+$
end for
Compute mean μ_Y of Y as explained in [50]
sum $\leftarrow$ payoff $-$ (control $- \mu_Y$)
Compute mean μ of sum
Price $\leftarrow \exp(-rT)\mu$

grid, which can be done by using Algorithm 5.8 only. The second advantage regards the analysis of the methodology and the completion accuracy: Since we use the same set of simulations for every Chebyshev price, the MC simulation does not introduce any further error to the completion. Moreover, we will see in Section 5.4.2 that this splitting procedure allows for a qualitative analysis of the rank structure of $\mathcal{P}$.

Next, we perform numerical experiments for different settings of model parameters, first for uncorrelated then for correlated assets.

Basket options of uncorrelated assets

In this example we consider the special case of uncorrelated assets. We investigate the performance of the proposed method for two different interpolation orders $n_1 = \cdots = n_d =: n = 4$ and $n_1 = \cdots = n_d =: n = 6$. We apply the combined methodology (Algorithm 5.6) to portfolios consisting of $d \in \{5, 10, 15, 20, 25\}$ assets. The set of fixed parameters is given by

$$T = 0.25, \quad K = 1, \quad r = 0, \quad \sigma_i = 0.2 \ \forall i, \quad \Sigma = I_d, \quad \omega_i = \frac{1}{d} \ \forall i,$$

where I_d denotes the $d \times d$ identity matrix. We let $\mathbf{S}_0$ vary in the hyper-rectangular

$$[1; 1.5]^d,$$

so that we consider ITM (in-the-money) options and ATM (at-the-money) options as well.

For each value of d, we start by performing Algorithm 5.7 with NumberSim $= 10^3$ for $n = 4$ and with NumberSim $= 10^4$ for $n = 6$. In a second moment we construct the tensor $\mathcal{P}$ by applying the tensor completion with the adaptive sampling strategy of Algorithm 5.4 (first strategy). Table 5.3 shows the completion parameters for each value of d and each interpolation order. The results of the tensor completions are displayed in Table 5.4. As in the previous subsection, we report the final size of the set Ω, the relative error measured on the last set Ω_C^{new}, the completion time and the memory needed to store both the obtained tensor in the TT format and the full tensor. For the TT ranks of the completed tensor, we do not report the full tuple $(r_0, \cdots, r_d)$ but only the quantity $\max_{\mu \in \{0,\cdots,d\}} r_\mu$.

	d	ρ	tol	tol'	$r_{\max}$	initial $\lvert\Omega\rvert$	$\lvert\Omega_C\rvert$	p
$n = 4$	5	0	10^{-2}	10^{-8}	5	31	31	10^{-1}
	10	0	10^{-2}	10^{-8}	5	78	78	10^{-2}
	15	0	10^{-2}	10^{-8}	5	214	214	10^{-5}
	20	0	10^{-2}	10^{-8}	5	763	763	10^{-8}
	25	0	10^{-2}	10^{-8}	5	2086	2086	10^{-11}
$n = 6$	5	0	10^{-3}	10^{-8}	7	17	17	10^{-1}
	10	0	10^{-3}	10^{-8}	7	282	141	10^{-3}
	15	0	10^{-3}	10^{-8}	7	475	475	10^{-6}
	20	0	10^{-3}	10^{-8}	7	798	798	10^{-10}
	25	0	10^{-3}	10^{-8}	7	1341	1341	10^{-15}

Table 5.3 – Completion parameters for constructing $\mathcal{P}$. Case of uncorrelated assets.

It is interesting to analyze the size of the finally obtained set Ω in Algorithm 5.4 for different values of d and n (different sizes of $\mathcal{P}$). Figure 5.6 shows a plot of $|\Omega|$ (final) against d for the two chosen interpolation orders. The graphical representation clearly suggests that the number of sampled entries, i.e. $|\Omega|$, required for the chosen tolerance $tol = 10^{-2}$ for a fixed interpolation order $n = 4$ and $tol = 10^{-3}$ for a fixed $n = 6$ is roughly of $\mathcal{O}(d^2)$, whereas the size of the full tensor is n^d. On the practical side, this means that by the completion algorithm we can reduce the complexity of the first step of the offline phase from an exponential growth down to a quadratic growth in the dimensionality. The exponential growth typically is referred to as curse of dimensionality. The reduction in absolute numbers is already tremendous for $d = 5$ and $n = 4$, where we observe $|\Omega| = 124$

	d	final $\lvert\Omega\rvert$	rel err on last Ω_C^{new}	completion time (s)
$n = 4$	5	124	$3.42 \cdot 10^{-3}$	9.90
	10	546	$2.54 \cdot 10^{-6}$	67.44
	15	1712	$3.55 \cdot 10^{-8}$	171.14
	20	2289	$5.03 \cdot 10^{-8}$	193.90
	25	4172	$3.96 \cdot 10^{-9}$	226.38
$n = 6$	5	204	$2.40 \cdot 10^{-4}$	52.55
	10	987	$1.20 \cdot 10^{-6}$	198.27
	15	1900	$2.28 \cdot 10^{-7}$	429.39
	20	3192	$2.97 \cdot 10^{-7}$	732.49
	25	4023	$1.35 \cdot 10^{-7}$	999.25

	d	max r_μ reached	store(TT) (bytes)	store(full) (bytes)
$n = 4$	5	5	2080	$2.50 \cdot 10^4$
	10	4	3440	$7.81 \cdot 10^7$
	15	4	5840	$2.44 \cdot 10^{11}$
	20	4	5800	$7.63 \cdot 10^{14}$
	25	4	10920	$2.38 \cdot 10^{18}$
$n = 6$	5	4	2688	$1.34 \cdot 10^5$
	10	6	9912	$2.26 \cdot 10^9$
	15	6	13720	$3.80 \cdot 10^{13}$
	20	5	11536	$6.38 \cdot 10^{17}$
	25	4	12600	$1.07 \cdot 10^{22}$

Table 5.4 – Completion results on $\mathcal{P}$ for the basket option pricing problem in the Black-Scholes model. Case of uncorrelated assets.

and the full tensor size equals $(n+1)^d = 3125$. The compression is dramatic for $n = 6$ and $d = 25$, namely the numbers of required entries shrinks by a factor of more than 3×10^{17}.

As in the previous numerical example, the computation time to build the tensor $\mathcal{C}$ of interpolation coefficients is negligible in the offline phase. Indeed, for all choices of d and n it is less than 0.01 seconds, for instance 0.0045 seconds for $n = 4$ and $d = 5$, and 0.0095 seconds for $n = 6$ and $d = 25$.

We now perform the online phase of Algorithm 5.6 in order to see how efficient becomes pricing basket options in the new setting. We start by computing 100 basket option prices via Chebyshev interpolation (combined methodology), choosing random initial asset prices $\mathbf{S}_0$ in the reference hypercube $[1; 1.5]^d$. We then compare the obtained prices

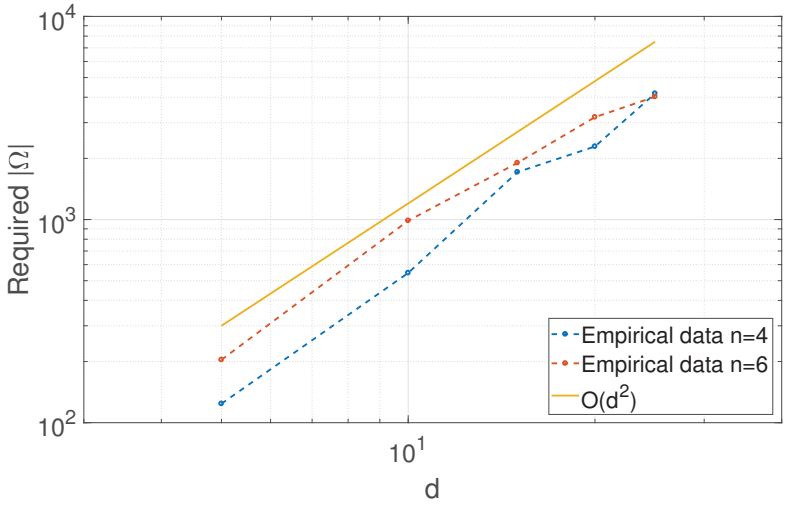

Figure 5.6 – Required size of Ω *for the completion to go below tol* $= 10^{-2}$ *for* $n = 4$ *and tol* $= 10^{-3}$ *for* $n = 6$*. Case of uncorrelated assets.*

with reference prices computed by applying the reference method (Monte Carlo with control variates) with 10^4 *new* simulations for $n = 4$ and 10^5 *new* simulations for $n = 6$. In particular, we measure again the maximal absolute error over all computed prices

$$\max(|P_{\text{Int}} - P_{\text{Ref}}|),$$

where P_{Int} is a vector containing all 100 interpolated prices for the different choices of $\mathbf{S}_0$; analogously is P_{Ref} for the reference method. The errors together with the computational times are shown in Table 5.5. Note that we report again the computational time to compute one single option price.

One can see that the online phase of the new procedure compared to the MC reference method accelerates the computation of a factor between 200 and 400 for $n = 4$ and of a factor between 2000 and 4000 for $n = 6$. Note that the difference in the acceleration between the two chosen interpolation orders is given by the different numbers of simulations chosen in the MC reference method (10^4 for $n = 4$ and 10^5 for $n = 6$). Therefore, for both interpolation orders and for all choices of d, the acceleration is dramatic. In order to judge the accuracy of our method we have computed the 95% confidence interval of the reference method, which results to be of a size between 10^{-4} and $5 \cdot 10^{-4}$ for all choices of $\mathbf{S}_0$ and d or n. This, together with the last column of Table 5.5, leads us to the conclusion that the new method is as accurate as the reference MC algorithm.

	d	time reference method (s)	time interpolation (s)	max abs error
$n = 4$	5	0.18	$0.45 \cdot 10^{-3}$	$3.75 \cdot 10^{-3}$
	10	0.19	$0.64 \cdot 10^{-3}$	$5.21 \cdot 10^{-4}$
	15	0.20	$0.73 \cdot 10^{-3}$	$4.38 \cdot 10^{-4}$
	20	0.20	$1.09 \cdot 10^{-3}$	$3.16 \cdot 10^{-4}$
	25	0.21	$0.97 \cdot 10^{-3}$	$2.08 \cdot 10^{-4}$
$n = 6$	5	1.84	$0.40 \cdot 10^{-3}$	$5.20 \cdot 10^{-4}$
	10	1.91	$0.61 \cdot 10^{-3}$	$1.42 \cdot 10^{-4}$
	15	1.99	$0.78 \cdot 10^{-3}$	$1.02 \cdot 10^{-4}$
	20	2.04	$0.93 \cdot 10^{-3}$	$1.01 \cdot 10^{-4}$
	25	2.10	$1.04 \cdot 10^{-3}$	$9.36 \cdot 10^{-5}$

Table 5.5 – Basket option prices computed via Chebyshev interpolation (combined methodology) versus MC reference method with 10^4 *simulations for* $n = 4$ *and* 10^5 *simulations for* $n = 6$*. Case of uncorrelated assets.*

Finally, in Figure 5.7 we show the gain in efficiency of the new method when computing basket option prices for $d = 25$ and both choices of interpolation orders. In particular, on the x-axis we consider a possible number of computed prices and on the y-axis we present

1. the computational time of the reference MC method,

2. the computational time of the new combined methodology (offline phase + online phase),

required to compute the corresponding amount of prices.

The plots in Figure 5.7 show that after an initial investment the computational time grows very slowly in the number of computed prices for the new method. This is due to the fact that the online phase in Algorithm 4 is very cheap, as shown in the numerical experiments. This proves that the method is useful whenever one can split the task in a pre-computational phase during idle times and a run-time phase where execution is required to be fast. Moreover, it will outperform the reference methods if a large number of prices needs to be computed. The first plot in Figure 5.7 indicates that for the case $n = 4$ it is convenient to use the reference MC method if we want to compute up to 1000 option prices. For the case $n = 6$ the break-even point is already reached with 500 prices.

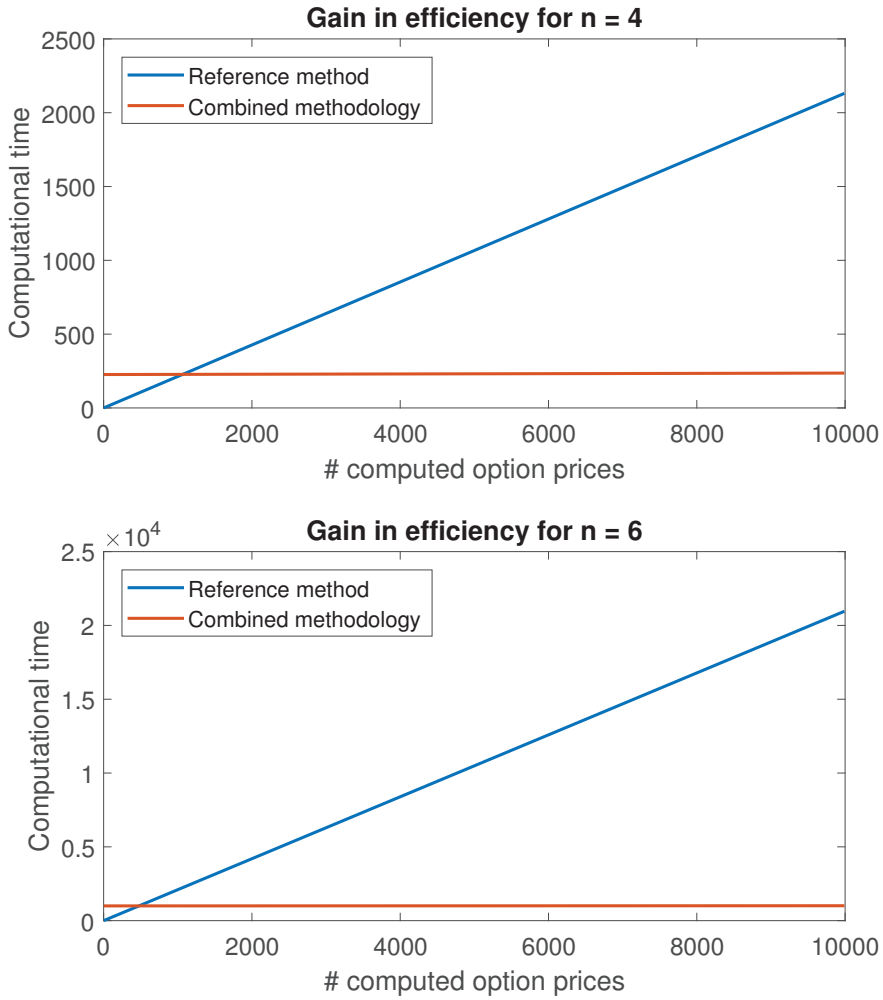

Figure 5.7 – Computational time for computing basket option prices. Comparison MC versus combined methodology for $n = 4$ and $n = 6$. Case of uncorrelated assets.

Basket options of correlated assets

In this second numerical experiment we repeat the test of the previous subsection but, this time, we consider correlated assets. In particular, we choose again the interpolation orders $n = 4$, $n = 6$ and the other parameters are given by

$$T = 0.25, \quad K = 1, \quad r = 0, \quad \sigma_i = 0.2 \ \forall i, \quad \Sigma = R_d, \quad \omega_i = \frac{1}{d} \ \forall i,$$

where R_d denotes a random correlation matrix. The free parameters S_0^i, $i = 1, \cdots, d$ are again contained in [1;1.5]. We perform the offline phase by considering again the set of completion parameters listed in Table 5.3. The obtained results of the completion are now in Table 5.6 and Figure 5.8 shows the required size of Ω to go below the tolerance $tol = 10^{-2}$ for $n = 4$ and $tol = 10^{-3}$ for $n = 6$. We notice that the completion results are similar to the case of uncorrelated assets and that $|\Omega|$ scales again like $\mathcal{O}(d^2)$. The computational time to construct $\mathcal{C}$ was again measured to be less than 0.01 seconds for all choices of d and n.

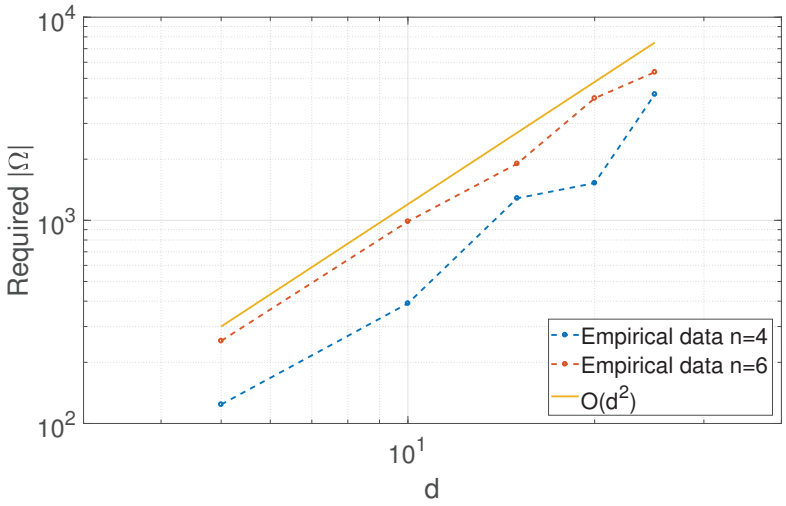

Figure 5.8 – Required size of Ω for the completion to go below $tol = 10^{-2}$ for $n = 4$ and $tol = 10^{-3}$ for $n = 6$. Case of correlated assets..

The online phase is performed similarly to the previous chapter, in particular we compute again 100 prices using the new method and the reference one. The MC parameters are set as before and the results are shown in Table 5.7. The performance of the new method in terms of accuracy and computational efficiency is similar to the one observed in the case of uncorrelated assets. To summarize, the new methodology achieves a very good performance for uncorrelated as well as for correlated assets.

Rank structure of $\mathcal{P}$

In this section we qualitatively analyze the rank structure of the tensor $\mathcal{P}$. For simplicity, we perform this analysis for the standard Monte Carlo approach (without any variance

	d	final $\lvert\Omega\rvert$	rel err on last Ω_C^{new}	completion time (s)
$n=4$	5	124	$1.86 \cdot 10^{-3}$	8.95
	10	390	$2.19 \cdot 10^{-4}$	65.73
	15	1284	$1.72 \cdot 10^{-7}$	118.73
	20	1526	$2.49 \cdot 10^{-8}$	168.20
	25	4172	$7.52 \cdot 10^{-9}$	215.44
$n=6$	5	255	$4.40 \cdot 10^{-4}$	66.54
	10	987	$2.06 \cdot 10^{-4}$	200.15
	15	1900	$1.79 \cdot 10^{-7}$	432.58
	20	3990	$1.82 \cdot 10^{-8}$	852.13
	25	5364	$2.88 \cdot 10^{-7}$	1335.76

	d	max r_μ reached	store(TT) (bytes)	store(full) (bytes)
$n=4$	5	5	2320	$2.50 \cdot 10^{4}$
	10	3	2440	$7.81 \cdot 10^{7}$
	15	5	8960	$2.44 \cdot 10^{11}$
	20	4	7040	$7.63 \cdot 10^{14}$
	25	3	8040	$2.38 \cdot 10^{18}$
$n=6$	5	5	2520	$1.34 \cdot 10^{5}$
	10	5	6832	$2.26 \cdot 10^{9}$
	15	4	6664	$3.80 \cdot 10^{13}$
	20	4	12040	$6.38 \cdot 10^{17}$
	25	4	8960	$1.07 \cdot 10^{22}$

Table 5.6 – Completion results on $\mathcal{P}$ for the basket option pricing problem in the Black-Scholes model. Case of correlated assets.

reduction technique). Assume that we have already simulated the realizations of the correlated geometric Brownian motions stored in the matrix M (Algorithm 5.7). Then, the price in the point $\mathbf{S}_0$ is given by the function

$$p : D \to \mathbb{R},$$
$$p(\mathbf{S_0}) := \frac{e^{-rT}}{N_S} \sum_{n=1}^{N_S} \left[w^\top (\mathbf{S}_0 \circ M(n,:)^\top) - K \right]^+,$$

where D is the hyper-rectangular domain for the interpolation, $M(n,:)$ is the n-th row of M and N_S is the number of Monte Carlo simulations. This expression can be rewritten

	d	time reference method (s)	time interpolation (s)	max abs error
$n = 4$	5	0.18	$0.50 \cdot 10^{-3}$	$1.39 \cdot 10^{-3}$
	10	0.20	$0.56 \cdot 10^{-3}$	$4.82 \cdot 10^{-4}$
	15	0.20	$0.70 \cdot 10^{-3}$	$2.82 \cdot 10^{-4}$
	20	0.23	$0.91 \cdot 10^{-3}$	$2.93 \cdot 10^{-4}$
	25	0.23	$1 \cdot 10^{-3}$	$4.30 \cdot 10^{-4}$
$n = 6$	5	1.85	$0.38 \cdot 10^{-3}$	$3.55 \cdot 10^{-4}$
	10	1.90	$0.57 \cdot 10^{-3}$	$5.58 \cdot 10^{-4}$
	15	1.99	$0.74 \cdot 10^{-3}$	$1.39 \cdot 10^{-4}$
	20	2.06	$0.90 \cdot 10^{-3}$	$1.41 \cdot 10^{-4}$
	25	2.15	$0.96 \cdot 10^{-3}$	$9.28 \cdot 10^{-5}$

Table 5.7 – Basket option prices computed via Chebyshev interpolation (combined methodology) versus MC reference method with 10^4 simulations for $n = 4$ and 10^5 simulations for $n = 6$. Case of correlated assets.

in the form

$$p(\mathbf{S_0}) = \frac{e^{-rT}}{N_S} \sum_{n=1}^{N_S} \left(\sum_{i=1}^{d} \alpha_i(n) S_0^i - K \right)^+,$$

where the $\alpha_i(n)$'s are coefficients multiplying S_0^i depending on the n-th simulation and on the i-th weight ω_i. The function p is piecewise affine in the variables S_0^i.

To explore the rank structure of $\mathcal{P}$ let us consider the case of a single Monte Carlo simulation $N_S = 1$. Then p is of the form

$$p(\mathbf{S_0}) = e^{-rT} \left(\sum_{i=1}^{d} \alpha_i S_0^i - K \right)^+.$$

Now we analyze three different cases. First, consider the case where the price is positive for any $\mathbf{S}_0$ in the hyper-rectangular D. Here, p is affine. This implies that the TT ranks are bounded by d. This follows from the fact that the CP rank (rank of the Canonical Polyadic Decomposition, see [82]) of $\mathcal{P}$, which is an upper bound for each r_μ in the TT ranks (see [59]), is equal to d. Second, if we observe a vanishing price for all $\mathbf{S}_0$ in the hyper-rectangular, then $\mathcal{P}$ is the zero-tensor, which has rank 0. These two cases obviously yield a low-rank structure of $\mathcal{P}$, a favorable case for the new combined methodology.

In the third case where p is only piecewise affine the situation is more complex and to

gain an intuition we consider the case $d = 2$, where p is of the form

$$p(S_0^1, S_0^2) = e^{-rT}(\alpha_1 S_0^1 + \alpha_2 S_0^2 - K)^+,$$

on a squared domain D. Now, define the set

$$L := \{(S_0^1, S_0^2) \in D \mid \alpha_1 S_0^1 + \alpha_2 S_0^2 - K = 0\}.$$

When L intersects the domain D it cuts it in two regions. Only if α_1, α_2 and K are of a specific form that leads L to be the diagonal of D, the rank of $\mathcal{P}$ is almost full. In the Monte Carlo simulation context, this special case is very unlikely. In all other cases, $\mathcal{P}$ exhibits a lower rank structure. In particular, we expect the rank to be the lower the more the sizes of the two regions differ.

In order to visualize these findings we consider three different pairs (α_1, α_2) together with $r = 0$, $K = 1$ and evaluate the corresponding p on the discretized $D = [1; 1.5]^2$ using 50 equidistant points in each direction. Figure 5.9 shows the sparsity pattern and the rank of the obtained matrices $\mathcal{P}$.

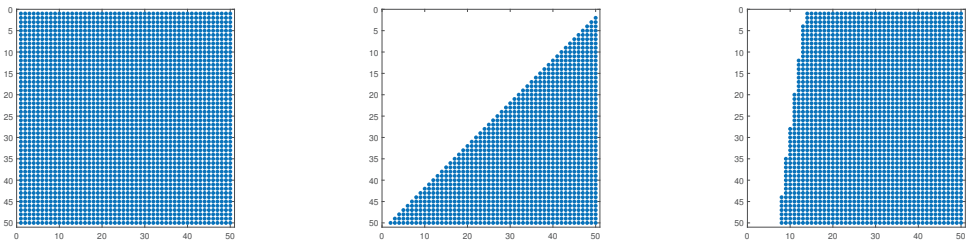

Figure 5.9 – Sparsity patterns and ranks for evaluated p on $D = [1; 1.5]^2$ for different values of (α_1, α_2). Left: $(\alpha_1, \alpha_2) = (0.9, 0.8)$ and rank $= 2$. Center: $(\alpha_1, \alpha_2) = (0.4, 0.4)$ and rank $= 49$. Right: $(\alpha_1, \alpha_2) = (0.1, 0.8)$ and rank $= 8$.

This qualitative explanation indicates that the rank structure of $\mathcal{P}$ depends on D. We expect the rank to be lower for domains D with an asymmetry with respect to the strike K. Next we construct $\mathcal{P}$ as in the experiments of Section 5.4.2 for $K = 1$, $d = 2$ and different interpolation orders n for both $D = [0.5; 1.5]^2$ and $D = [1; 1.5]^2$. In particular, we first construct the matrix M via Algorithm 5.7 with 10^5 simulations and subsequently compute $\mathcal{P}$ using Algorithm 5.8. In Figure 5.10 we display the decay of the singular values for all treated cases. As expected, the decay is faster for $D = [1; 1.5]^2$. However, also for $D = [0.5; 1.5]^2$ the decay of the singular values is reasonably fast. This implies

that the new methodology would be still beneficial in this case.

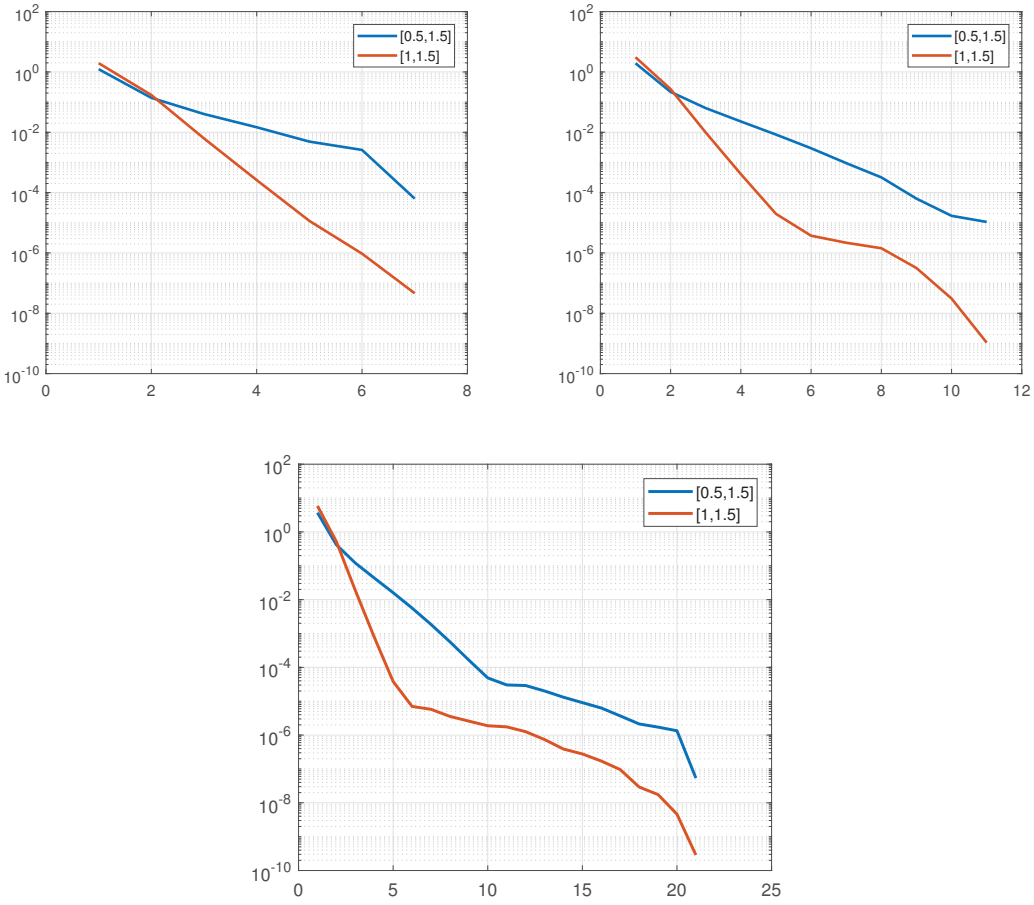

Figure 5.10 – Singular value decay of the matrix $\mathcal{P}$ for sampling intervals $[0,5;1,5]$ *and* $[1;1.5]$ *for different interpolation orders: Top left:* $n = 6$*, Top right:* $n = 10$*, Bottom:* $n = 20$*.*

5.5 Conclusion

We have presented a unified approach to efficiently compute parametric option prices. The starting point of our methodology was the Chebyshev interpolation technique developed in [46], which we briefly summarized in Section 5.3.1. We refined both the offline and the online phase to treat high-dimensional problems with parameter spaces up to dimension 25. We have exploited the low-rank structure (the notion of low-rank approximation is presented in Section 5.3.2) of the tensors involved in the interpolation procedure, which have been stored in the TT format (summarized in Section 5.3.3). In particular, we

have developed a completion technique (explained in Section 5.3.4) which allows us the construct the tensor $\mathcal{P}$, containing the option prices in the Chebshev tensor grid. All ingredients have been efficiently assembled to finally build a combined methodology, explained in Section 5.3.5.

In the last part of the chapter, Section 5.4, we have tested our approach in two different concrete option pricing settings: We have treated the American option pricing problem in the Heston model (Section 5.4.1) and the European basket option pricing problem in the d-dimensional Black-Scholes model (Section 5.4.2). Both examples show that our approach allows for a substantial gain in efficiency, while maintaining very accurate results, whose precision is comparable to the one of the considered reference methods. For instance, the interpolation of American option prices in 5 parameters accelerates the procedure by a factor of 75, when compared to the FD reference method [60]. For basket option pricing with 25 underlyings the efficiency gain reaches factors up to 4000. See Tables 5.2, 5.5, 5.7 and Figure 5.7 for further results. Finally, for both examples we qualitatively investigated the rank structure of $\mathcal{P}$, which confirmed that our initial low-rank assumption was indeed reasonable. For instance, for the American put, we obtain a compression factor of 115 of the completed tensor $\mathcal{P}$ with respect to the full one, with a relative error in the 5th digit only, see Table 5.1. For the basket option the full tensor containing prices in the Chebyshev grid is too large to be computed, however in Section 5.4.2 it is qualitatively explained why $\mathcal{P}$ is expected to have a low-rank structure. This is also confirmed by the compression rates observed in Tables 5.4 and 5.6 that go up to 3×10^{17}.

Seen the promising performance of this new approach and considering the fact that this methodology can be easily tailored to different problem settings, we expect it to be applicable in several domains in finance. For instance, pricing, calibration and sensitivity analysis in equity markets, fixed income and credit, and parameter uncertainty quantification are some of the possible domains of application.

6 Combining function approximation and Monte Carlo simulation for efficient option pricing

In this chapter we propose a methodology to compute single and multi-asset European option prices, and more generally expectations of scalar functions of (multivariate) random variables. This method is a generalization of the approach developed in [97] for the numerical computation of multivariate integrals with respect to the Lebesgue measure and its main idea is to combine the Monte Carlo simulation with function approximation, resulting in a variance reduction technique. The method offers an efficient way to deal with the high dimensionality arising when treating multi-asset options, which represents a big computational challenge as mentioned several times throughout the thesis, see e.g. Chapter 1 and Chapter 5, Section 5.1. Moreover, it can be applied to polynomial models.

The method developed by Nakatsukasa in [97] can be briefly described as follows. Consider the problem of computing a multivariate integral of the form

$$\int_{\Omega} f(\mathbf{x}) d\mathbf{x},$$

where Ω is the unit d-dimensional cube $[0,1]^d$ and f is an arbitrary L^2-integrable function. The algorithm, denoted by MCLS (Monte Carlo with Least Squares), is based on the principle "approximate and integrate" and mainly consists of three steps: i) generate N sample points $\{\mathbf{x}_i\}_{i=1}^N \in \Omega$, uniformly at random; ii) approximate the integrand f with a linear combination of a priori chosen basis functions $f(\mathbf{x}) \approx p(\mathbf{x}) := \sum_{j=0}^n c_j \phi_j(\mathbf{x})$, where the coefficients $\{c_j\}_{j=0}^n$ are computed by solving a least-squares problem of the form $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\mathbf{V}\mathbf{c} - \mathbf{f}\|_2$; iii) finally, integrate $p(\mathbf{x})$ to approximate the integral $\sum_{i=0}^n c_j \int_\Omega \phi_j(\mathbf{x}) d\mathbf{x} \approx \int_\Omega f(\mathbf{x}) d\mathbf{x}$.

Chapter 6. Combining function approximation and Monte Carlo simulation for efficient option pricing

In this work we extend MCLS and apply it to the problem of pricing single and multi-asset European options. Firstly, we generalize the approach to approximate integrals of the form

$$\int_E f(\mathbf{x}) d\mu(\mathbf{x}),$$

where $(E, \mathcal{A}, \mu)$ is a probability space. This is done in Section 6.1.

Secondly, we note that a computational bottleneck in MCLS is the storage requirement arising when solving the least-squares problem. Indeed, when the number of simulations N and of basis functions n are too large, it is not feasible to explicitly store the Vandermonde[1] matrix $\mathbf{V} = (\phi_i(\mathbf{x}_j))_{i=0,\dots,n;j=1,\dots,N}$, which is needed in step ii) to approximate the integrand f. This limits the number of basis functions, the dimension d, and the number of simulations for which the approach is feasible. In order to overcome this limitation we propose to combine MCLS with the randomized extended Kaczmarz (REK) algorithm [124] for solving the least-squares problem. Its benefit is that no explicit storage of $\mathbf{V}$ is needed in order to invoke the algorithm and this allows us to treat least-squares problems of big size. However, for REK to converge at a reasonably fast rate, the least-squares problem needs to be well conditioned, i.e., the condition number $\kappa_2(\mathbf{V})$ has to be sufficiently small. This can be obtained by applying the optimal sampling strategy proposed in [26] to MCLS in step i). We review this strategy in Section 6.1.1, while the randomized extended Kaczmarz algorithm is summarized in Section 6.1.2. We provide a cost and a convergence analysis in Section 6.2.

In Section 6.3 we apply MCLS to price single and multi-asset European options in polynomial models. The fact that the conditional moments are given in closed form via the moment formula (2.7) naturally suggests the choice of polynomials as basis functions $\{\phi_j\}_{j=0}^n$, for which step iii) of MCLS becomes very efficient. Here, we obtain good numerical results for different models and payoff profiles.

Lastly, in Section 6.4 we apply our extension of MCLS to a high-dimensional integration problem with respect to the Lebesgue measure, emphasizing that the limitations of the approach in [97] due to the high dimensionality have been indeed overcome.

[1] Note that the matrix $\mathbf{V}$ is not a Vandermonde matrix in the standard sense. In statistics, such a matrix is usually referred to as "design matrix". However, in the rest of the chapter we keep the name as in [97] and we call it Vandermonde matrix.

6.1 Method

In this section we introduce a methodology to compute the definite integral

$$I_\mu := \int_E f(\mathbf{x}) d\mu(\mathbf{x}),$$

for some probability space $(E, \mathcal{A}, \mu)$, $E \subseteq \mathbb{R}^d$, and for a function $f : E \to \mathbb{R}$, which we assume to be square-integrable, i.e., in

$$L^2_\mu = \{f : E \to \mathbb{R} \mid \|f\|^2_\mu = \int_E f(\mathbf{x})^2 d\mu(\mathbf{x}) < \infty, f \text{ measurable}\},$$

which is a Hilbert space with the inner product $\langle f, g\rangle_\mu = \int_E f(\mathbf{x})g(\mathbf{x})d\mu(\mathbf{x})$. As already mentioned, the method is an extension of the method proposed in [97] for integrals with respect to the Lebesgue measure.

To start, we choose a set of n basis functions $\{\phi_j\}_{j=1}^n$, with $\phi_0 \equiv 1$, which will be used to approximate the integrand f. Key to exploit the advantage of function approximation for integration is to choose basis functions $\{\phi_j\}_{j=0}^n$ that can be easily, possibly exactly, integrated, and that ideally the linear combination of $\{\phi_j\}_{j=0}^n$ approximates f well. For instance, polynomials can be a good choice as shown in the applications of Section 6.3 and Section 6.4. Then, the steps of MCLS are as follows. First, as in standard Monte Carlo methods, one generates N sample points $\{\mathbf{x}_i\}_{i=1}^N \in E$, according to μ. Second, the integrand f and the set of basis functions are evaluated at all simulated points $\{\mathbf{x}_i\}_{i=1}^N$ leading to the following linear least-squares problem:

$$\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \left\| \underbrace{\begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \dots & \phi_n(\mathbf{x}_1) \\ 1 & \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \dots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & & \\ 1 & \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \dots & \phi_n(\mathbf{x}_N) \end{bmatrix}}_{=:\mathbf{V}} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} - \underbrace{\begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_N) \end{bmatrix}}_{=:\mathbf{f}} \right\|_2, \tag{6.1}$$

which we denote as $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\mathbf{V}\mathbf{c} - \mathbf{f}\|_2$. Note that (6.1) can be seen as a discrete version of the projection problem $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|f - \sum_{j=0}^n c_j\phi_j\|_\mu$. Third, one solves (6.1), whose

solution is known to be explicitly given by

$$\hat{\mathbf{c}} = (\mathbf{V}^T\mathbf{V})^{-1}\mathbf{V}^T\mathbf{f}.$$

At this point, the linear combination $p(\mathbf{x}) := \sum_{j=0}^{n} \hat{c}_j \phi_j(\mathbf{x})$ is an approximation of f. Finally, the last step consists of computing the integral of the approximant p, and I_μ is approximated by

$$I_\mu \approx \hat{I}_{\mu,N} = \int_E p(\mathbf{x}) d\mu(\mathbf{x}) = \hat{c}_0 + \sum_{j=1}^{n} \hat{c}_j \int_E \phi_j(\mathbf{x}) d\mu(\mathbf{x}). \tag{6.2}$$

We summarize the procedure in Algorithm 6.1.

We remark that there is an interesting connection between MCLS and the standard Monte Carlo (MC) method: If one takes $n = 0$, i.e. one approximates f with a constant function, the resulting approximation is the solution of the least-squares problem

$$\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \left\| \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} c_0 - \begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_N) \end{bmatrix} \right\|_2,$$

which is exactly given by $\hat{c}_0 := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$, the standard Monte Carlo estimator. We recall that in the standard MC method, for sufficiently large N the error scales like (see e.g. [21])

$$\frac{\left(\int_E (f(\mathbf{x}) - I_\mu)^2 d\mu(\mathbf{x})\right)^{\frac{1}{2}}}{\sqrt{N}},$$

which is equivalent to

$$\frac{\min_{c\in\mathbb{R}} \|f - c\|_\mu}{\sqrt{N}} =: \frac{\sigma(f)}{\sqrt{N}}. \tag{6.3}$$

The quantity $(\sigma(f))^2$ is usually referred to as *the variance of* f. This relation between MC and MCLS leads to an asymptotic error analysis, which we detail in Section 6.2.1. This connection can be also exploited in order to increase the speed of convergence by combining it with quasi-Monte Carlo. In [97] also other ways to speed up the procedure are proposed, for example by an adaptive choice of the basis functions (MCLSA).

Algorithm 6.1 Generalized MCLS

Input: Function f, basis functions $\{\phi_j\}_{j=1}^n$, $\phi_0 \equiv 1$, integer $N(> n)$, probability distribution $\mu(\mathbf{x})$ over domain E.
Output: Approximate integral $\hat{I}_{\mu,N} \approx \int_E f(\mathbf{x})d\mu(\mathbf{x})$
1: Generate sample points $\{\mathbf{x}_i\}_{i=1}^N \in E$, according to μ.
2: Evaluate $f(\mathbf{x}_i)$ and $\phi_j(\mathbf{x}_i)$, for $i = 1, \ldots, N$ and $j = 1, \ldots, n$.
3: Solve the least-squares problem (6.1) for $\hat{\mathbf{c}} = [\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_n]^T$.
4: Compute $\hat{I}_{\mu,N} = \hat{c}_0 + \sum_{j=1}^n \hat{c}_j \int_E \phi_j(\mathbf{x})d\mu(\mathbf{x})$.

It is observed in [97] that the method performs well for dimensions d up to $d = 6$. For higher dimensions solving the least-squares problem (6.1) becomes computationally expensive, this is mainly due to two effects:

(i) The size of $\mathbf{V}$, being $N \times (n+1)$, rapidly becomes very large, posing memory limitations.

(ii) The condition number of $\mathbf{V}$ typically gets large.

In the following we address these issues by combining MCLS with an optimal sampling strategy and with the randomized extended Kaczmarz algorithm for solving the least-squares problem.

6.1.1 Well conditioned least-squares problem via optimal sampling

It is crucial that the matrix $\mathbf{V}$ in (6.1) is well conditioned, from both a computational and a function approximation perspective. Computationally, an ill-conditioned $\mathbf{V}$ means the least-squares problem is harder to solve using e.g. the conjugate gradient method (see e.g. [52, Chapter 11]), and the randomized Kaczmarz method described in Section 6.1.2. From an approximation viewpoint, $\mathbf{V}$ having a large condition number $\kappa_2(\mathbf{V})$ implies that the function approximation error (in the continuous setting) $\|f - \sum_{j=0}^n \hat{c}_j\phi_j\|_\mu$ could be as large as $\kappa_2(\mathbf{V})\|f - \sum_{j=0}^n c_j^*\phi_j\|_\mu$ (see [97, Proposition 5.2]), where $\mathbf{c}^* := \operatorname{argmin}_{\mathbf{c}\in\mathbb{R}^{n+1}}\|f - \sum_{j=0}^n c_j\phi_j\|_\mu$. Hence in practice we devise the MCLS setting (choice of ϕ and sampling strategy) so that $\mathbf{V}$ is well conditioned with high probability.

A first step to attempt to obtain a well conditioned Vandermonde matrix $\mathbf{V}$ is to choose the basis $\{\phi_j\}_{j=0}^n$ to be orthonormal with respect to the scalar product $\langle\cdot\rangle_\mu$, for instance

by applying a Gram-Schmidt orthonormalization procedure. Then, the strong law of large numbers (see e.g. [57, Chapter 6]) yields

$$\frac{1}{N}(\mathbf{V}^T\mathbf{V})_{i+1,j+1} = \frac{1}{N}\sum_{l=1}^{N}\phi_i(\mathbf{x}_l)\phi_j(\mathbf{x}_l) \xrightarrow{p} \int_E \phi_i(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = \delta_{ij}$$

as $N \to \infty$. Therefore, for a large number of samples N we expect $\frac{1}{N}\mathbf{V}^T\mathbf{V}$ to be close to the identity matrix $\mathbf{Id}_{n+1} \in \mathbb{R}^{(n+1)\times(n+1)}$. This implies that $\kappa_2(\mathbf{V})$ gets close to 1. In practice, however, the condition number often is large even if the basis functions are orthonormal. This is because the number N of sample points required to obtain a well conditioned $\mathbf{V}$ might be very large. For example, if we consider the one-dimensional interval $E = [-1, 1]$ with the uniform probability measure and an orthonormal basis of Legendre polynomials, one can show that at least $N = \mathcal{O}(n^2 \log(n))$ sample points are needed to obtain a well conditioned $\mathbf{V}$. This example and others are discussed in [24, 26].

To overcome this problem and lower the required sample size N to obtain a well conditioned problem, Cohen and Migliorati [26] introduce a *weighted* sampling for least-squares fitting. Its use for MCLS was suggested in [97], which we summarize here. Define the nonnegative function w via

$$\frac{1}{w(\mathbf{x})} = \frac{\sum_{j=0}^{n}\phi_j(\mathbf{x})^2}{n+1}. \tag{6.4}$$

The orthonormality of $\{\phi_j\}_{j=0}^n$ implies that $\frac{1}{w} \geq 0$ on E and $\int_E \frac{1}{w(\mathbf{x})}d\mu(\mathbf{x}) = 1$. We then take samples $\{\tilde{\mathbf{x}}_i\}_{i=1}^N$ according to $\frac{d\mu}{w}$. Intuitively this means that we sample more often in areas where $\sum_{i=0}^{n}\phi_i(\mathbf{x})^2$ takes large values.

The least-squares problem (6.1) with the samples $\sim \frac{d\mu}{w}$ becomes

$$\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{\mathbf{W}}(\mathbf{V}\mathbf{c} - \mathbf{f})\|_2, \tag{6.5}$$

where $\sqrt{\mathbf{W}} = \mathrm{diag}(\sqrt{w(\tilde{\mathbf{x}}_1)}, \sqrt{w(\tilde{\mathbf{x}}_2)}, \ldots, \sqrt{w(\tilde{\mathbf{x}}_N)})$, and $\mathbf{V}, \mathbf{f}$ are as before in (6.1) with $\mathbf{x} \leftarrow \tilde{\mathbf{x}}$. This is again a least-squares problem $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\widetilde{\mathbf{V}}\mathbf{c} - \tilde{\mathbf{f}}\|_2$, with coefficient matrix $\widetilde{\mathbf{V}} := \sqrt{\mathbf{W}}\mathbf{V}$ and right-hand side $\tilde{\mathbf{f}} := \sqrt{\mathbf{W}}\mathbf{f}$, whose solution is $\hat{\mathbf{c}} = (\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}})^{-1}\widetilde{\mathbf{V}}^T\sqrt{\mathbf{W}}\mathbf{f}$. With high probability, the matrix $\widetilde{\mathbf{V}}$ is then well conditioned, provided that $N \gtrsim n \log n$, see Theorem 2.1 in [26].

Remark 6.1. *Note that the left-multiplication by $\sqrt{\mathbf{W}}$ forces all the rows of $\widetilde{\mathbf{V}}$ to have the same norm (here $\sqrt{n+1}$); a property that proves useful in Section 6.1.2.*

A simple strategy to sample from w is as follows: for each of the N samples, choose a basis function ϕ_j from $\{\phi_j\}_{j=0}^n$ uniformly at random, and sample from a probability distribution proportional to ϕ_j^2. We refer to [62] for more details.

6.1.2 Randomized extended Kaczmarz to solve the least-squares problem

A standard least-squares solver that uses the QR factorization [52, Ch. 5] costs $\mathcal{O}(Nn^2)$ operations, which quickly becomes prohibitive (relative to standard MC) when $n \gg 1$. As an alternative, the conjugate gradient method (CG) applied to the normal equation $(\mathbf{V}^T\mathbf{V})\mathbf{c} = \mathbf{V}^T\mathbf{f}$ is suggested in [97]. For $\kappa_2(\mathbf{V}) = \mathcal{O}(1)$ this reduces the computational cost to $\mathcal{O}(Nn)$. However, CG still requires to build and possibly store all the $\mathcal{O}(Nn)$ entries of $\mathbf{V}$. Indeed in practice, building and storing the matrix $\mathbf{V}$ becomes a major bottleneck in MCLS.

To overcome this issue, here we suggest a further alternative, the randomized extended Kaczmarz algorithm developed by Zouzias and Freris [124]. REK is a randomized iterative method to solve least-squares problems. It builds upon Strohmer and Vershynin's pioneering work [115] and Needell's extension to inconsistent systems [98], and converges to the minimum-norm solution by simultaneously performing projection and solution refinement at each iteration. The convergence is geometric in expectation and, as already observed in [115], Kaczmarz methods can sometimes even outperform the conjugate gradient method in speed for well conditioned systems. A block version of REK was introduced in [99], which sometimes additionally improves the performance.

Here we focus on REK and consider its application to MCLS. A pseudocode of REK is given in Algorithm 6.2. MATLAB notation is used, in which $\mathbf{V}(:, j)$ denotes the jth column of $\mathbf{V}$ and $\mathbf{V}(i, :)$ the ith row. The $\mathbf{z}^{(k)}$ iterates are the projection steps, which converge to $\mathbf{f}^{\perp}$, the part of $\mathbf{f}$ that lies in the orthogonal complement of $\mathbf{V}$'s column space. The vectors $\mathbf{c}^{(k)}$ are the solution iterates. REK works by simultaneously projecting out the $\mathbf{f}^{\perp}$ component while refining the least-squares solution. The solution refinement step (Step 6 in Algorithm 6.2) without $\mathbf{z}^{(k)}$, i.e.

$$\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} - \frac{\mathbf{V}(i_k, :)^{\top}\mathbf{c}^{(k)} - f_{i_k}}{\|\mathbf{V}(i_k, :)\|_2^2}\mathbf{V}(i_k, :),$$

Algorithm 6.2 REK: Randomized extended Kaczmarz method

Input: $\mathbf{V} \in \mathbb{R}^{N\times(n+1)}$ and $\mathbf{f} \in \mathbb{R}^N$.
Output: Approximate solution $\mathbf{c}$ for $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\mathbf{V}\mathbf{c}-\mathbf{f}\|_2$
1: Initialize $\mathbf{c}^{(0)} = 0$ and $\mathbf{z}^{(0)} = \mathbf{f}$
2: **for** $k = 1, 2, \ldots, M$ **do**
3: Pick $i = i_k \in \{1, \ldots, N\}$ with probability $\|\mathbf{V}(i,:)\|_2^2/\|\mathbf{V}\|_F^2$
4: Pick $j = j_k \in \{1, \ldots, n+1\}$ with probability $\|\mathbf{V}(:,j)\|_2^2/\|\mathbf{V}\|_F^2$
5: Set $\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} - \frac{\mathbf{V}(:,j_k)^\top \mathbf{z}^{(k)}}{\|\mathbf{V}(:,j_k)\|_2^2}\mathbf{V}(:,j_k)$
6: Set $\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} + \frac{f_{i_k} - z_{i_k}^{(k)} - \mathbf{V}(i_k,:)^\top \mathbf{c}^{(k)}}{\|\mathbf{V}(i_k,:)\|_2^2}\mathbf{V}(i_k,:)$
7: **end for**
8: $\mathbf{c} = \mathbf{c}^{(M)}$

corresponds to the orthogonal projection of the current estimate onto the affine hyperplane defined by the i_kth equation of the least-squares problem, i.e. $\mathbf{V}(i_k,:)^\top\mathbf{c} = \mathbf{f}_{i_k}$. Since the index i is chosen at random at every step, we can interpret REK as a particular stochastic gradient descent method, in which at each iteration we minimize the objective function in the i_kth component.

Let us comment on REK (Algorithm 6.2) and its implementation, particularly in the MCLS context:

- Employing the optimal weighted sampling strategy of Section 6.1.1 significantly simplifies Algorithm 6.2. Following Remark 6.1, the norm of the rows of $\widetilde{\mathbf{V}}$ are constant and equal to $\sqrt{n+1}$. This also implies that $\|\widetilde{\mathbf{V}}\|_F^2 = N(n+1)$. The index i_k in line 3 is therefore simulated uniformly at random. This has a practical significance in MCLS as the probability distribution $(\|\mathbf{V}(i,:)\|_2^2/\|\mathbf{V}\|_F^2)_{i=1,\ldots,N}$ does not have to be computed before starting the REK iterates. This results in (a potentially enormous) computational reduction; an additional benefit of using the optimal sampling strategy, besides improving conditioning.

- The number of iterations M is usually not chosen a priori but by checking convergence of $\mathbf{c}^{(k)}$ infrequently. The suggestion in [124] is to check every $8\min(N,n)$ iterations for the conditions

$$\frac{\|\mathbf{V}\mathbf{c}^{(k)} - (\mathbf{f} - \mathbf{z}^{(k)})\|_2}{\|\mathbf{V}\|_F\|\mathbf{c}^{(k)}\|_2} \le \varepsilon, \qquad \text{and} \qquad \frac{\|\mathbf{V}^T\mathbf{z}^{(k)})\|_2}{\|\mathbf{V}\|_F\|\mathbf{c}^{(k)}\|_2} \le \varepsilon \tag{6.6}$$

  for a prescribed tolerance $\varepsilon > 0$. In our numerical examples in Sections 6.3 and 6.4

we adopt this strategy to check the convergence of the algorithm and we set the tolerance value to $\varepsilon = 10^{-5}$.

- A significant advantage of REK is that it renders unnecessary the storage of the whole matrix $\mathbf{V}$: only the i_kth row and the j_kth column are needed, taking $\mathcal{O}(N)$ memory cost. In practice, one can even sample in an online fashion: early samples can be discarded once the REK update is completed. Note that, also to compute the Frobenius norm $\|\mathbf{V}\|_F$ for an arbitrary input matrix, the storage of the whole matrix is unnecessary. Indeed, the norm can be computed by exploiting the relation $\|\mathbf{V}\|_F^2 = \sum_{i=0}^{N} \|\mathbf{V}(i,:)\|_2^2$, in which we sequentially sum up the squared 2-norm of each row, which is discarded right after.

The convergence of REK is known to be geometric in the expected mean squared sense [124, Thm 4.1]: after M iterations, we have

$$\widetilde{\mathbb{E}}[\|\mathbf{c}^{(M)} - \hat{\mathbf{c}}\|_2^2] \leq \left(1 - \frac{(\sigma_{\min}(\mathbf{V}))^2}{\|\mathbf{V}\|_F^2}\right)^{\lfloor \frac{M}{2} \rfloor} (1 + 2\kappa_2^2(\mathbf{V}))\|\hat{\mathbf{c}}\|_2^2, \tag{6.7}$$

where $\hat{\mathbf{c}}$ is the solution for $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\mathbf{V}\mathbf{c} - \mathbf{f}\|_2$ and the expectation is taken over the random choices of the algorithm. When $\mathbf{V}$ is close to having orthonormal columns (as would hold with the optimal sampling and/or $N \to \infty$ with orthonormal basis functions ϕ), the convergence in (6.7) becomes $\mathcal{O}((1 - \frac{1}{n})^{\frac{M}{2}})$.

The cost of REK is analyzed in [124, Section 4.3]. If the stopping criterion (6.6) is chosen, then it is shown that the expected number of arithmetic operations of REK is proportional to the number of non-zero elements of $\mathbf{V}$ (which is bounded by $N(n+1)$) times the square condition number of $\mathbf{V}$. Therefore, in the case $\kappa_2(\mathbf{V}) = \mathcal{O}(1)$, the cost of REK is of the same order as the one of the CG algorithm. This fact will prove useful in our cost analysis in Section 6.2.

For the applications we consider, our experiments suggest that conjugate gradients applied to the normal equation is faster than Kaczmarz, so we recommend CG whenever it is feasible. However, as mentioned above, an advantage of (extended) Kaczmarz is that there is no need to store the whole matrix to execute the iterations. For these reasons, we suggest to choose the solver for the LS problem (6.1) according to the scheme shown in Figure 6.1. Preliminary numerical experiments have shown that the threshold 10 for $\kappa_2(\mathbf{V})$ is a good choice.

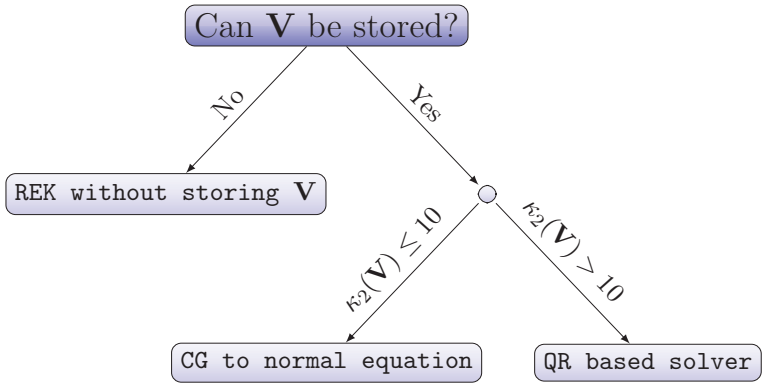

Figure 6.1 – Choice of algorithm to solve the least-squares problem.

6.2 Convergence and cost analysis

In this section we first present convergence results, on which basis we will derive a cost analysis.

6.2.1 Convergence

First, we obtain a convergence result and consequently asymptotic confidence intervals, applying the central limit theorem (CLT). The following statement and proof is a straightforward generalization of [97, Theorems 3.1 and 5.1] for an arbitrary integrating probability measure μ.

Proposition 6.2. *Fix n and the L^2_μ-basis functions $\{\phi_j\}_{j=0}^n$ and let either $w = 1$ or w as in* (6.4)*. Then with the weighted sampling $\frac{d\mu}{w}$, the corresponding MCLS estimator $\hat{I}_{\mu,N}$ in* (6.2)*, as $N \to \infty$ we have*

$$\sqrt{N}(\hat{I}_{\mu,N} - I_\mu) \xrightarrow{d} \mathcal{N}(0, \min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f - \sum_{j=0}^{n} c_j\phi_j)\|_\mu^2),$$

where $\xrightarrow{d}$ denotes convergence in distribution.

Proof. Note that the approximate function $\sum_{j=0}^n \hat{c}_j\phi_j$ and thus $\hat{I}_{\mu,N}$ only depends on the span of the basis functions $\{\phi_j\}_{j=0}^n$ and not on the specific choice of the basis. Therefore, without loss of generality we can assume that the chosen basis functions $\{\phi_j\}_{j=0}^n$ form an

orthonormal basis (ONB) in L^2_μ, i.e. $\int_E \phi_i(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = \delta_{ij}$.

We decompose the function f into a sum of orthogonal terms

$$f = \sum_{j=0}^{n} c_j^* \phi_j + g =: f_1 + g, \tag{6.8}$$

where g satisfies $\int_E g(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = 0$ for all $j = 0, \cdots, n$. Note that $\|g\|_\mu = \min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|f - \sum_{j=0}^n c_j\phi_j\|_\mu$. Assume now that we sample according to $\frac{d\mu}{w}$ and obtain the points $\{\tilde{\mathbf{x}}_i\}_{i=1}^N$. Then, the vector of sample values in the weighted least-squares problem can be decomposed as

$$\tilde{\mathbf{f}} = [\tilde{f}_1(\tilde{\mathbf{x}}_1) + \tilde{g}(\tilde{\mathbf{x}}_1), \ldots, \tilde{f}_1(\tilde{\mathbf{x}}_N) + \tilde{g}(\tilde{\mathbf{x}}_N)]^T = \widetilde{\mathbf{V}}\mathbf{c}^* + \tilde{\mathbf{g}},$$

where $\widetilde{\mathbf{V}}$ and $\tilde{f}$ are defined as in (6.5) and $\tilde{g} := \sqrt{w}g$ and hence

$$\tilde{\mathbf{g}} = [\sqrt{w(\tilde{\mathbf{x}}_1)}g(\tilde{\mathbf{x}}_1), \ldots, \sqrt{w(\tilde{\mathbf{x}}_N)}g(\tilde{\mathbf{x}}_N)].$$

Let $\hat{\mathbf{c}}$ be again the least-squares solution to (6.5), then

$$\hat{\mathbf{c}} = \mathrm{argmin}_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\widetilde{\mathbf{V}}\mathbf{c} - (\widetilde{\mathbf{V}}\mathbf{c}^* + \tilde{\mathbf{g}})\|_2 = (\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}})^{-1}\widetilde{\mathbf{V}}^T(\widetilde{\mathbf{V}}\mathbf{c}^* + \tilde{\mathbf{g}}) = \mathbf{c}^* + (\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}})^{-1}\widetilde{\mathbf{V}}^T\tilde{\mathbf{g}},$$

where the second summand is exactly $\mathbf{c}_g := \mathrm{argmin}_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\widetilde{\mathbf{V}}\mathbf{c} - \tilde{\mathbf{g}}\|_2$. It thus follows that the integration error is $\hat{I}_{\mu,N} - I_\mu = c_{g,0} = [1, 0, \ldots, 0](\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}})^{-1}\widetilde{\mathbf{V}}^T\tilde{\mathbf{g}}$.

Now by the strong law of large numbers we have

$$\begin{aligned}\frac{1}{N}(\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}})_{i+1,j+1} =& \frac{1}{N}\sum_{l=1}^{N} w(\tilde{\mathbf{x}}_l)\phi_i(\tilde{\mathbf{x}}_l)\phi_j(\tilde{\mathbf{x}}_l) \\ &\to \int_E w(\tilde{\mathbf{x}})\phi_i(\tilde{\mathbf{x}})\phi_j(\tilde{\mathbf{x}})\frac{d\mu(\tilde{\mathbf{x}})}{w(\tilde{\mathbf{x}})} = \int_E \phi_i(\tilde{\mathbf{x}})\phi_j(\tilde{\mathbf{x}})d\mu(\tilde{\mathbf{x}}) = \delta_{ij}\end{aligned}$$

almost surely and in probability as $N \to \infty$, by the orthonormality of $\{\phi_j\}_{j=0}^n$. Therefore we have $\frac{1}{N}\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}} \xrightarrow{p} \mathbf{Id}_{n+1}$ as $N \to \infty$, where $\mathbf{Id}_{n+1}$ denotes the identity matrix in $\mathbb{R}^{(n+1)\times(n+1)}$. Moreover, $\sqrt{N}\left(\frac{1}{N}\sum_{i=1}^N w(\tilde{\mathbf{x}}_i)g(\tilde{\mathbf{x}}_i)\right) \xrightarrow{d} Z \sim \mathcal{N}(0, \|\sqrt{w}g\|_\mu^2)$ for $N \to \infty$ by the central limit theorem, where we used the fact $\int_E g(\mathbf{x})d\mu(\mathbf{x}) = 0$ for the mean and $\int_E (w(\tilde{x})g(\tilde{\mathbf{x}}))^2\frac{d\mu(\tilde{\mathbf{x}})}{w(\tilde{\mathbf{x}})} = \|\sqrt{w}g\|_\mu^2$ for the variance. Thanks to Slutsky's theorem (see e.g.

[57, Chapter 5]) we finally obtain

$$\sqrt{N}(\hat{I}_{\mu,N} - I_\mu) = \sqrt{N}[1, 0, \ldots, 0]^T \frac{1}{N}(\frac{1}{N}\widetilde{\mathbf{V}}^T\widetilde{\mathbf{V}})^{-1}\widetilde{\mathbf{V}}^T\tilde{\mathbf{g}} \xrightarrow{d} \mathcal{N}(0, \|\sqrt{w}g\|_\mu^2).$$

□

The above proposition shows that the MCLS estimator yields an approximate integral $\hat{I}_{\mu,N}$ that asymptotically (for $N \to \infty$ and $\{\phi_j\}_{j=1}^n$ fixed) satisfies[2]

$$\mathbb{E}[|\hat{I}_{\mu,N} - I_\mu|] \approx \frac{\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f - \sum_{j=0}^n c_j\phi_j)\|_\mu}{\sqrt{N}}, \tag{6.9}$$

highlighting the fact that the asymptotic error is still $\mathcal{O}(1/\sqrt{N})$ (as in the standard MC), but with variance $(\sigma(f))^2$ reduced from $\min_{c\in\mathbb{R}} \|f - c\|_2^2$ (standard MC, see (6.3)) to $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f - \sum_{j=0}^n c_j\phi_j)\|_\mu^2$ (MCLS). In other words, the variance is reduced thanks to the approximation of the function f and the constant in front of the $\mathcal{O}(1/\sqrt{N})$ convergence in MCLS is equal to the function approximation error in the L_μ^2 norm.

After solving the least-squares problem (6.5), the variance $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f - \sum_{j=0}^n c_j\phi_j)\|_\mu^2$ can be estimated via[3]

$$\widetilde{\sigma}_{LS}^2 := \frac{1}{N-n-1}\sum_{i=1}^N (w(\tilde{\mathbf{x}}_i))^2(f(\tilde{\mathbf{x}}_i) - p(\tilde{\mathbf{x}}_i))^2 = \frac{1}{N-n-1}\|\mathbf{W}(\mathbf{V}\hat{\mathbf{c}} - \mathbf{f})\|_2^2, \tag{6.10}$$

where the sampling points $\tilde{\mathbf{x}}_i, i = 1, \cdots, N$ are taken according to $\frac{d\mu}{w}$. This leads to approximate confidence intervals, for example the 95% confidence interval is approximately given by

$$\left[\hat{I}_{\mu,N} - 1.96\frac{\widetilde{\sigma}_{LS}}{\sqrt{N}}, \hat{I}_{\mu,N} - 1.96\frac{\widetilde{\sigma}_{LS}}{\sqrt{N}}\right]. \tag{6.11}$$

As explained in [97], the MCLS estimator is not unbiased, in the sense that $\mathbb{E}[\hat{I}_{\mu,N}] \neq I_\mu$. However, one can show along the same lines as in the proof of [97, Proposition 3.1] that with the MCLS estimator $\hat{I}_{\mu,N}$ with n and $\{\phi_j\}_{j=0}^n$ fixed, one has

$$|I_\mu - \mathbb{E}[\hat{I}_{\mu,N}]| = \mathcal{O}\left(\frac{1}{N}\right).$$

[2] We use the notation "$\approx$" with the statement "for $N \to \infty$" to mean that the relation holds for sufficiently large N. E.g. (6.9) means $\mathbb{E}[|\hat{I}_{\mu,N} - I_\mu|] = \frac{\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f - \sum_{j=0}^n c_j\phi_j)\|_\mu}{\sqrt{N}} + o(\frac{1}{\sqrt{N}})$ for $N \to \infty$.

[3] This approximation is commonly used in linear regression, see e.g. [65].

This shows that the bias is of a smaller order than the error.

In the case of optimal weighting, we moreover have a finite sample error bound, which is presented in [26, Theorem 2.1 (iv)]. Note that as this is not only an asymptotic result, it is especially useful in practice. We review it in the following proposition.

Proposition 6.3. *Assume that we adopt the weighted sampling* $\frac{d\mu}{w}$*. For any* $r > 0$*, if* n *and* N *are such that* $n \leq \kappa \frac{N}{\log(N)} - 1$ *for* $\kappa = \frac{1-\log(2)}{2+2r}$*, then*

$$\mathbb{E}[\|f - \tilde{p}\|_\mu^2] \leq \left(1 + \frac{4\kappa}{\log(N)}\right) \min_{\mathbf{c} \in \mathbb{R}^{n+1}} \|f - \sum_{j=0}^{n} c_j \phi_j\|_\mu^2 + 2\|f\|_\mu^2 N^{-r}, \tag{6.12}$$

where $\tilde{p}$ *is defined as*

$$\tilde{p} := \begin{cases} p, & \text{if } \|\frac{1}{N}\mathbf{V}^T\mathbf{V} - \mathbf{I}\|_2 \leq \frac{1}{2} \\ 0, & \text{otherwise,} \end{cases}$$

with $p = \sum_{j=0}^{n} \hat{c}_j \phi_j$*, for* $\hat{\mathbf{c}}$ *being the solution of* (6.5)*.*

We note the slight difference between p and $\tilde{p}$; this is introduced to deal with the tail case in which $\mathbf{V}$ becomes ill-conditioned (which happens with low probability). This is used for a theoretical purpose, but in practice, this modification is not necessary and we do not employ it in our experiments.

Proposition 6.3 allows us to define a non-asymptotic, proper bound for the expected error we commit when estimating the vector $\mathbf{c}^*$, solving the LS problem (6.5). To see this, we first decompose the function f into a sum of orthogonal terms as in (6.8). Then,

$$\begin{aligned} \mathbb{E}[\|f - \sum_{j=0}^{n} \hat{c}_j \phi_j\|_\mu^2] &= \mathbb{E}[\|f - f_1 - \sum_{j=0}^{n} \hat{c}_j \phi_j + f_1\|_\mu^2] \\ &= \|f - f_1\|_\mu^2 + \mathbb{E}[\|\mathbf{c}^* - \hat{\mathbf{c}}\|_\mu^2] = \|g\|_\mu^2 + \mathbb{E}[\|\mathbf{c}^* - \hat{\mathbf{c}}\|_\mu^2]. \end{aligned}$$

This, together with the bound (6.12) yields

$$\mathbb{E}[\|\mathbf{c}^* - \hat{\mathbf{c}}\|_\mu^2] \leq \frac{4\kappa}{\log(N)} \min_{\mathbf{c} \in \mathbb{R}^{n+1}} \|f - \sum_{j=0}^{n} c_j \phi_j\|_\mu^2 + 2\|f\|_\mu^2 N^{-r}. \tag{6.13}$$

When we are primarily interested in integration, we aim at an upper bound for the expected error of the first component of $\mathbf{c}^* - \hat{\mathbf{c}}$. The bound (6.13) clearly holds for the

first component and this gives us a bound for $\mathbb{E}[|\hat{I}_{\mu,N} - I_\mu|^2]$. Intuitively, we expect that the error in the elements of $\mathbf{c}^* - \hat{\mathbf{c}}$ are not concentrated in any of the components. This suggests a heuristic bound

$$\mathbb{E}[|\hat{I}_{\mu,N} - I_\mu|^2] \lessapprox \frac{1}{n}\left(\frac{4\kappa}{\log(N)} \min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|f - \sum_{j=0}^{n} c_j\phi_j\|_\mu^2 + 2\|f\|_\mu^2 N^{-r}\right). \tag{6.14}$$

This argument has already been proposed in [97]. A rigorous argument still remains an open problem. Observing that the first term of the right hand side is the dominant one (for $N \to \infty$) and assuming $n \approx \frac{N}{\log(N)}$, we can see that the heuristic bound (6.14) matches the asymptotic result derived in Proposition 6.2.

6.2.2 Cost analysis

The purpose of this section is to reveal the relationship between the error vs. cost (in flops). The cost of MCLS is analyzed in [97] and in Table 6.1 we report a cost and error comparison between MC and MCLS as given in Table 3.1 in [97]. Here, we highlight some cases for which MCLS outperforms MC in terms of accuracy or cost.

Remark 6.4. *Note that the cost of MCLS in the Table 6.1 is reported to be $C_f N + \mathcal{O}(Nn)$. As already mentioned at the beginning of Section 6.1.2, this reflects the cost of MCLS when applying the CG algorithm to solve the least-squares problem (whenever $\kappa_2(\mathbf{V}) = \mathcal{O}(1)$). In the case that we combine MCLS with the REK algorithm and $\kappa_2(\mathbf{V}) = \mathcal{O}(1)$, which happens with high probability whenever the optimal sampling strategy is used (see [26, Theorem 2.1]), the cost is also given by $C_f N + \mathcal{O}(Nn)$. This is shown in [124, Lemma 9] and in the subsequent discussion, and it has been already mentioned at the end of Section 6.1.2. The following cost analysis includes therefore the two options CG and REK.*

First, consider the situation of a limited budget of sample points N that cannot be increased further, and the goal is to approximate the integral I_μ in the best possible way. This is a typical task in financial institutions. For instance, in portfolio risk management, simulation can be extremely expensive because a large number of risk factors and positions contribute to the company's portfolio. In this case even if MCLS is more expensive than MC (second column of Table 6.1), MCLS is preferable to MC as it yields a more accurate approximation (third column of Table 6.1).

	Cost	Convergence
MC	$C_f N$	$\frac{1}{\sqrt{N}} \min_{c\in\mathbb{R}} \|f-c\|_\mu$
MCLS	$C_f N + \mathcal{O}(Nn)$	$\frac{1}{\sqrt{N}} \min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f-\sum_{j=0}^{n} c_j\phi_j)\|_\mu$

Table 6.1 – Comparison between MC and MCLS. N is the number of sample points and C_f denotes the cost for evaluating f at a single point. As explained in the Remark 6.4, the cost of MCLS represented in this table refers to MCLS combined with CG or REK.

Second, we show under mild conditions that MCLS also asymptotically becomes more accurate than MC at the same cost. This can be of practical relevance whenever the integral I_μ needs to be computed at a very high accuracy and one is able to spend a high computational cost. Let us fix some notation:

$$
\begin{aligned}
&e_n := \min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f-\sum_{j=0}^{n} c_j\phi_j)\|_\mu \text{ for } n \geq 0,\\
&\mathrm{Cost}_{MC}(N) := C_f N,\\
&\mathrm{Cost}_{MCLS}(N',n) := C_f N' + C_M N' n \text{ for some } C_M > 0,\\
&\mathrm{error}_{MC}(N) := \frac{e_0}{\sqrt{N}},\\
&\mathrm{error}_{MCLS}(N',n) := \frac{e_n}{\sqrt{N'}},
\end{aligned}
$$

where the last two definitions reflect the asymptotic error behaviour for large N and N' (for a fixed n), depicted in Table 6.1. We are now in the position to present the result.

Proposition 6.5. *Assume that $e_n = o\Big(\frac{1}{\sqrt{n}}\Big)$. Then there exists $\tilde{n} \in \mathbb{N}$ such that for any fixed $n > \tilde{n}$, $error_{MCLS} < error_{MC}$ as $Cost_{MCLS} = Cost_{MC} \to \infty$.*

Proof. We first determine the value of $N = N(N',n)$ such that $\mathrm{Cost}_{MCLS} = \mathrm{Cost}_{MC}$:

$$
\mathrm{Cost}_{MCLS} = \mathrm{Cost}_{MC} \iff N = N'\Big(1 + \frac{C_M}{C_f} n\Big).
$$

Consider now the error ratio under the constraint $\mathrm{Cost}_{MCLS} = \mathrm{Cost}_{MC}$, given by

$$
ER := \frac{\mathrm{error}_{MC}}{\mathrm{error}_{MCLS}} = \frac{e_0}{e_n\sqrt{1+\frac{C_M}{C_f}n}},
$$

yielding

$$ER > 1 \iff e_n \sqrt{1 + \frac{C_M}{C_f} n} < e_0.$$

The assumption $e_n = o\left(\frac{1}{\sqrt{n}}\right)$ implies that there exists some $\tilde{n}$ such that $ER > 1$ for all $n > \tilde{n}$. Now, fixing an arbitrary $n > \tilde{n}$ and letting N' and consequently N going to infinity yields the result. □

Remark 6.6. *Note that the quantity ER in the proof of Proposition 6.5 only reflects the error ratio asymptotically for $N, N' \to \infty$. Therefore we restrict the statement of the result to the asymptotic case where $Cost_{MCLS} = Cost_{MC} \to \infty$.*

To show the practical implication of this asymptotic analysis, in Figures 6.2 and 6.3 we examine the convergence of MC and MCLS. We consider the problem of integrating smooth and non-smooth functions for several dimensions d, on the unit cube $[0, 1]^d$ and with respect to the Lebesgue measure. We solve the least-squares problem by means of the CG algorithm. Even though the result of Proposition 6.5 holds for a fixed value of n, in practice the convergence rate can be improved by varying n together with N, as illustrated in [97], where such an adaptive strategy is denoted by MCLSA. For this reason, we show numerical results where we let the cost increase (represented on the x-axis) and for different choices of n (n fixed and n varying)[4].

As expected, the numerical results reflect our analysis presented above. For all dimensions and chosen functions, we achieve an efficiency gain by an appropriate choice of n and N, asymptotically. Note that the erratic convergence with fixed n is a consequence of ill-conditioning; an effect described also in [97]. Namely, when the number of sample points N is not enough, $\mathbf{V}$ tends to be ill-conditioned and the least-squares problem $\min_{\mathbf{c} \in \mathbb{R}^{n+1}} \|\mathbf{V}\mathbf{c} - \mathbf{f}\|_2$ requires many CG iterations, resulting in higher cost than with a larger N. Therefore, the function "N $\mapsto$ Cost(N)" is not necessarily monotonically increasing in N. That is the reason for which the curves in the Figures 6.2 and 6.3, particularly for n fixed, do not necessarily continuously go "from left to right". The single points have been joined according to N increasing.

We now comment further on the condition $e_n = o\left(\frac{1}{\sqrt{n}}\right)$ in Proposition 6.5 in relation to our numerical experiments presented in the Sections 6.3 and 6.4. The integrand f

[4] These figures differ from those in [97] in that the x-axis is the cost rather than the number of sample points N.

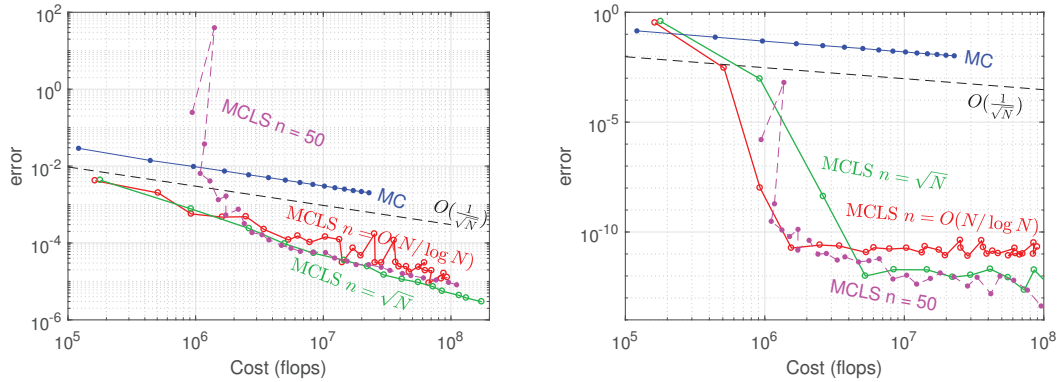

Figure 6.2 – Cost vs Convergence plots for MC and MCLS with varying n: $n = 50$, $n = \sqrt{N}$ and $n = N/\log N$, for $d = 1$. Cost is computed as $2N(n+1)k$, the flop counts in the CG iteration, where k is the number of CG steps required. Left: Non-smooth function $f(x) = |x - \frac{1}{2}|$. Right: analytic function $f(x) = \sin(30x)$.

in the example in Section 6.4 is smooth and the integral is defined on a compact set. Therefore, the above condition is satisfied for this case. For the experiments in option pricing of Section 6.3, the integrands are only continuous and, in some cases, E is not compact. In general, it is difficult to estimate the approximation error e_n in such cases. Intuitively, we cannot expect the above condition to be satisfied. However, the left plots of the Figures 6.2 and 6.3, which are generated for integrands that are only continuous and present similar types of irregularities as the ones of the considered payoff functions, show that MCLS becomes asymptotically more accurate than MC at the same cost. From a practical point of view, this behavior might be expected in our numerical examples in option pricing as well.

6.3 Application to European option pricing

We now apply MCLS to price European options. As mentioned in Chapter 1, the price at time $t = 0$ of a European option with payoff function $f : E \to \mathbb{R}$ and maturing at time T is given by

$$e^{-rT}\mathbb{E}[f(\mathbf{X}_T)] = e^{-rT}\int_E f(\mathbf{x})d\mu(\mathbf{x}), \tag{6.15}$$

where $(\mathbf{X}_t)$ models the asset price over the time interval $[0, T]$, r is a risk-free interest rate and μ denotes the distribution of $\mathbf{X}_T$ whose support is assumed to be E.

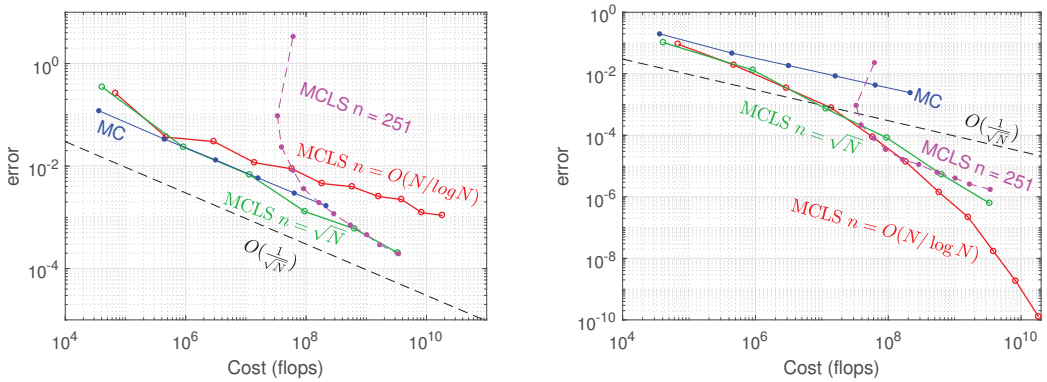

Figure 6.3 – Same as in 6.2, but with $d = 5$. The fixed value $n = 251$ comes from $n = \binom{d+k}{k} - 1$ for degree $k = 5$. Left: non-smooth function $f(x) = \sum_{i=1}^{d} exp(-|x - \frac{1}{2}|)$. Right: analytic function $f(x) = \sin(\sum_{i=1}^{d} x_i)$.

6.3.1 MCLS for European option pricing

We explain how to adapt MCLS to compute European option prices. When applying MCLS for computing (6.15) we observe two potential issues. First, the distribution μ often is not known explicitly. Therefore, we cannot directly perform the sampling part, namely the first step of MCLS, as described in Algorithm 6.1. Second, it is crucial that the basis functions $\{\phi_j\}_{j=0}^{n}$ are easily integrable with respect to μ. Therefore we need to find an appropriate selection of the basis functions.

Concerning the sampling part, if μ is explicitly known, as for example in the Black-Scholes framework (see Section 6.3.2 for two examples), we can just generate sample points according to μ. If μ is not explicitly known, typically the process $(\mathbf{X}_t)$ can still be expressed as the solution of a stochastic differential equation (SDE), as for example in the case of polynomial diffusions, see Chapter 2. In this case, we propose to simulate N paths of $(\mathbf{X}_t)$ by discretizing its governing SDE and collect the realizations of $\mathbf{X}_T$. More details follow below and an example can be found in the Section 6.3.2.

To obtain an appropriate choice of the basis functions $\{\phi_j\}_{j=0}^{n}$ we need $\mathbb{E}[\phi_j(\mathbf{X}_T)]$ to be easy to evaluate. To do so we exploit the structure of the underlying asset model. If $(\mathbf{X}_t)$ belongs to the wide class of affine processes, which is true for a large set of popular models including the Black-Scholes and the Heston model, then the characteristic function of $\mathbf{X}_t$ can be easily computed, as explained e.g. in [35]. Also, if $(\mathbf{X}_t)$ is a Lévy process, the characteristic functions are given by the Lévy-Khintchine formula, see e.g. [106]. In

Algorithm 6.3 Generalized MCLS for European option pricing

Input: Payoff function f, basis functions $\{\phi_j\}_{j=0}^n$, $\phi_0 \equiv 1$, integer $N(> n)$, governing SDE of $\mathbf{X}_t$.

Output: Approximate option price $\hat{I}_{\mu,N} \approx \int_E f(\mathbf{x})d\mu(x)$

1: Simulate N paths of the process $(\mathbf{X}_t)$ from $t = 0$ to $t = T$, and collect the realizations of $\mathbf{X}_T$ in $\mathbf{x}_i$, $i = 1, \ldots, N$.
2: Evaluate $f(\mathbf{x}_i)$ and $\phi_j(\mathbf{x}_i)$, for $i = 1, \ldots, N$ and $j = 1, \ldots, n$.
3: Solve the least-squares problem (6.1) for $\hat{\mathbf{c}} = [\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_n]^T$.
4: Compute $\hat{I}_{\mu,N} = \sum_{j=0}^n \hat{c}_j \int_E \phi_j(\mathbf{x})d\mu(x)$.

these cases, it is therefore natural to choose exponentials as basis functions. If $(\mathbf{X}_t)$ is a polynomial (jump-)diffusion, then its conditional moments are given in closed form via the moment formula (2.7). In this case, polynomials are an excellent choice of basis functions. In our numerical experiments in the Section 6.3.2 we consider some of the polynomial models presented in the Chapter 2 and already used in the rest of this thesis.

To summarize, the main steps of MCLS for option pricing are as follows (if μ is not known explicitly):

1. Simulate N paths of the process $(\mathbf{X}_t)$, from $t = 0$ to $t = T$ (time to maturity), by discretization of the governing SDE.

2. Let $\mathbf{x}_i$ for $i = 1, \ldots, N$ be the realizations of $\mathbf{X}_T$ for each simulated path. Then, we evaluate $f(\mathbf{x}_i)$ and $\phi_j(\mathbf{x}_i)$, for $i = 1, \ldots, N$ and $j = 1, \ldots, n$.

3. Solve the least-squares problem (6.1) to obtain the approximation of f. The solver can be chosen according to the scheme represented in Figure 6.1.

4. Finally, the option price is approximated via (we omit the discounting factor)

$$\mathbb{E}[f(\mathbf{X}_T)] = \int_E f(\mathbf{x})d\mu(\mathbf{x}) \approx \hat{I}_{\mu,N} := \sum_{j=0}^n \hat{c}_j \int_E \phi_j(\mathbf{x})d\mu(\mathbf{x}) = \sum_{j=0}^n \hat{c}_j \mathbb{E}[\phi_j(\mathbf{X}_T)].$$

   Note that we select the basis functions in such a way that the quantities $\mathbb{E}[\phi_j(\mathbf{X}_T)]$ can be easily evaluated. In particular, no Monte Carlo simulation is required.

Algorithm 6.3 summarizes this procedure.

In the case that μ is explicitly known, the error resulting from MCLS is analysed in

Proposition 6.2 and Proposition 6.3. In case we discretize the governing SDE of $(\mathbf{X}_t)$, we introduce a second source of error, which we address in the following.

Assume that $(\mathbf{X}_t)$ is the solution of an SDE of the form

$$\begin{aligned} d\mathbf{X}_t &= b(\mathbf{X}_t)dt + \Sigma(\mathbf{X}_t)d\mathbf{W}_t, \quad 0 < t \leq T \\ \mathbf{X}_0 &= \mathbf{x}_0, \end{aligned} \tag{6.16}$$

where $(\mathbf{W}_t)$ denotes a d-dimensional Brownian motion, $b : \mathbb{R}^d \mapsto \mathbb{R}^d$, $\Sigma : \mathbb{R}^d \mapsto \mathbb{R}^{d\times d}$, and $\mathbf{x}_0 \in \mathbb{R}^d$. An approximation of the solution $(\mathbf{X}_t)$ of (6.16) can be computed via a uniform Euler-Maruyama scheme, defined in the following.

Definition 6.7. *Consider an equidistant partition of $[0, T]$ in N_s intervals, i.e.*

$$\Delta t = T/N_s, \quad t_i = i\Delta t \quad \textit{for } i = 0, \cdots, N_s,$$

together with

$$\Delta\widetilde{\mathbf{W}}_i = \mathbf{W}_{t_{i+1}} - \mathbf{W}_{t_i} \quad \textit{for } i = 0, \cdots, N_s.$$

Then, the Euler-Maruyama discretization scheme of (6.16) *is given by*

$$\begin{aligned} \bar{\mathbf{X}}_{i+1} &= \bar{\mathbf{X}}_i + b(\bar{\mathbf{X}}_i)\Delta t + \Sigma(\bar{\mathbf{X}}_i)\Delta\widetilde{\mathbf{W}}_i, \quad \textit{for } i = 0, \cdots, N_s - 1, \\ \bar{\mathbf{X}}_0 &= \mathbf{x}_0, \end{aligned} \tag{6.17}$$

and the Euler-Maruyama approximation of $\mathbf{X}_T$ is given by $\bar{\mathbf{X}}_{N_s}$.

Assume that we sample N independent copies of $\bar{\mathbf{X}}_{N_s}$ (first step of Algorithm 6.3) and we apply MCLS to approximate (6.15). Then the error naturally splits into two components as

$$|\mathbb{E}[f(\mathbf{X}_T)] - \bar{I}_{\mu,N}| \leq |\mathbb{E}[f(\mathbf{X}_T)] - \mathbb{E}[f(\bar{\mathbf{X}}_{N_s})]| + |\mathbb{E}[f(\bar{\mathbf{X}}_{N_s})] - \bar{I}_{\mu,N}|.$$

The second summand can then be approximated as in (6.9). We collect the result in the following proposition.

Proposition 6.8. *Let $\bar{I}_{\mu,N}$ be the MCLS estimator obtained by sampling according to the Euler-Maruyama scheme as in Definition 6.7. Then, the MCLS error asymptotically (for*

n fixed and $N \to \infty$) satisfies

$$|\mathbb{E}[f(\mathbf{X}_T)] - \bar{I}_{\mu,N}| \lessapprox |\mathbb{E}[f(\mathbf{X}_T)] - \mathbb{E}[f(\bar{\mathbf{X}}_{N_s})]| + \frac{\min_{\boldsymbol{c}\in\mathbb{R}^{n+1}} \|f - \sum_{j=0}^{n} c_j\phi_j\|_{\bar{\mu}}}{\sqrt{N}}, \quad (6.18)$$

where $\bar{\mu}$ is the distribution of $\bar{\mathbf{X}}_{N_s}$.

The first term in the right-hand-side of (6.18) is usually referred to as *time-discretization error*, while the second summand denotes the so-called *statistical error*. The time-discretization error and, more generally, the Euler-Maruyama scheme together with its properties, are well studied in the literature, see e.g. [81]. Depending on the regularity properties of f, b and Σ, one can conclude, for example, that the time-discretization error is bounded from above by $C|\Delta t|$, for a constant $C > 0$. In this case, we say that the Euler-Maruyama scheme converges *weakly* with order 1. Finally, note that the statistical error can be further approximated as in (6.10) using

$$\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|f - \sum_{j=0}^{n} c_j\phi_j\|_{\bar{\mu}} \approx \frac{1}{N-n-1}\sum_{i=1}^{N}(f(\mathbf{x}_i) - p(\mathbf{x}_i))^2 = \frac{1}{N-n-1}\|\mathbf{V}\hat{\mathbf{c}} - \mathbf{f}\|_2^2,$$

where the $\mathbf{x}_i$'s are sampled according to $\bar{\mu}$.

6.3.2 Numerical examples in option pricing

Next, we apply MCLS to numerically compute European option prices (6.15) for several types of payoff functions f and in different polynomial diffusion models. All algorithms have been implemented in MATLAB version 2017a and run on a standard laptop (Intel Core i7, 2 cores, 256kB/4MB L2/L3 cache).

In all of our numerical experiments the solver for numerical solution of the least-squares problem (6.1) is chosen according to the scheme in Figure 6.1. The choice of the examples leads us to test all of the three choices in the scheme. For the univariate pricing examples in Heston's and the Jacobi model the CG algorithm is appropriate. Later on, when basket options of medium dimensionality in the multivariate Black-Scholes model are considered, a QR based method is employed, because the condition number of $\mathbf{V}$ was usually larger than $\mathcal{O}(1)$. In these both cases we directly sample from the distribution of the underlying random variable $\mathbf{X}_T$, where in the univariate case we solve an SDE. Finally, we consider pricing a rainbow option in a high-dimensional multivariate Black-Scholes

model, where the randomized extended Kaczmarz algorithm combined with the optimal sampling strategy yields a good performance.

Call option in stochastic volatility models

We consider the Heston model in its log-asset price formulation as presented in Chapter 2. We recall that Corollary 2.6 states that the moments of X_T are given as in (2.14).

In the following we apply MCLS in the Heston model in order to price single-asset European call options with payoff function given by

$$f(x) = (e^x - e^k)^+,$$

for a log-strike value k. We compare MC and MCLS to the Fourier pricing method introduced in [23] and reviewed in Chapter 1.

In this experiment we use an ONB (with respect to the corresponding L^2_μ space, where μ is the distribution of X_T) of polynomials as basis functions $\{\phi_j\}_{j=0}^n$. Conveniently the ONB can be obtained by applying the Gram-Schmidt orthogonalization process to the monomial basis. Note that, even if the distribution μ is not known explicitly, we still can apply the Gram-Schmidt orthogonalization procedure since the corresponding scalar product and the induced norm can be computed via the moment formula (2.14).

Since the distribution of X_T is not known explicitly, we apply the Euler-Maruyama scheme as defined in (6.17) and obtain

$$\begin{aligned} V_0 &= v_0, \\ X_0 &= x_0, \\ V_{t_i} &= V_{t_{i-1}} + \kappa(\theta - V_{t_{i-1}})\Delta t + \sigma\sqrt{V_{t_{i-1}}}\sqrt{\Delta t}Z_i^1, \\ X_{t_i} &= X_{t_{i-1}} + (r - V_{t_{i-1}}/2)\Delta t + \rho\sqrt{V_{t_{i-1}}}\sqrt{\Delta t}Z_i^1 + \sqrt{V_{t_{i-1}}}\sqrt{1-\rho^2 V_{t_{i-1}}}\sqrt{\Delta t}Z_i^2, \end{aligned} \tag{6.19}$$

for all $i = 1, \cdots, N_s$ and where Z_i^1 and Z_i^2 are independent standard normal distributed random variables. For the following experiments we consider the model parameters

$$\sigma = 0.15,\ \ v_0 = 0.04,\ \ x_0 = 0,\ \ \kappa = 0.5,\ \ \theta = 0.01,\ \ \rho = -0.5,\ \ r = 0.01.$$

For this choice of parameters, the arguments of the square roots in (6.19) are always positive. The Euler-Maruyama scheme is therefore well-defined. However, for a different choice of parameters this might not be the case. If this happens, the scheme can be modified by taking the absolute value or the positive part of the arguments of the square roots. Such a modification is discussed, e.g., in [80]. The same remark holds for the forthcoming numerical examples.

First, we apply MCLS to an in-the-money (ITM) call option, with payoff parameters

$$k = -0.1, \quad T = 1/12,$$

and we use $N_s = 100$ time steps for the discretization of the SDE. We use an ONB consisting of polynomials of maximal degrees 0 (standard MC), $1, 3$ and 5 and we obtain the results shown in Figure 6.4[5]. In particular, we plot the absolute error of the prices and the width of the obtained 95% confidence interval computed as in (6.10) and (6.11), against the number of simulated points N.

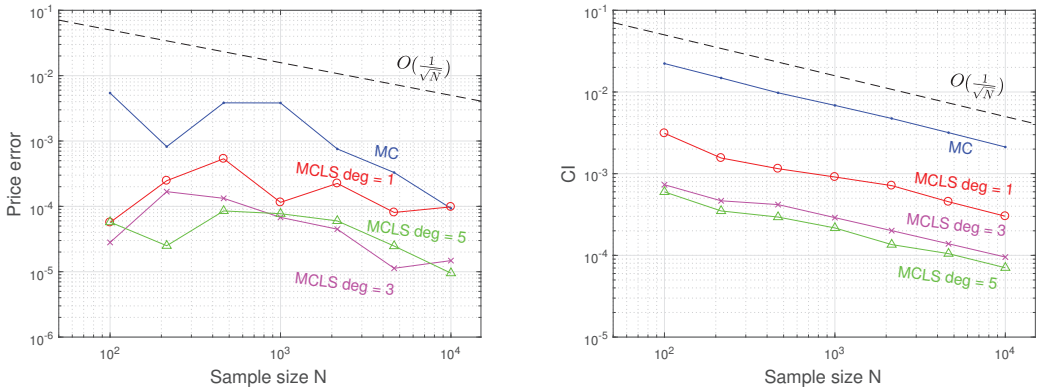

Figure 6.4 – MCLS for ITM call option in Heston model for different polynomial degrees. Left: Absolute price error. Right: Width of 95% *confidence interval.*

Second, we apply again MCLS but this time to an at-the-money (ATM) call option with parameters

$$k = 0, \quad T = 1/12,$$

[5]Note that all the left-figures of this chapter are the outcome of a random choice of sample points. Different samples would give different figures. In these experiments, we haven't chosen any particular sample. The right-figures showing the confidence intervals, instead, are more robust in the choice of the sample points and might be used for a better assessment of the method, as also done in [97].

and to an out-of-the-money (OTM) call option with parameters

$$k = 0.1, \quad T = 1/12.$$

The results are shown in Figure 6.5 and in Figure 6.6, respectively.

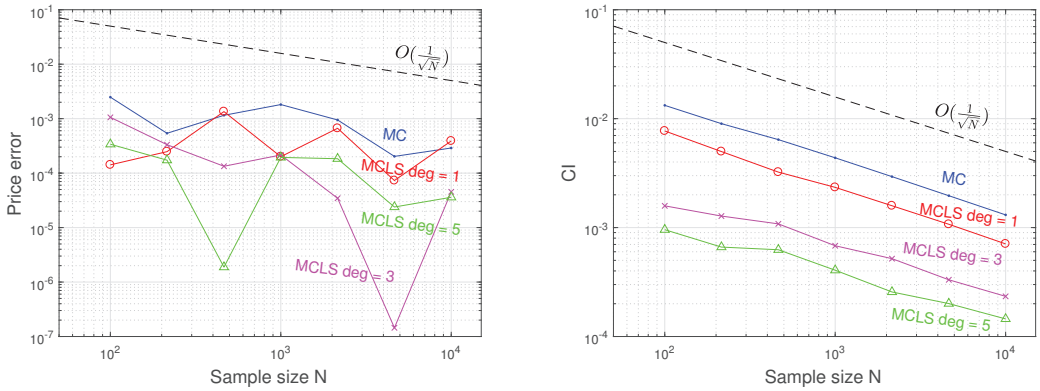

Figure 6.5 – MCLS for ATM call option in Heston model for different polynomial degrees. Left: Absolute price error. Right: Width of 95% *confidence interval.*

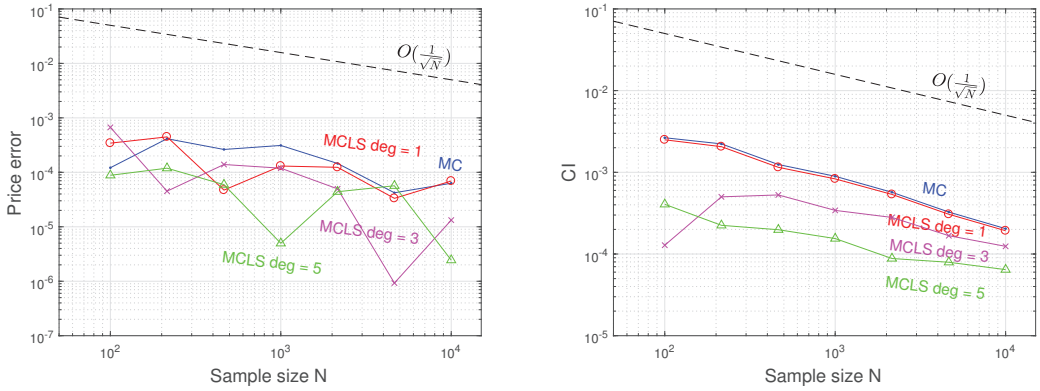

Figure 6.6 – MCLS for OTM call option in Heston model for different polynomial degrees. Left: Absolute price error. Right: Width of 95% *confidence interval.*

In this setting, for all different choices of payoff parameters, we show in Table 6.2 the implied volatility absolute errors (in percentage) for the MC and MCLS prices computed with a basis of polynomials of maximal degree 5. The implied volatility error is measured against the implied volatility of the reference method.

Before commenting on the numerical results, we apply MCLS to a second stochastic volatility model, the Jacobi model presented in Chapter 2, Section 2.2. The moments can

Implied vol absolute errors

	$k=-0.1$		$k=0$		$k=0.1$	
N	MC	MCLS	MC	MCLS	MC	MCLS
100	–	0.21	2.16	0.29	0.50	0.37
215	4.35	0.09	0.47	0.15	1.56	0.49
464	9.16	0.31	1.00	0.00	1.03	0.26
1000	9.13	0.28	1.58	0.17	1.21	0.02
2154	2.44	0.22	0.82	0.16	0.59	0.19
4642	1.15	0.09	0.18	0.02	0.18	0.24
10000	0.34	0.04	0.25	0.03	0.28	0.01

Table 6.2 – Implied volatility errors (in %) for MC and MCLS with basis of polynomials of maximal degree 5 *in the Heston model, for different sizes N of the sample set.*

be computed again as in Corollary 2.6 where G_n corresponds now to the Jacobi model and can be computed as explained in Section 2.2. For the numerical experiments we consider the set of model parameters

$$\sigma = 0.15,\ \ v_0 = 0.04,\ \ x_0 = 0,\ \ \kappa = 0.5,\ \ \theta = 0.04,$$
$$v_{\min} = 10^{-4},\ \ v_{\max} = 0.08,\ \ \rho = -0.5,\ \ r = 0.01.$$

We again consider single-asset European call options with payoff parameters

$$k = \{-0.1, 0, 0.1\}, \quad T = 1/12.$$

As reference pricing method we choose the polynomial expansion technique introduced in [4] and reviewed in Chapter 2, Section 2.3. Also, note that the same reference pricing method has been used in Chapter 3, Section 3.1.4, where we give more details about the method tailored to the Jacobi model. Here, we truncate the polynomial expansion of the price after 50 terms.

We simulate the whole path of (X_t) from 0 to T in order to get the sample points x_i,

$i = 1, \cdots, n$. The discretization scheme of the SDE is given by

$$
\begin{aligned}
V_0 &= v_0, \\
X_0 &= x_0, \\
V_{t_i} &= V_{t_{i-1}} + \kappa(\theta - V_{t_{i-1}})\Delta t + \sigma\sqrt{Q(V_{t_{i-1}})}\sqrt{\Delta t}Z_i^1, \\
X_{t_i} &= X_{t_{i-1}} + (r - V_{t_{i-1}}/2)\Delta t + \rho\sqrt{Q(V_{t_{i-1}})}\sqrt{\Delta t}Z_i^1 + \sqrt{V_{t_{i-1}} - \rho^2 Q(V_{t_{i-1}})}\sqrt{\Delta t}Z_i^2
\end{aligned}
$$

for all $i = 1, \cdots, N_s$, where Z_i^1 and Z_i^2 are independent standard normal distributed random variables and the rest of the parameters are as specified in the example for the Heston model.

We use again an ONB consisting of polynomials of maximal degrees 0 (standard MC), 1, 3 and 5 and we obtain the results shown in Figures 6.7, 6.8 and 6.9, for ITM, ATM and OTM call options, respectively. Lastly, we show in Table 6.3 the implied volatility absolute percentage errors for the MC and MCLS prices computed with a basis of polynomials with maximal degree 5.

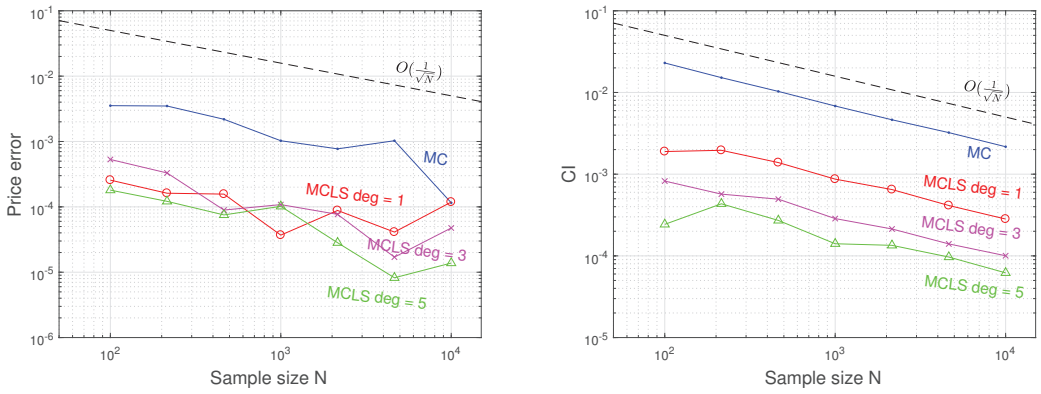

Figure 6.7 – MCLS for ITM call option in Jacobi model for different polynomial degrees. Left: Absolute price error. Right: Width of 95% confidence interval.

We can observe that MCLS strongly outperforms the standard MC in terms of price errors, confidence interval width and implied volatility errors, for every type of moneyness, for almost all values of N, and in both chosen stochastic volatility models. MCLS is therefore very effective when applied to price single-asset options.

The last remark concerns the condition number of the Vandermonde matrix $\mathbf{V}$. Thanks to the choice of the ONB and to the fact that $N \gg n$, in both models, its condition

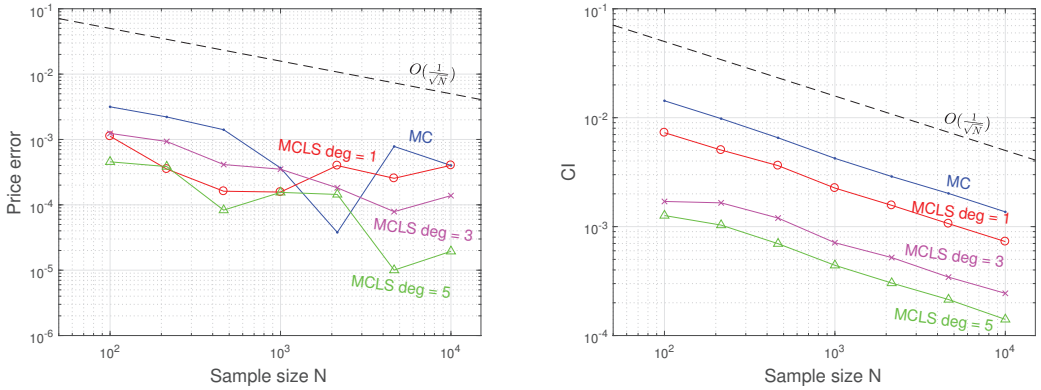

Figure 6.8 – MCLS for ATM call option in Jacobi model for different polynomial degrees. Left: Absolute price error. Right: Width of 95% *confidence interval.*

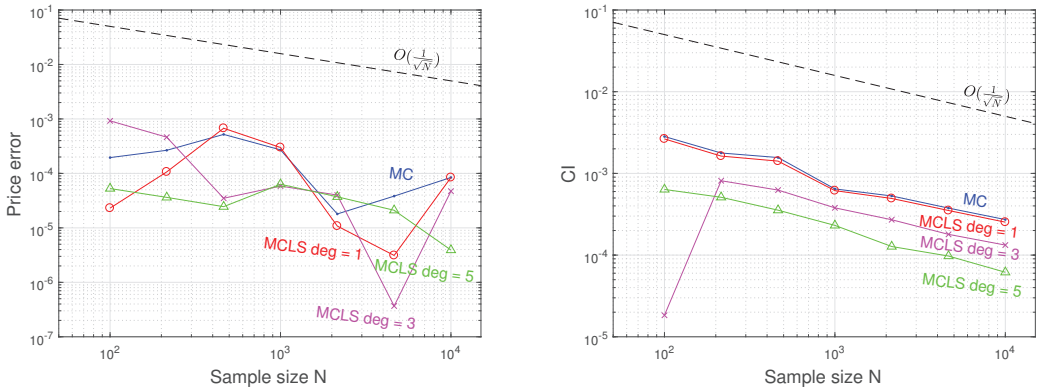

Figure 6.9 – MCLS for OTM call option in Jacobi model for different polynomial degrees. Left: Absolute price error. Right: Width of 95% *confidence interval.*

number is at most of order 10. Therefore, the CG algorithm has been selected. As another consequence of the low condition number we did not implement optimal sampling[6].

Basket options in Black-Scholes models - medium size problems

In this section we address multi-dimensional option pricing problems of medium size in the Black-Scholes model, meaning with number of assets $d \leq 10$ and $N \leq 10^5$. The risk neutral dynamics of the asset prices $(S_t^1, \ldots, S_t^d)$ and other relevant properties of the model are specified in the Chapter 2, Section 2.2. In particular the moments can be

[6]Note that it is not clear how to efficiently implement the optimal sampling strategy in connection with an Euler-Maruyama scheme. This remains an open problem.

Implied vol absolute errors

	$k=-0.1$		$k=0$		$k=0.1$	
N	MC	MCLS	MC	MCLS	MC	MCLS
100	8.75	0.67	2.75	0.40	0.72	0.20
215	8.67	0.46	1.92	0.33	0.96	0.14
464	–	0.30	1.23	0.07	1.77	0.10
1000	3.27	0.39	0.32	0.13	1.16	0.24
2154	2.55	0.11	0.03	0.13	0.07	0.14
4642	3.26	0.03	0.68	0.01	0.15	0.08
10000	0.47	0.05	0.35	0.02	0.32	0.02

Table 6.3 – Implied volatility errors (in %) for MC and MCLS with basis of polynomials with maximal degree 5 *in the Jacobi model, for different sizes* N *of the sample set.*

computed as in Corollary 2.5 and the corresponding matrix G_n is diagonal and given as in the Corollary 2.4.

For the following numerical experiments we consider basket options with payoff function

$$f(s_1, \cdots, s_d) = \left(\sum_{i=1}^{d} w_i s_i - K \right)^+ \tag{6.20}$$

for different moneyness with payoff parameters

$$K = \{0.9, 1, 1.1\}, \quad T = 1, \quad w_i = \frac{1}{d} \ \forall i.$$

Model parameters are chosen to be

$$S_0^i = 1 \ \forall i, \quad \sigma_i = \text{rand}(0, 0.5) \ \forall i, \quad \Sigma = R_d, \quad r = 0.01,$$

where each σ_i is a randomly chosen volatility parameter in $[0, 0.5]$, and R_d denotes a random correlation matrix of size $d \times d$. In this example we consider dimensions $d = 5$ and $d = 10$.

We compare MCLS to a reference price computed via a standard Monte Carlo algorithm with 10^6 simulations. We plot again the absolute price errors and the width of the 95% confidence intervals (computed as in (6.10) and (6.11)) for different chosen polynomial degrees (0 (MC), maximally 1 and maximally 3). To be more precise, we used the monomial basis as functions $\{\phi_j\}_{j=0}^n$. Note that the distribution of the prices $(S_t^1, \cdots, S_t^d)$

is known to be the geometric Brownian distribution so that there is no need to simulate the whole path but only the price at final time T.

The results are shown in Figures 6.10, 6.11 and 6.12. In the legend the represented number indicates again the maximal total degree of the basis monomials. For instance, if $d = 2$ and the maximal total degree is $\deg = 3$, this means that the basis functions ϕ_j are chosen to be $\{1, s_1, s_2, s_1^2, s_1 s_2, s_2^2, s_1^3, s_1^2 s_2, s_1 s_2^2, s_2^3\}$.

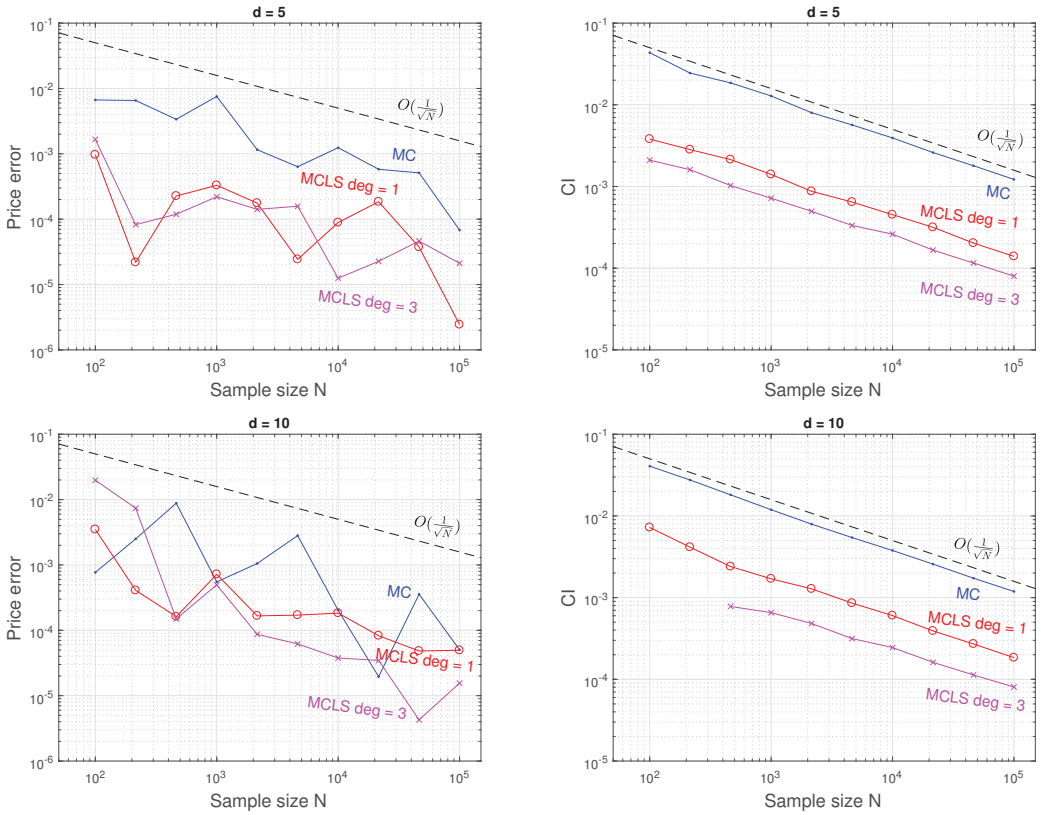

Figure 6.10 – MCLS for basket options (with $K = 0.9$) in Black-Scholes model for different dimensions and polynomial degrees. Left: absolute price errors with respect to a reference price computed with 10^6 simulations. Right: Width of 95% confidence interval.

We observe that also in these multidimensional examples MCLS strongly outperforms the standard MC in terms of absolute price errors and width of the confidence intervals. Due to the use of the multivariate monomials as basis functions, the condition number of $\mathbf{V}$ is relatively high, reaching values up to order 10^5. However, the QR based algorithm chosen according to the selection scheme 6.1 for the numerical solution of the least-squares problem (6.1) still yields accurate results. The Vandermonde matrix $\mathbf{V}$ is here still

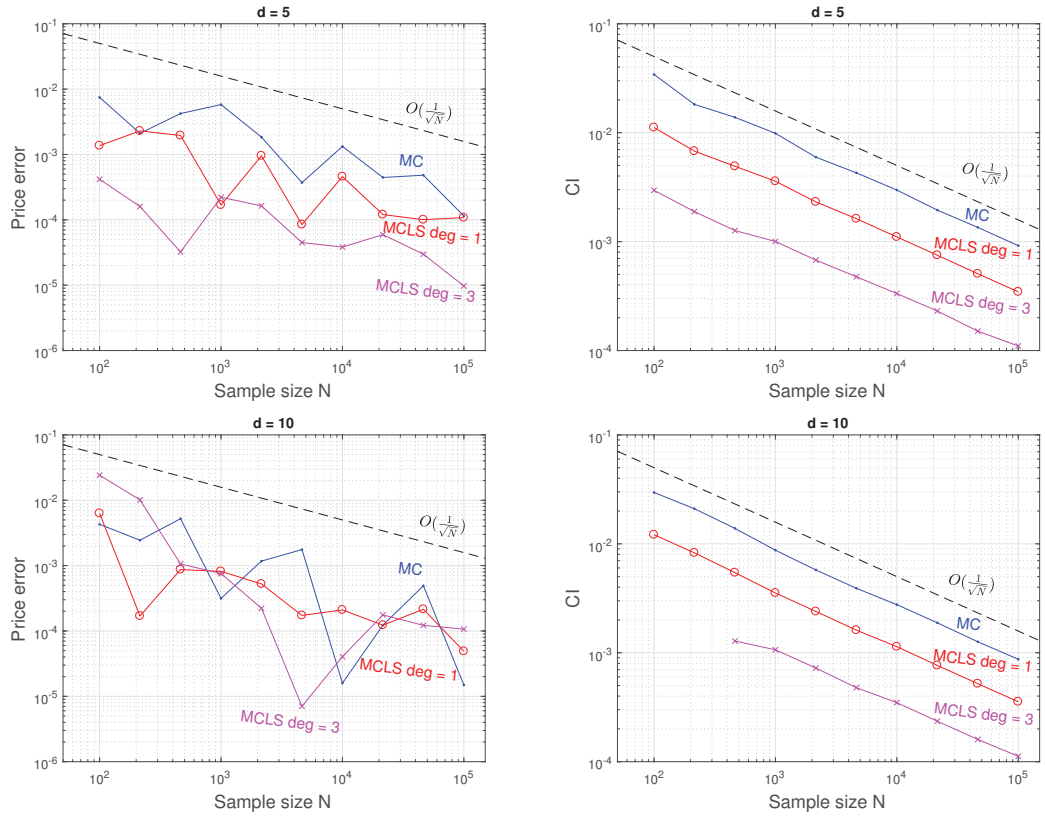

Figure 6.11 – MCLS for basket options (with $K = 1$) in Black-Scholes model for different dimensions and polynomial degrees. Left: absolute price errors with respect to a reference price computed with 10^6 simulations. Right: Width of 95% confidence interval.

storable, being of size at most $10^5 \times 286$. In the next section we treat problems of higher dimensionality leading to a Vandermonde matrix of bigger size. There, its storage is not feasible any more and neither CG nor QR based solvers can be used.

Basket options in Black-Scholes models - big size problems

In the multivariate Black-Scholes model we now consider rainbow options with payoff function

$$f(s_1, \cdots, s_d) = (K - \min(s_1, \cdots, s_d))^+,$$

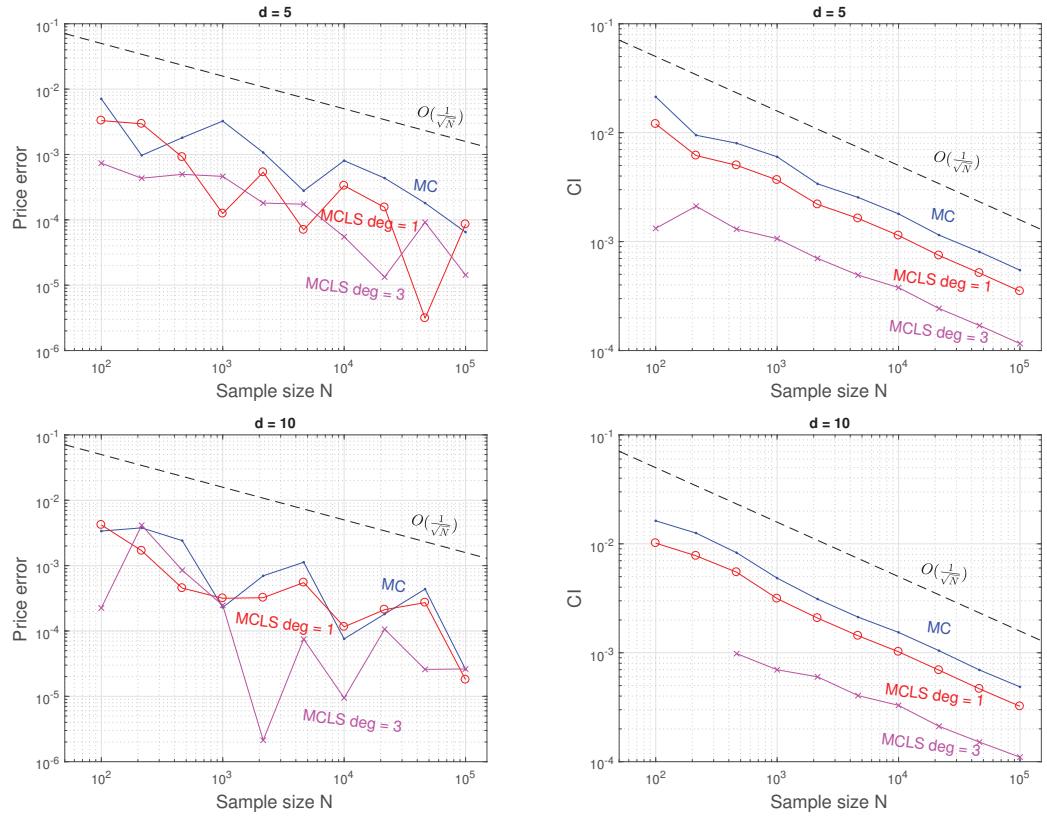

Figure 6.12 – MCLS for basket options (with $K = 1.1$) in Black-Scholes model for different dimensions and polynomial degrees. Left: absolute price errors with respect to a reference price computed with 10^6 simulations. Right: Width of 95% confidence interval.

so that we apply MCLS in order to compute the quantity

$$\mathrm{e}^{-rT}\mathbb{E}[(K - \min(S_T^1, \cdots, S_T^d))^+] = \mathrm{e}^{-rT} \int_{\mathbb{R}_+^d} (K - \min(s_1, \cdots, s_d))^+ d\mu(s_1, \cdots, s_d),$$

where μ is the distribution of $(S_T^1, \cdots, S_T^d)$. In contrast to the payoff (6.20) which presents one type of irregularity that derives from taking the positive part $(\cdot)^+$, this payoff function presents two types of irregularities: one again due to $(\cdot)^+$, and the second one deriving from the $\min(\cdot)$ function. This example is therefore more challenging.

As similarly done in [97], we rewrite the option price with respect to the Lebesgue measure

$$\mathrm{e}^{-rT} \int_{[0,1]^d} \left(K - \min_{i=1,\ldots,d} \left(S_0^i \exp\left((r - \frac{\sigma_i^2}{2})t + \sigma_i \sqrt{T} \mathbf{L} \Phi^{-1}(\mathbf{x}) \right) \right) \right)^+ d\mathbf{x},$$

where $\mathbf{L}$ is the Cholesky decomposition of the correlation matrix and Φ^{-1} is the inverse map of the cumulative distribution function of the multivariate standard normal distribution.

The model and payoff parameters are chosen to be

$$S_0^i = 1, \quad K = 1, \quad \sigma_i = 0.2 \ \forall i, \quad \Sigma = I_d \quad T = 1, \quad r = 0.01,$$

so that we consider a basket option of uncorrelated assets.

We apply MCLS for $d = \{5, 10, 20\}$ using different total degrees for the approximating polynomial space and we compare it to a reference price computed using the standard MC algorithm with 10^7 simulations. Also, we consider different numbers of simulations that go up to 10^6. We choose a basis of tensorized Legendre polynomials, that form an ONB with respect to the Lebesgue measure on the unit cube $[0, 1]^d$ and we perform the sampling step of MCLS (step 1) according to the optimal distribution as introduced in [26] and reviewed in Section 6.1.1. The solver for the least-squares problem is chosen according to the scheme shown in Figure 6.1, where we assume that the Vandermonde matrix $\mathbf{V}$ can be stored whenever the number of entries is less than 10^8. This implies that also, for example, for the case $d = 5$ with polynomial degree 5 and 10^6 simulations $\mathbf{V}$ can not be stored. Indeed, for $d = 5$, deg $= 5$ and $N = 10^6$ the matrix $\mathbf{V}$ has $2.52 \cdot 10^8$ entries. For all of these cases, we therefore solve the least-squares problem by applying the randomized extended Kaczmarz algorithm.

In Figure 6.13 we plot the obtained price absolute errors and the width of the 95% confidence intervals for all considered problems. We notice that MCLS outperforms again MC in terms of confidence interval width and price errors, as observed for medium dimensions. The choice of the optimal sampling strategy combined with the ONB allowed us to obtain a well conditioned matrix $\mathbf{V}$, according to the theory presented in the previous sections.

These examples and the obtained numerical results show therefore that our extension of MCLS is effective and allows us to efficiently price single and multi-asset European options. In the next section we test our extended MCLS in a slightly different setting where the integrating function is smooth.

6.4 Application to high-dimensional integration

In this section we apply the extended MCLS algorithm to compute the definite integral

$$\int_{[0,1]^d} \sin\left(\sum_{j=1}^{d} x_j\right) d\mathbf{x}. \tag{6.21}$$

The same integration problem has been considered in [97] where the author managed to apply MCLS to compute (6.21) for dimension at most $d = 6$ and with at most $N = 10^5$ simulations. Our goal is to show that, thanks to our extension, we can increase the considered dimension d and the number of simulations N.

We apply MCLS for $d = 10$ and $d = 30$ using a basis of tensorized Legendre polynomials of total degree 5 and 4, respectively. We compare it to the reference result which for $d = 2 \pmod 4$ is explicitly given by

$$\int_{[0,1]^d} \sin\left(\sum_{j=1}^{d} x_j\right) d\mathbf{x} = \sum_{j=0}^{d} (-1)^{j+1} \binom{d}{j} \sin(j).$$

Also, we consider different sample sizes that go up to 10^7. We perform the sampling step of MCLS (step 1) according to the optimal distribution as introduced in [26] and reviewed in Section 6.1.1. The choice of the solver for the least-squares problem is again taken according to the scheme in Figure 6.1 and we assume that the Vandermonde matrix V can be stored whenever the number of entries is less than 10^8.

The obtained results are shown in the Figure 6.14. Again, we have plotted the obtained absolute error computed with respect to the reference result (left) and the width of the 95% confidence interval (right). First, we note that MCLS performs much better than the standard MC, as in the previously shown examples. Furthermore, the obtained results are much better than the ones obtained in the previous section, see Figure 6.13. This is due to the fact that the integrand is now smooth, while in the multi-asset option example it was only continuous. Indeed, the function approximation error $\min_{\mathbf{c}\in\mathbb{R}^{n+1}} \|\sqrt{w}(f - \sum_{j=0}^{n} c_j \phi_j)\|_\mu$ is expected to be much smaller in this case, since polynomials provide a more suitable approximating space for smooth functions than for irregular functions. According to Proposition 6.2, this results in a stronger variance reduction and in a better approximation of the integral. Finally, the very good numerical

results show that our extension is effective since we managed to consider problems of size much larger than in [97].

6.5 Conclusion

We have presented a numerical technique to price single and multi-asset European options. The starting point was the algorithm (MCLS) for multidimensional integration developed in [97]. After extending MCLS to numerically evaluate integrals with respect to an arbitrary probability measure, we have proposed to combine it with the optimal sampling strategy (Section 6.1.1) introduced in [26] and with the randomized extended Kaczmarz algorithm [124] (Section 6.1.2). The optimal sampling strategy allows us to obtain a well conditioned least-squares problem that, in a second moment, can be solved by applying the REK algorithm. This combination allows us to treat problems of big size since REK does not need the storage of the full matrix $\mathbf{V}$, which was the main bottleneck in using classical algorithms for the numerical solution of least-squares problems. Afterwards, in Section 6.2 we have presented a convergence and a cost analysis. Here, we have shown that MCLS asymptotically outperforms the standard Monte Carlo method as the cost goes to infinity, provided that the integrand satisfies certain regularity conditions.

In the second part of the chapter, Section 6.3, we have applied the new method to the problem of European option pricing. First, we have adapted our generalization to compute single and multi-asset option prices, where we have proposed to modify the sampling step of MCLS by discretizing the governing SDE of the underlying price process, whenever needed. The modification of the first step introduces a new source of error, which has been analyzed in Proposition 6.8. In Section 6.3.2 we have applied the algorithm to price single and multi-asset European options in the Heston model, in the Jacobi model and in the multidimensional Black-Scholes model. Here, we have exploited the fact the these models belong to the class of polynomial diffusions and the moments can be computed in closed form. For these examples, MCLS strongly outperformed the standard MC in terms of implied volatility, see Table 6.2 and Table 6.3, and in terms of option price errors and confidence interval width, see for instance Figures 6.4-6.9 and Figure 6.13. In particular the choice of the Kaczmarz algorithm combined with the optimal sampling strategy allowed us to consider problems of big size, where we treated options based on 20 assets and 10^6 simulations. Finally, in the Section 6.4 we have tested our extension of MCLS in the computation of a multidimensional integral of a smooth function, which

had already been considered in [97]. There, the author managed to consider a maximal dimension $d = 6$ and $N = 10^5$. Thanks to the application of the REK algorithm we managed to reach $d = 30$ and to consider N up to 10^7, showing the effectiveness of our extended approach.

To extend the approach further to even higher dimensions, computational bottlenecks arising are to be addressed. Solving the storage issue in the least-squares problem with Kaczmarz leaves us with a high number of function calls. We do not need to store the full Vandermode matrix, but instead rows and columns are required many times during the iteration. This leads to a high computational cost. One can reduce this cost by 1) reducing the number of function calls and by 2) making the function calls more efficient. To achieve 1), one can for instance store the rows and columns of the Vandermonde matrix which are called with highest probability. To achieve 2) one can exploit further insight of the functions, for instance using a low-rank approximation [55] or functional analogues of tensor decomposition approximation [53].

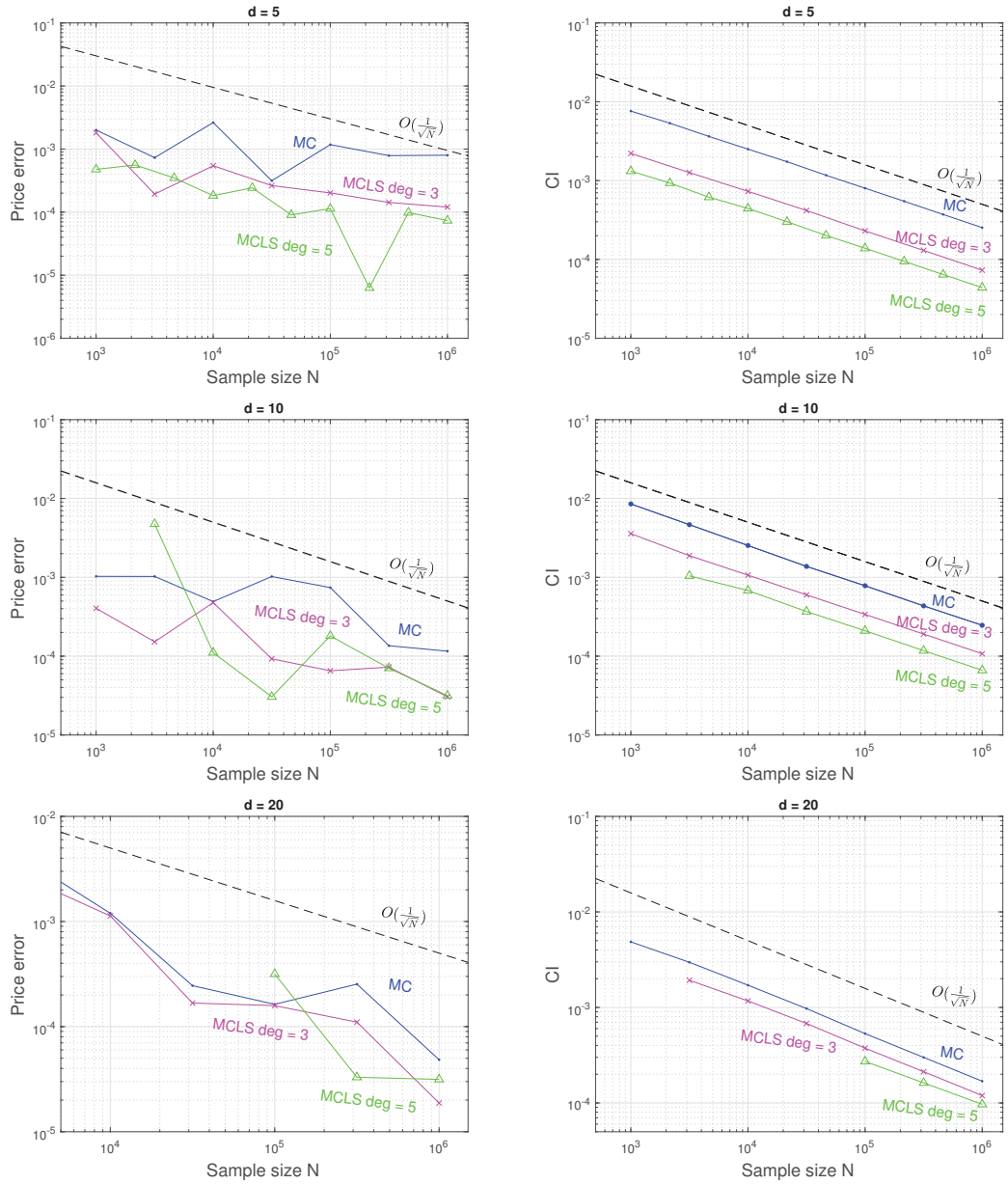

Figure 6.13 – MCLS for rainbow options in Black-Scholes model for different dimensions and polynomial degrees. Left: absolute price errors with respect to a reference price computed with 10^7 *simulations. Right: width of the* 95% *confidence interval.*

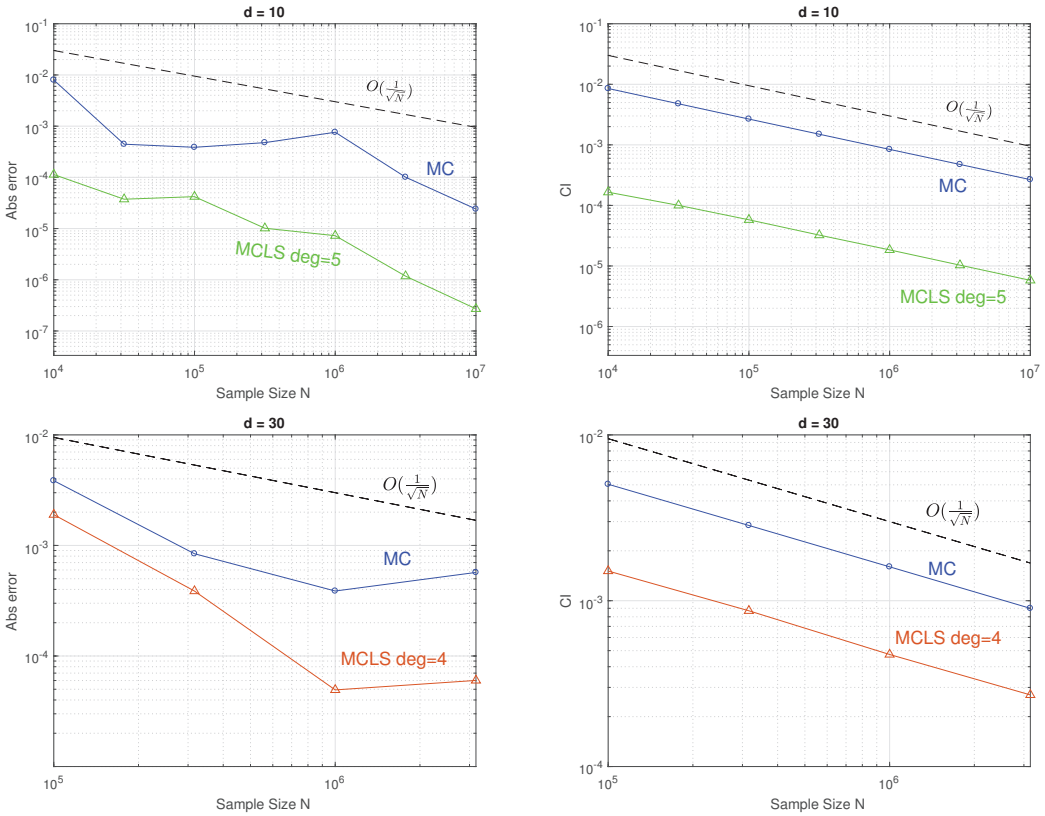

Figure 6.14 – MCLS for integrating the function $\sin(\sum_{j=1}^{d} x_j)$ *for* $d = \{10, 30\}$ *and different polynomial degrees. Left: absolute errors with respect to a reference result given in closed form. Right: width of the* 95% *confidence interval.*

7 Conclusion

In this thesis we have discussed the development of numerical methods for option pricing. In particular, we have considered the class of polynomial models, which have been reviewed in Chapter 2, and we have focused on treating important challenges that arise when developing option pricing techniques, discussed in Chapter 1. For instance, treating the high dimensionality arising in option pricing problems and reducing the algorithmic complexity are important points that we have addressed.

In Chapter 3, we have exploited the availability of all the moments of polynomial jump-diffusions to introduce a pricing technique based on the computation of polynomial bounds for option prices. For both the European and the American case, we have explained how to compute the bounds by solving sequences of optimization problems using two different numerical approaches: a technique based on semidefinite programming, and an algorithm based on the cutting plane procedure. For the one-dimensional European case, we have obtained new convergence results where we have considered a general class of non-piecewise polynomial payoff functions. Numerical experiments have shown that the method yields sharp bounds for different payoff profiles and for different models. Finally, we have introduced a black box algorithm for European option pricing (Algorithm 3.2), able to take model and payoff parameters as input, and to return the option price.

In Chapter 4, we have developed `incexpm`, a new efficient algorithm that computes sequences of nested block triangular matrix exponentials. Since the sequence $G_0, G_1, \ldots$ of the matrix representations of the generator $\mathcal{G}$ restricted to $\mathrm{Pol}_0(E), \mathrm{Pol}_1(E), \ldots$ exhibits a nested block triangular structure, our algorithm can be used to reduce the complexity of

the pricing procedures that require an incremental computation of the moment sequence. An example is the aforementioned Algorithm 3.2. The complexity is reduced by combining `incexpm` with the moment formula (2.7). At each step k of the incremental procedure, `incexpm` reduces the complexity of computing $\exp(G_k)$ from $\mathcal{O}(k^3b^3)$ to $\mathcal{O}(k^2b^3)$, where b is the size of the diagonal blocks. Moreover, we have developed an adaptive scaling procedure which allows us to avoid inaccurate results and, in the context of polynomial models, we have derived estimation techniques to choose the scaling parameter. Finally, the numerical experiments have confirmed that using `incexpm` instead of MATLAB's `expm` reduces the complexity of some pricing techniques while maintaining the same accuracy.

In Chapter 5, we have developed a complexity reduction technique for parametric option pricing based on the tensorized Chebyshev interpolation. In particular, we have extended the approach proposed in [46] to treat high-dimensional parameter spaces. The core idea is to exploit the low-rank structure of the tensors involved in the interpolation task by expressing the whole procedure in the TT format. This allows us to reduce the storage complexity from $\mathcal{O}(n^d)$ to $\mathcal{O}(dnr^2)$, where n is the interpolation order, d the dimension of the parameter space, and r the largest rank of all involved tensors. Then, we have developed a method based on tensor completion to efficiently approximate the interpolation coefficients in an offline phase. In the online phase, expressing the interpolated price as an inner product between two tensors in the TT format allows us to reduce the computational complexity from $\mathcal{O}(n^d)$ to $\mathcal{O}(dnr^2)$. The numerical experiments have confirmed that our method dramatically reduces the storage requirement and the computational complexity of the pricing procedure while maintaining a very high accuracy.

In Chapter 6, we have proposed a technique to price single and multi-asset European options. The method is a combination of Monte Carlo simulation and function approximation, and is a generalization of the algorithm (MCLS) developed in [97]. In particular, we have adapted it to compute integrals with respect to general probability measures, and we have shown that it reduces the variance of the estimator thanks to the function approximation step, as in [97]. We have proposed a new cost analysis and we have combined MCLS with the randomized Kaczmarz algorithm (REK), and with an optimal sampling strategy. Employing REK reduces the memory requirement of MCLS since it does not require the storage of the full Vandermonde matrix. The numerical experiments in option pricing and the computation of a large-scale multivariate integral have shown the effectiveness of the method and of our extension in both low and high dimensions.

Bibliography

[1] Y. Achdou and O. Pironneau. *Computational methods for option pricing*, volume 30 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2005.

[2] D. Ackerer and D. Filipović. Linear credit risk models. *Forthcoming in Finance Stoch.*, 2019.

[3] D. Ackerer and D. Filipović. Option pricing with orthogonal polynomial expansions. *Forthcoming in Math. Finance*, 2019.

[4] D. Ackerer, D. Filipović, and S. Pulido. The Jacobi stochastic volatility model. *Finance Stoch.*, 22(3):667–700, 2018.

[5] N. Akhiezer and N. Kemmer. *The classical moment problem and some related questions in analysis*. University mathematical monographs. Oliver & Boyd, 1977.

[6] M. Bachmayr and A. Cohen. Kolmogorov widths and low-rank approximations of parametric elliptic PDEs. *Math. Comp.*, 86(304):701–724, 2017.

[7] B. W. Bader and T. G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Trans. Math. Software*, 32(4):635–653, 2006.

[8] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015.

[9] J. Ballani and L. Grasedyck. Hierarchical tensor approximation of output quantities of parameter-dependent PDEs. *SIAM/ASA J. Uncertain. Quantif.*, 3(1):852–872, 2015.

[10] D. Barrera, S. Crépey, B. Diallo, G. Fort, E. Gobet, and U. Stazhynski. Stochastic approximation schemes for economic capital and risk margin computations. *ESAIM: ProcS*, 65:182–218, 2019.

[11] C. Bayer, M. Siebenmorgen, and R. Tempone. Smoothing the payoff for efficient computation of basket option prices. *Quant. Finance*, 18(3):491–505, 2018.

[12] A. Beck. *Introduction to nonlinear optimization*, volume 19 of *MOS-SIAM Ser. Optim.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2014.

[13] C. Berg and P. H. Maserick. Exponentially bounded positive definite functions. *Illinois J. Math.*, 28(1):162–179, 1984.

[14] D. Bertsimas and I. Popescu. On the relation between option and stock prices: a convex optimization approach. *Oper. Res.*, 50(2):358–374, 2002.

[15] F. Biagini and Y. Zhang. Polynomial diffusion models for life insurance liabilities. *Insurance Math. Econom.*, 71:114–129, 2016.

[16] D. A. Bini, S. Dendievel, G. Latouche, and B. Meini. Computing the exponential of large block-triangular block-Toeplitz matrices encountered in fluid queues. *Linear Algebra Appl.*, 502:387–419, 2016.

[17] J. W. Blankenship and J. E. Falk. Infinitely constrained optimization problems. *J. Optimization Theory Appl.*, 19(2):261–281, 1976.

[18] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004.

[19] P. Brandimarte. *Numerical methods in finance and economics*. Statistics in Practice. Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, 2nd edition, 2006.

[20] M. Broadie and J. B. Detemple. Option pricing: Valuation models and applications. *Management Science*, 50(9):1145–1177, 2004.

[21] R. E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numer.*, 7:1–49, 1998.

[22] L. Capriotti, Y. Jiang, and A. Macrina. AAD and least-square Monte Carlo: fast Bermudan-style options and XVA Greeks. *Algorithmic Finance*, 6(1-2):35–49, 2017.

[23] P. Carr and D. B. Madan. Option valuation using the fast fourier transform. *J. Comput. Finance*, 2:61–73, 1999.

[24] A. Chkifa, A. Cohen, G. Migliorati, F. Nobile, and R. Tempone. Discrete least squares polynomial approximation with random evaluations—application to parametric and stochastic elliptic PDEs. *ESAIM Math. Model. Numer. Anal.*, 49(3):815–837, 2015.

[25] S. Christensen. A method for pricing American options using semi-infinite linear programming. *Math. Finance*, 24(1):156–172, 2014.

[26] A. Cohen and G. Migliorati. Optimal weighted least-squares methods. *SMAI J. Comput. Math.*, 3:181–203, 2017.

[27] C. Cuchiero. Polynomial processes in stochastic portfolio theory. *Stochastic Process. Appl.*, 129(5):1829–1872, 2019.

[28] C. Cuchiero, M. Keller-Ressel, and J. Teichmann. Polynomial processes and their applications to mathematical finance. *Finance Stoch.*, 16(4):711–740, 2012.

[29] W. Dahmen, R. DeVore, L. Grasedyck, and E. Süli. Tensor-sparsity of solutions to high-dimensional elliptic partial differential equations. *Found. Comput. Math.*, 16(4):813–874, 2016.

[30] F. Delbaen and W. Schachermayer. *The mathematics of arbitrage*. Springer Finance. Springer-Verlag, Berlin, 2006.

[31] F. Delbaen and H. Shirakawa. An interest rate model with upper and lower bounds. *Asia-Pacific Financial Markets*, 9(3):191–209, Sep 2002.

[32] M. A. H. Dempster, J. Kanniainen, J. Keane, and E. Vynckier. *High-performance computing in finance: problems, methods, and solutions*. Chapman & Hall/CRC, 1st edition, 2018.

[33] L. Dieci and A. Papini. Padé approximation for the exponential of a block triangular matrix. *Linear Algebra Appl.*, 308(1-3):183–202, 2000.

[34] L. Dieci and A. Papini. Conditioning of the exponential of a block triangular matrix. *Numer. Algorithms*, 28(1-4):137–150, 2001.

[35] D. Duffie, D. Filipović, and W. Schachermayer. Affine processes and applications in finance. *Ann. Appl. Probab.*, 13(3):984–1053, 2003.

[36] D. J. Duffy. *Finite difference methods in financial engineering: a partial differential equation approach*. Wiley Finance Series. John Wiley & Sons, Ltd., Chichester, 2006.

[37] B. Eriksson and M. Pistorius. Method of moments approach to pricing double barrier contracts in polynomial jump-diffusion models. *Int. J. Theor. Appl. Finance*, 14(7):1139–1158, 2011.

[38] D. Filipović and M. Larsson. Polynomial diffusions and applications in finance. *Finance Stoch.*, 20(4):931–972, 2016.

[39] D. Filipović and M. Larsson. Polynomial jump-diffusion models. *Forthcoming in Stochastic Systems*, 2019.

[40] D. Filipović, M. Larsson, and A. B. Trolle. Linear-rational term structure models. *The Journal of Finance*, 72(2):655–704, 2017.

[41] D. Filipović, M. Larsson, and T. Ware. Polynomial processes for power prices. *arXiv preprint arXiv:1710.10293*, 2018.

[42] D. Filipović, E. Mayerhofer, and P. Schneider. Density approximations for multivariate affine jump-diffusion processes. *J. Econometrics*, 176(2):93–111, 2013.

[43] D. Filipović and S. Willems. A term structure model for dividends and interest rates. *arXiv preprint arXiv:1803.02249*, 2019.

[44] D. Filipović, E. Gourier, and L. Mancini. Quadratic variance swap models. *Journal of Financial Economics*, 119(1):44 – 68, 2016.

[45] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[46] M. Gaß, K. Glau, M. Mahlstedt, and M. Mair. Chebyshev interpolation for parametric option pricing. *Finance Stoch.*, 22(3):701–731, 2018.

[47] A. Gil, J. Segura, and N. M. Temme. *Numerical methods for special functions*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.

[48] M. B. Giles. Multilevel Monte Carlo methods. *Acta Numer.*, 24:259–328, 2015.

[49] M. B. Giles and Y. Xia. Multilevel Monte Carlo for exponential Lévy models. *Finance Stoch.*, 21(4):995–1026, 2017.

[50] P. Glasserman. *Monte Carlo methods in financial engineering*, volume 53 of *Applications of Mathematics*. Springer-Verlag, New York, 2004.

[51] K. Glau, D. Kressner, and F. Statti. Low-rank tensor approximation for Chebyshev interpolation in parametric option pricing. *arXiv preprint arXiv:1902.04367*, 2019.

[52] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 4th edition, 2012.

[53] A. Gorodetsky, S. Karaman, and Y. Marzouk. A continuous analogue of the tensor-train decomposition. *Computer Methods in Applied Mechanics and Engineering*, 347:59–84, 2019.

[54] C. Gourieroux and J. Jasiak. Multivariate Jacobi process with application to smooth transitions. *J. Econometrics*, 131(1-2):475–505, 2006.

[55] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitt.*, 36(1):53–78, 2013.

[56] M. Griebel and M. Holtz. Dimension-wise integration of high-dimensional functions with applications to finance. *J. Complexity*, 26(5):455–489, 2010.

[57] A. Gut. *Probability: a graduate course*. Springer Texts in Statistics. Springer, New York, 2nd edition, 2013.

[58] S. Güttel and Y. Nakatsukasa. Scaled and squared subdiagonal Padé approximation for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 37(1):145–170, 2016.

[59] W. Hackbusch. *Tensor spaces and numerical tensor calculus*, volume 42 of *Springer Series in Computational Mathematics*. Springer, Heidelberg, 2012.

[60] T. Haentjens and K. J. in't Hout. ADI schemes for pricing American options under the Heston model. *Appl. Math. Finance*, 22(3):207–237, 2015.

[61] W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pac. J. Optim.*, 2(1):35–58, 2006.

[62] A.-L. Haji-Ali, F. Nobile, R. Tempone, and S. Wolfers. Multilevel weighted least squares polynomial approximation. *arXiv preprint arXiv:1707.00026*, 2017.

[63] D. Han, X. Li, D. Sun, and J. Sun. Bounding option prices of multi-assets: a semidefinite programming approach. *Pac. J. Optim.*, 1(1):59–79, 2005.

[64] B. Hashemi and L. N. Trefethen. Chebfun in three dimensions. *SIAM J. Sci. Comput.*, 39(5):C341–C363, 2017.

[65] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, 2nd edition, 2009.

[66] S. 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.

[67] N. J. Higham. *Functions of matrices*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008.

[68] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Rev.*, 51(4):747–764, 2009.

[69] N. Hilber, N. Reich, C. Schwab, and C. Winter. Numerical methods for Lévy processes. *Finance Stoch.*, 13(4):471–500, 2009.

[70] N. Hilber, O. Reichmann, C. Schwab, and C. Winter. *Computational methods for quantitative finance*. Springer Finance. Springer, Heidelberg, 2013.

[71] M. Holtz. *Sparse grid quadrature in high dimensions with applications in finance and insurance*, volume 77 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2011.

[72] S. Holtz, T. Rohwedder, and R. Schneider. On manifolds of tensors of fixed TT-rank. *Numer. Math.*, 120(4):701–731, 2012.

[73] R. A. Horn and C. R. Johnson. *Topics in matrix analysis*. Cambridge University Press, Cambridge, 1991.

[74] J. Hull. *Options, futures, and other derivatives*. Pearson Prentice Hall, Upper Saddle River, NJ, 2006.

[75] J. Hull and A. White. The pricing of options on assets with stochastic volatilities. *The Journal of Finance*, 42(2):281–300, 1987.

[76] K. J. in't Hout and J. Toivanen. Application of operator splitting methods in finance. In *Splitting methods in communication, imaging, science, and engineering*, Sci. Comput., pages 541–575. Springer, Cham, 2016.

[77] E. Jondeau and M. Rockinger. Gram-charlier densities. *Journal of Economic Dynamics and Control*, 25(10):1457–1483, 2001.

[78] B. N. Khoromskij. *Tensor numerical methods in scientific computing*, volume 19 of *Radon Series on Computational and Applied Mathematics*. De Gruyter, Berlin, 2018.

[79] B. N. Khoromskij and C. Schwab. Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs. *SIAM J. Sci. Comput.*, 33(1):364–385, 2011.

[80] P. Kloeden and A. Neuenkirch. Convergence of numerical methods for stochastic differential equations in mathematical finance. In *Recent developments in computational finance*, volume 14 of *Interdiscip. Math. Sci.*, pages 49–80. World Sci. Publ., Hackensack, NJ, 2013.

[81] P. E. Kloeden and E. Platen. *Numerical solution of stochastic differential equations*. Springer-Verlag, Berlin, 1992.

[82] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, 2009.

[83] D. Kressner, R. Luce, and F. Statti. Incremental computation of block triangular matrix exponentials with application to option pricing. *Electron. Trans. Numer. Anal.*, 47:57–72, 2017.

[84] D. Kressner, M. Steinlechner, and B. Vandereycken. Low-rank tensor completion by Riemannian optimization. *BIT*, 54(2):447–468, 2014.

[85] D. Kressner, M. Steinlechner, and B. Vandereycken. Preconditioned low-rank Riemannian optimization for linear systems with tensor product structure. *SIAM J. Sci. Comput.*, 38(4):A2018–A2044, 2016.

[86] D. Kressner and C. Tobler. Low-rank tensor Krylov subspace methods for parametrized linear systems. *SIAM J. Matrix Anal. Appl.*, 32(4):1288–1316, 2011.

[87] K. S. Larsen and M. Sørensen. Diffusion models for exchange rates in a target zone. *Math. Finance*, 17(2):285–306, 2007.

[88] J. B. Lasserre. *Moments, positive polynomials and their applications*, volume 1 of *Imperial College Press Optimization Series*. Imperial College Press, London, 2010.

[89] J. B. Lasserre, T. Prieto-Rumeau, and M. Zervos. Pricing a class of exotic options via moments and SDP relaxations. *Math. Finance*, 16(3):469–494, 2006.

[90] P. L'Ecuyer. Quasi-Monte Carlo methods with applications in finance. *Finance Stoch.*, 13(3):307–349, 2009.

[91] M. Lenga. *Representable options*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2017.

[92] S. Liu, C. Oosterlee, and S. Bohte. Pricing options and computing implied volatilities using neural networks. *Risks*, 7:16, 02 2019.

[93] J. Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Trans. Automat. Control*, 54(5):1007–1011, 2009.

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

[95] A.-M. Matache, P.-A. Nitsche, and C. Schwab. Wavelet Galerkin pricing of American options on Lévy driven assets. *Quant. Finance*, 5(4):403–424, 2005.

[96] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[97] Y. Nakatsukasa. Approximate and integrate: Variance reduction in Monte Carlo integration via function approximation. *arXiv preprint arXiv:1806.05492*, 2018.

[98] D. Needell. Randomized Kaczmarz solver for noisy linear systems. *BIT*, 50(2):395–403, 2010.

[99] D. Needell, R. Zhao, and A. Zouzias. Randomized block Kaczmarz method with projection for solving least squares. *Linear Algebra Appl.*, 484:322–343, 2015.

[100] R. Orús. A practical introduction to tensor networks: matrix product states and projected entangled pair states. *Ann. Physics*, 349:117–158, 2014.

[101] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[102] B. N. Parlett. A recurrence among the elements of functions of triangular matrices. *Linear Algebra Appl.*, 14(2):117–121, 1976.

[103] C. Reisinger and R. Wissmann. Numerical valuation of derivatives in high-dimensional settings via PDE expansions. *J. Comput. Finance*, 18(4):95–127, 2015.

[104] W. Rudin. *Real and complex analysis*. McGraw-Hill Book Co., New York, third edition, 1987.

[105] W. Rudin. *Functional Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1991.

[106] K. Sato. *Lévy processes and infinitely divisible distributions*, volume 68 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2013.

[107] R. Schneider and A. Uschmajew. Approximation rates for the hierarchical tensor format in periodic Sobolev spaces. *J. Complexity*, 30(2):56–71, 2014.

[108] R. U. Seydel. *Tools for computational finance*. Universitext. Springer-Verlag, London, sixth edition, 2017.

[109] S. T. Smith. Optimization techniques on Riemannian manifolds. In *Hamiltonian and gradient flows, algorithms and control*, volume 3 of *Fields Inst. Commun.*, pages 113–136. Amer. Math. Soc., Providence, RI, 1994.

[110] F. Statti. Polynomial bounds for European option pricing. *EPFL technical report*, 2019. Available at http://sma.epfl.ch/~anchpcommon/publications/polbound.pdf.

[111] E. M. Stein and J. C. Stein. Stock price distributions with stochastic volatility: An analytic approach. *Review of Financial Studies*, 4:727–752, 1991.

[112] M. Steinlechner. Riemannian optimization for high-dimensional tensor completion. *SIAM J. Sci. Comput.*, 38(5):S461–S484, 2016.

[113] M. Steinlechner. *Riemannian optimization for solving high-dimensional problems with low-rank tensor structure*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2016.

[114] J. Stoyanov and G. D. Lin. Hardy's condition in the moment problem for probability distributions. *Theory Probab. Appl.*, 57(4):699–708, 2013.

[115] T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *J. Fourier Anal. Appl.*, 15(2):262, 2009.

[116] L. N. Trefethen. *Approximation theory and approximation practice*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2013.

[117] R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Math. Program.*, 95(2, Ser. B):189–217, 2003.

[118] A. Uschmajew and B. Vandereycken. Greedy rank updates combined with Riemannian descent methods for low-rank optimization. In *Proceedings of the 2015 International Conference on Sampling Theory and Applications (SampTA)*, pages 420–424. IEEE, 2015.

[119] C. Van Loan. *Computational frameworks for the fast Fourier transform*, volume 10 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[120] E. Wong. The construction of a class of stationary Markoff processes. In *Proc. Sympos. Appl. Math., Vol. XVI*, pages 264–276. Amer. Math. Soc., Providence, R.I., 1964.

[121] S. Y. Wu, S. C. Fang, and C. J. Lin. Relaxed cutting plane method for solving linear semi-infinite programming problems. *J. Optim. Theory Appl.*, 99(3):759–779, 1998.

[122] J. ya Gotoh and H. Konno. Bounding option prices by semidefinite programming: A cutting plane algorithm. *Management Science*, 48(5):665–678, 2002.

[123] H. Zhou. Itô conditional moment generator and the estimation of short-rate processes. *Journal of Financial Econometrics*, 1(2):250–271, 06 2003.

[124] A. Zouzias and N. M. Freris. Randomized extended Kaczmarz for solving least squares. *SIAM J. Matrix Anal. Appl.*, 34(2):773–793, 2013.

Curriculum Vitae

Personal information

Name	Francesco Statti
Date of birth	17.05.1990.
Place of birth	Catanzaro, Italy
Nationality	Swiss and Italian

Education

since Sep 2015	**Doctoral studies in Applied Mathematics** École Polytechnique Fédérale de Lausanne Thesis: *Numerical methods for option pricing: polynomial approximation and high dimensionality* Phd Advisors: Prof. D. Filipović, Prof. D. Kressner
2013 – 2015	**M.Sc. ETH in Mathematics** Department of Mathematics, ETH Zurich Thesis: *Unspanned stochastic volatility in multifactor CIR models* Master Thesis advisor: Prof. M. Larsson
2010 – 2013	**B.Sc. ETH in Mathematics** Department of Mathematics, ETH Zurich Thesis: *Functions of bounded variation* Bachelor Thesis advisor: Prof. H. M. Soner
2005–2009	**Maturità liceale** (secondary school diploma), Liceo di Lugano 2

Work experience

since Sep 2015	**Research and teaching assistant** École Polytechnique Fédérale de Lausanne
2012–2014	**Student teaching assistant** Department of Mathematics, ETH Zurich
2009–2010	**Swiss Army group leader** Switzerland

Publications and preprints

- F. Statti. *Polynomial bounds for European option pricing.* EPFL technical report, 2019.
- K. Glau, D. Kressner, and F. Statti. *Low-rank tensor approximation for Chebyshev interpolation in parametric option pricing.* EPFL technical report, 2019.
- D. Filipović, M. Larsson, and F. Statti. *Unspanned stochastic volatility in the multifactor CIR model.* Mathematical Finance, 29(3):827-836, 2019.
- D. Kressner, R. Luce and F. Statti. *Incremental computation of block triangular matrix exponentials with application to option pricing.* Electron. Trans. Numer. Anal., Vol 47, 57-72, 2017.

Conferences and schools

- Mathicse retreat, June 11–13 2019, Champéry, Switzerland.
  Talk: *Function approximation and Monte Carlo simulation for efficient numerical integration.*
- SIAM conference on Financial Mathematics and Engineering, June 4-7 2019, University of Toronto, Canada.
  Talk: *Low-rank tensor approximation for Chebyshev interpolation in parametric option pricing.*
- Swiss numerics day, May 10 2019, University of Lugano, Switzerland.
  Poster: *Efficient option pricing via function approximation and Monte Carlo simulation.*
- Numerical linear algebra seminar, January 15 2019, NII, Tokyo, Japan.
  Talk: *Incremental computation of the block triangular matrix exponential.*

- Mathicse retreat, June 19–21 2018, St. Croix, Switzerland.
  Talk: *Optimization for European and American option pricing in polynomial models.*
- GAMM annual meeting, March 19–23 2018, TU Munich, Germany.
  Talk: *Incremental computation of the block triangular matrix exponential.*
- Winter school on optimization and operations research, January 14-19 2018, Zinal, Switzerland.
- Opening Conference of the Verona Paris stochastic modeling semester, Dec 18-21 2017, Verona, Italy.
  Poster: *An SDP approach for bounding option prices in polynomial models.*
- Second international conference on computational finance, Sep 4-8 2017, Lisbon, Portugal.
  Talk: *Incremental computation of block triangular matrix exponentials with application to option pricing.*
- Mathicse retreat, June 14–16 2017, Leysin, Switzerland.
  Talk: *An SDP approach for bounding option prices in polynomial models.*
- School and workshop on dynamical models in finance, May 22-24 2017, EPFL, Lausanne, Switzerland.
  Talk: *Incremental computation of block triangular matrix exponentials with application to option pricing.*
- Swiss numerics day, April 28 2017, University of Basel, Switzerland.
  Poster: *Incremental computation of block triangular matrix exponentials with application to option pricing.*
- Mathicse retreat, June 27–29 2016, Leysin, Switzerland.
  Talk: *Incremental moments computation for option pricing.*
- Brown bag seminar, July 1 2015, EPFL, Lausanne, Switzerland.
  Talk: *Unspanned stochastic volatility in multifactor CIR models.*

**Ce document a été imprimé au Centre d'impression EPFL,
imprimerie climatiquement neutre, certifiée myClimate.**