Sparse Grid Quadrature Methods for Computational Finance

Habilitationsschrift

an der

Mathematisch–Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich–Wilhelms–Universität Bonn

eingereicht von

Thomas Gerstner

aus

München

Bonn 2007

For Eva

Contents

1 Introduction **9**

2 Financial Derivatives **17**

2.1 Introduction 17

2.2 Standard Options 18

2.2.1 European Options 18

2.2.2 American Options 19

2.2.3 Bermudean Options 20

2.3 Path-Dependent Options 20

2.3.1 Asian Options 20

2.3.2 Barrier Options 21

2.3.3 Lookback Options 22

2.4 Multi-Asset Options 23

2.4.1 Basket Options 23

2.4.2 Performance-Dependent Options 23

2.5 Interest Rate Derivatives 26

2.5.1 CMO Problem 26

2.6 Greeks 27

3 Stochastic Market Models **29**

3.1 Introduction 29

3.2 Market Assumptions 29

3.3 Single-Asset Models 30

3.3.1 Black-Scholes Model 30

3.3.2 Further Single-Asset Models 31
3.3.3 Parameter Estimation 32
3.4 Multi-Asset Models 33
3.4.1 Full Black-Scholes Model 33
3.4.2 Reduced Black-Scholes Model 34

4 Pricing Approaches **37**
4.1 Introduction 37
4.2 Pricing Principles 37
4.3 Martingale Approach 39
4.3.1 Standard Options 39
4.3.2 Path-Dependent Options 41
4.3.3 Multi-Asset Options 41

5 Valuation Formulas **43**
5.1 Introduction 43
5.2 European Options 43
5.3 Path-Dependent Options 45
5.3.1 Asian Options 45
5.3.2 Barrier Options 46
5.3.3 Lookback Options 46
5.4 Performance-Dependent Options 47
5.4.1 Full Model Valuation Formula 47
5.4.2 Reduced Model Valuation Formula 50

6 Hyperplane Arrangements **57**
6.1 Introduction 57
6.2 Definitions 58
6.3 Enumeration 59
6.3.1 Simple Cell Enumeration Algorithm 60
6.3.2 Correspondence between Intersection Points and Cells 61
6.3.3 Intersection Points with a Box 63

6.3.4 Cell Enumeration Algorithm . . . 64
6.4 Orthant Decomposition . . . 65
6.4.1 Signed Polyhedral Decomposition . . . 66
6.4.2 Orthant Decomposition Algorithm . . . 68
6.5 Computational Results . . . 69

7 Simulation Methods 73
7.1 Introduction . . . 73
7.2 Tree methods . . . 73
7.2.1 CRR model . . . 74
7.2.2 Binomial Method . . . 75
7.3 Stochastic Meshes . . . 76
7.4 Univariate Integration Methods . . . 77
7.4.1 Trapezoidal Rule . . . 78
7.4.2 Clenshaw-Curtis Formulas . . . 79
7.4.3 Gauss and Gauss–Patterson Formulas . . . 79
7.4.4 Domain Transformation . . . 81
7.5 Multivariate Integration Methods . . . 82
7.5.1 Product Approach . . . 82
7.5.2 Monte Carlo Methods . . . 83
7.5.3 Quasi-Monte Carlo Methods . . . 84
7.6 Path Discretization . . . 85
7.6.1 Random Walk . . . 85
7.6.2 Brownian Bridge . . . 86

8 Sparse Grids 87
8.1 Regular Sparse Grids . . . 87
8.1.1 Basic Construction . . . 88
8.1.2 Implementation . . . 89
8.1.3 Error Bounds . . . 90
8.2 Dimension-Adaptive Sparse Grids . . . 90
8.2.1 Dimension-Adaptive Refinement . . . 91

8.2.2 Generalized Sparse Grids 92
8.2.3 Basic Algorithm 93
8.2.4 Error Estimation 95
8.2.5 Data Structures 97
8.2.6 Complexities 101
8.3 Derivative Pricing using Sparse Grids 101
8.3.1 Transformation 102
8.3.2 Integration of Multivariate Normal Distributions 103

9 Numerical Results 105
9.1 Example Problem 105
9.2 Path-Dependent Derivatives 106
9.3 CMO problem 108
9.4 Performance-Dependent Options 111
9.4.1 Full Model 111
9.4.2 Reduced Model 114

10 Conclusions 117

List of Figures 119

List of Tables 123

Bibliography 123

Chapter 1

Introduction

Computational Finance

Computational finance (also known as financial engineering) is an interdisciplinary field which uses mathematical finance, stochastic methods, numerical algorithms and computer simulations to aid practitioners in banks, insurance companies or other financial institutions with trading, hedging and investment decisions. Its main aim is to determine as accurately as possible the financial risk that financial instruments create. Areas where computational finance techniques are employed include investment banking and management, corporate strategic planning, securities and derivatives trading and risk management.

Of particular interest in computational finance is the pricing of derivative securities, whose most well-known representatives are various types of options. The price of these derivatives depends on the future development of some underlying asset or a set of assets such as stocks, stock indices, bonds, exchange rates or commodities. Financial derivatives are typically traded at special derivatives exchanges or directly over-the-counter. Their (mathematically) fair price is an important guideline for all market participants.

The usual approach in derivative security pricing is to start with a suitable model for the future development of the underlying asset or assets. Typically, stochastic differential equations or systems of stochastic differential equations are employed here to account for the random nature of the price developments. Under these model assumptions, derivative prices and risks can be determined using techniques from stochastic calculus.

A fundamental result from financial derivatives pricing theory is that, under certain assumptions, the fair price of a derivative security can be represented as an expected value. If the expectation is written as an integral, its dimension is in many cases high or even infinite. This dimension depends on the number of independent stochastic factors, which are related, for example, to the number of assets under consideration or the number of time steps in a time discretization.

In nearly all cases, the arising integrals cannot be solved analytically or can be reduced into easy computational form. Thus, numerical methods, i.e. approximation methods, are required for their solution. Furthermore, often (for example for the computation of so-called Greeks or sensitivities) a high-accuracy solution is needed. This way, financial derivative pricing problems can easily become computationally quite challenging even for parallel supercomputers.

The Curse of Dimension

The main reason for this difficulty is the so-called *curse of dimension* (a term first called so by Bellman [5]), which can be understood in two ways. First, one observes that in classical numerical integration methods (i.e., those based on product approaches) the amount of work N required in order to achieve a prescribed accuracy ε grows exponentially with the dimension d,

$$\varepsilon(N) = O(N^{-r/d}), \tag{1.1}$$

for functions with bounded total derivatives up to order r (see, e.g., [19]). Thus, for a fixed smoothness already in moderate dimensions the order of convergence becomes so slow that high accuracies cannot be obtained in practice. The situation gets worse as the dimension increases unless the smoothness increases with the dimension as well. The latter assumption is usually not fulfilled in practice.

The curse of dimension can also be approached from the point of numerical complexity theory. There it has been shown that for many integration problems (i.e., for integrands from certain standard function spaces) even the minimum amount of work in order to achieve a prescribed accuracy grows exponentially with the dimension [123]. These lower bounds hold for all algorithms from a specific algorithmic class (i.e., those using linear combinations of function evaluations). Such integration problems are therefore usually called intractable. However, application problems are often in a different or smaller problem class and thus may be tractable, although the correct classification can be difficult. In addition, there may exist (e.g., non-linear or quantum) algorithms which stem from a different algorithmic class and thus may be able to break the curse of dimension.

Monte Carlo and Quasi-Monte Carlo Methods

Randomized algorithms, whose probably best-known representative are Monte Carlo methods, is one such class of algorithms. Here, the integrand is evaluated at a set of (pseudo-) randomly chosen points and the approximation of the integral is computed as the average of these function values. This way, the average amount of work (i.e., the number of function evaluations) in order to reach an accuracy ε is for integrands with bounded variance

$$\varepsilon(N) = O(N^{-1/2}) \tag{1.2}$$

and is thus independent of the dimension. Nevertheless, the convergence rate is quite low and a high accuracy is only achievable with a tremendous amount of work. Indeed, in computational finance, much of the computing time of today's supercomputers is used just for the generation of random numbers.

Therefore, so-called Quasi-Monte Carlo algorithms have attained a lot of attention in the last years. Here, the integrand is evaluated not at random but at structurally determined points such that the discrepancy (a measure for the maximum distance between points) of these points is smaller than that for random points. Then, for functions with bounded (mixed) variation, the complexity becomes

$$\varepsilon(N) = O(N^{-1}(\log N)^{d-1}) \tag{1.3}$$

and is thus almost half an order better than the complexity of the Monte Carlo approach [93]. In addition, the error bounds are deterministic. However, the dimension enters through a logarithmic term and this dependence on the dimension often causes trouble in high-dimensional problems.

Sparse Grids

In contrast to the product approach, the convergence rate of Monte Carlo and Quasi-Monte Carlo methods does not depend on the smoothness of the problem. Thus, in general, smoother integrands are not computed more efficiently than non-smooth ones. The first method which makes use of the smoothness of the integrand and at the same time does not suffer from the curse of the dimension is the so-called *sparse grid method* [133]. In its basic form, this method dates at least back to the Russian mathematician Smolyak [118]. In this approach, multivariate quadrature formulas are constructed by a combination of tensor products of univariate formulas. Of all possible combinations of one-dimensional quadrature formulas only those are taken whose corresponding indices are contained in the unit simplex. This way, the complexity becomes

$$\varepsilon(N) = O(N^{-s}(\log N)^{(d-1)(s+1)}), \tag{1.4}$$

for functions from spaces with bounded mixed derivatives up to order s. Thus, for $s > 1$, a better convergence rate than for Quasi-Monte Carlo can be expected. For very smooth integrands ($s \to \infty$), the convergence will even be exponential.

The sparse grid method is directly applicable to derivative security pricing problems which lead to smooth integrands which are often encountered during the pricing of interest rate derivatives [46, 120]. Further examples are mortgage-backed securities, collaterated debt obligations and insurance contracts [45].

For option pricing problems, however, the corresponding integrands are typically not smooth and the convergence of the sparse grid method deteriorates strongly. In many cases, the integrands have at least discontinuous first derivatives ($s = 1$), in some cases even the integrand itself is discontinuous ($s = 0$). This way, the efficiency of sparse grid approach suffers significantly up to the point where it is less efficient than Quasi-Monte Carlo or even Monte Carlo methods.

As a second problem, the sparse grid method is, just like Quasi-Monte Carlo methods, largely, but not completely independent of the dimension of the problem. The dimension d arises as the exponent of a logarithmic factor in the convergence rate. This leads to an (albeit slow) degradation of the convergence rate when d increases. Therefore, it is necessary to find novel numerical methods which can deal with these high-dimensional integration problems.

Pricing Financial Derivatives using Sparse Grids

In this thesis, we address these two problems of missing smoothness and dimension-dependence. To this end, we develop novel sparse grid quadrature methods which are able to deal with non-smooth and high-dimensional problems such as they arise in computational finance.

A closer look at the integrands of typical financial derivative pricing problems reveals that they are, in fact, mostly smooth. The discontinuities only arise along a lower-dimensional (usually (d–1)-dimensional) manifold. The location of this manifold is in general unknown beforehand. For numerical integration purposes, however, the manifold can be found pointwise along lines of integration in a predetermined direction by zero finding methods. Since the integrand is often zero on one side of the manifold, the integration domain can be mapped along this predetermined direction to cover only the nonzero part of the integrand. This way, the formally non-smooth integration problem can be transformed into a smooth one to which sparse grid integration methods can be applied without penalty. In practice, the additional computational costs for the zero finding are more than offset by the much higher convergence rate.

The second problem is the high dimensionality of many financial derivative pricing problems. For path-dependent options, the high dimension arises from the number of time steps in the time discretization. For multi-asset options, the dimension is determined by the number of assets or the number of stochastic processes which are used to describe the asset movements. At first sight, all these dimensions are of equal importance. However, a hierarchical discretization of the simulation paths using a Brownian bridge (in the case of path-dependent derivatives) or a principal components analysis of the covariance matrix underlying the stochastic processes leads to a different weighting of the individual directions thereby reducing the effective dimension of the problem. The classical sparse grid method cannot utilize this information since it treats all directions equally. Generalized sparse grid methods such as anisotropic and dimension-adaptive sparse grids can recognize the effective dimension of the integration problem. This way, sparse grid methods can also be applied to high-dimensional option pricing problems.

As the main application and proof-of-concept of this thesis, we take a closer look at so-called *performance-dependent options.* These are multi-asset derivatives whose payoff depends on the performance of one asset in comparison to a set of benchmark assets. Performance-dependent options are, for example, used to determine the fair prices of bonus programs of large companies. Their payoffs and thus the corresponding integrands are typically discontinuous. Thereby, the discontinuities arise not only on a single manifold but on several intersecting manifolds, which makes their valuation numerically quite challenging. For these options, we derive valuation formulas for so-called full and reduced multivariate Black-Scholes models. In the latter case, the manifolds of discontinuities form a hyperplane arrangement. We show that the cells in this hyperplane arrangement can be efficiently enumerated and decomposed into simple tensor-product (orthant) integration regions. Inside each region, the integrand is smooth and sparse grid methods can be applied. This way, performance-dependent options can be efficiently valuated also for large benchmarks.

Main Contributions of this Thesis

The main contributions of this thesis to computational finance are thus as follows:

- a sparse grid quadrature method utilizing zero-finding and transformation along lines of integration to numerically treat discontinuities along manifolds which typically arise in the payoff of options,
- a dimension-adaptive numerical integration method which uses dimension reduction based on a Brownian bridge discretization or a principal component analysis for the treatment of high-dimensional financial problems,
- a general valuation formula for performance-dependent options and a novel algorithm for its evaluation which uses the cell enumeration and orthant decomposition of hyperplane arrangements.

These techniques are applied to the valuation of standard options such as European call and put options, path-dependent derivatives such as Asian and barrier options and multi-asset derivatives such as basket and performance-dependent options.

Publications

Parts of this thesis have been published as journal articles and conference proceedings or are currently in the progress of publication. In particular, these are:

- T. Gerstner, M. Griebel, Numerical integration using sparse grids, *Numerical Algorithms*, 18:209–232, 1998.
- T. Gerstner, M. Griebel, Dimension-adaptive tensor-product quadrature, *Computing*, 71(1):65–87, 2003.
- T. Gerstner, M. Holtz, Geometric tools for the valuation of performance-dependent options, in *Computational Finance and its Applications II*, M. Costatino and C. and Brebbia, eds., WIT Press, pp. 161–170, 2006.

- T. Gerstner, M. Holtz, Valuation of performance-dependent options, *Applied Mathematical Finance*, 2007, to appear.
- T. Gerstner, M. Holtz, The cell enumeration and orthant decomposition of hyperplane arrangements, *Discrete and Computational Geometry*, 2007, in preparation.
- T. Gerstner, M. Holtz, R. Korn, Valuation of performance-dependent options in a Black-Scholes framework, in *Proceedings Numerical Methods for Finance*, CRC Press, 2007, to appear.

Outline

The outline of this thesis is as follows.

In *Chapter 2*, we illustrate the various types of financial derivatives which we will examine in this thesis. Besides standard European options, we particularly consider path-dependent and multi-asset options as well as interest rate derivatives.

Then, in *Chapter 3*, we take a look at stochastic market models which are used to describe the underlying market. Besides standard diffusion models which are based on geometric Brownian motion, we consider full and reduced multivariate Black-Scholes models.

In *Chapter 4*, we discuss the fundamental principles of option pricing. Here, the martingale approach as well as approaches based on partial differential equations are illustrated.

The topic of *Chapter 5* are valuation formulas, i.e. closed-form solutions, which can be obtained for special cases of models and derivatives. Besides standard results for European options and special path-dependent options, we derive novel pricing formulas for performance-dependent options.

Hyperplane arrangements play an important role for the valuation of these performance-dependent options and we take a look at them in *Chapter 6*. We derive efficient algorithms for the cell enumeration and for the orthant decomposition of general hyperplane arrangements.

In *Chapter 7*, we then illustrate the use of simulation for the pricing of financial derivatives. Thereby, deterministic and stochastic tree methods, hierarchical discretization methods of simulation paths and numerical integration methods for the computation of expectations such as Monte Carlo and Quasi-Monte Carlo methods are described.

The sparse grid approach is investigated in detail in *Chapter 8*. Besides classical sparse grids, we discuss anisotropic sparse grids, and dimension-adaptive sparse grids. Thereby, we especially discuss the efficient implementation of the different methods and their application to financial derivatives pricing. In this context, suitable transformations of the integrand, zero finding methods for the treatment of discontinuities in the integrand and dimension reduction techniques are required.

In *Chapter 9* we show numerical pricing results for different types of options. Thereby, we compare the various sparse grid methods with standard methods. We will see that in many cases the sparse grid approach shows a superior accuracy and convergence rate in comparison to the these methods.

Concluding remarks are finally drawn in *Chapter 10.* We reiterate the main results of the previous chapters and give an outlook on possible extensions of the shown methods and indicate further applications.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Dr. Michael Griebel for his constant support in these years. Without him, this work would not have been possible. I would also like to thank all of my current and former colleagues at the Institute for Numerical Simulation, especially Markus Holtz and Dr. Jochen Garcke, for their interest and help with various mathematical and not-so-mathematical problems. Furthermore, I would like to thank Prof. Dr. Ralf Korn for the introduction to performance-dependent options and for his help with the derivation of the pricing formulas. Many thanks for their interest in the field of computational finance also go to my former students Vera Gerig, Thomas Mertens, Torsten Nahm, Melanie Reiferscheid, Sebastian Wahl, Claudia Warawko and Allan Zulficar.

Last and most of all I would like to sincerely thank my wife Eva for her never ending support and love.

Chapter 2

Financial Derivatives

2.1 Introduction

Financial derivatives are securities whose value depends on the price of one or more other underlying assets, for example stocks, stock indices, bonds, exchange rates or commodities. Financial derivatives are either traded at special derivatives exchanges in a similar way to the underlying assets or directly over-the-counter between financial institutions.

The main topic of this work is the determination of the fair values of such financial derivatives under certain model assumptions. This fair value does not have to be equal to the market value of the derivative which results from supply and demand and thus the subjective notions of the value of the derivative from buyers and sellers. Nevertheless, it the fair value is an important notion for all market participants. Historically, mathematically well-founded fair prices which were derived by Black, Scholes and Merton [7] eventually enabled the systematic trade of financial derivatives after the introduction of derivative exchange at the Chicago Board of Trade in 1973.

Numerical methods, i.e. approximation algorithms, play a crucial rule for the valuation of financial derivatives since in almost all cases of derivatives and corresponding model assumptions no closed-form solution for their fair value can be derived. In the course of this work we will illustrate a variety of methods which have been developed for the pricing of different types of derivatives and model assumptions.

But first, we have to take a closer look at some types of financial derivatives which are currently traded in the markets or which are used for the assessment and hedging of risks. Besides standard European, American and Bermudean options, we particularly consider path-dependent and multi-asset options as well as interest rate derivatives. Let us remark here that this list is by no means comprehensive. The variety of financial derivatives has been growing constantly in the last years. An overview is given, e.g., in [70, 134].

2.2 Standard Options

Options are one of the most important types of financial derivatives. On one hand, they are bought by speculative investors due to their leverage effect. On the other hand they are used for hedging already entered positions against future developments of the market. Let us start with the definition of a standard option.

Definition 2.2.1 (Standard Option) *A standard (vanilla) option bears the right, but not the obligation, to buy or sell a certain number of the underlying securities for a prescribed price within a certain time period. Options which allow the holder to buy the underlying securities are called call options, while options with include the right to sell them are called put options. The prescribed price is often called strike or exercise price and the time in which the option can be exercised is called exercise time or exercise period.*

The number of underlying securities which can be bought or sold is typically determined by a subscription ratio, such as 1:10. Options are traded for a variety of underlyings, typically stocks but also stock indices, currencies, interest rates, bonds, commodities like gold or oil, and even other options. Since the price of an option depends on the price of its underlying, options are typical examples of financial derivatives.

Standard options are emitted typically by a bank or some other financial institution which fixes the subscription ratio, the strike price and the exercise period. The buyer of an option can exercise the option within the exercise period and thus buy or sell the underlying securities for the strike price, sell the option itself again, or, at the end of the exercise period have the option expire valuelessly. The buyer of the option pays a price for this exercise right. The determination of a fair value for this price is important for buyers as well as sellers of options.

We will now consider the most basic types of options, so-called European, American and Bermudean options. Note that these names have no geographical meaning, most traded standard options are of American type.

2.2.1 European Options

The simplest type of options are *European options.* Nevertheless, they are of great practical (and theoretical) importance.

Definition 2.2.2 (European Option) *An European option is a standard option where the exercise period consists of a single point in time in the future, the exercise time $T > 0$.*

Let $V(S,t)$ denote the value of a European option. This value depends on the current time $t = 0$ and the price of the underlying $S(t)$, which is assumed to vary with time. The strike price is denoted by K. Here and in the following we fix the subscription ratio at $1:1$ since the option prices scale in proportion with the subscription ratio.

Definition 2.2.3 (Payoff of Standard Options) *The value of a European call option at the exercise time T is given by the payoff*

$$V(S,T) := (S(T) - K)^+ := \max\{S(T) - K, 0\}. \tag{2.1}$$

The value of a European put option at the exercise time is correspondingly

$$V(S,T) = (K - S(T))^+. \tag{2.2}$$

If the price of the underlying at the exercise time $S(T)$ is larger than K, then the holder of a call option can buy the underlying for the price K and can sell it immediately for the price $S(T)$ and realize a profit of $S(T) - K$ (for the case that no transaction costs are paid). If the price is lower, then the holder will have the option expire valuelessly since the underlying is worth less than the exercise price. For put options, the roles of $S(T)$ and K are simply reversed.

When computing option prices one can, at least for European options, confine oneself either to call or to put options since the so-called put-call parity (see, e.g. [70, 131])

$$S(t) - V_{Put}(S,t) - V_{Call}(S,t) = Ke^{-r(T-t)} \tag{2.3}$$

holds. Here, r is the riskless interest rate, i.e., the interest a riskless investment generates, which is assumed to be constant over time.

2.2.2 American Options

In contrast to European options, *American options* can be exercised at any time $t \leq T$ up to the exercise time.

Definition 2.2.4 (American Option) *An American option is a standard option where the exercise period is the whole time interval $(0,T]$, where $t = 0$ corresponds to the current time.*

At time T, the value of an American option is equal to the value of a European option and given by the payoff functions (2.1) and (2.2) for call and put options, respectively. However, there exists no put-call parity for American options.

The value $V(S,t)$ of an American option is always at least as large as the value of a corresponding European option. Since the number of exercise times of an American option is a superset of exercise times of a European option, an American option gives the holder more rights and thus cannot have a lower value.

As already mentioned, most traded standard options are of American type. The ability to exercise the option at any time is of high practical value.

2.2.3 Bermudean Options

Somehow in between American and European Options (also in the geographical sense) are so-called *Bermudean options.* Bermudean options allow the holder to exercise the option at a prescribed set of times.

Definition 2.2.5 (Bermudean Option) *A Bermudean option is a standard option which can be exercised at a prescribed set of times $t_j > 0$, $1 \leq j \leq M$.*

Typically, Bermudean options can be exercised daily, weekly or monthly within the exercise period. If we set $T = \max_{1\leq j\leq M} t_j$, then the value of a Bermudean option at time T is again equal to the value of a European option and given by the payoff functions (2.1) and (2.2).

The value $V^{Ber}(S,t)$ of a Bermudean option is in between the values of a corresponding European option $V^{Eur}(S,t)$ and a corresponding American option $V^{Amer}(S,t)$, i.e.,

$$V^{Eur}(S,t) \leq V^{Ber}(S,t) \leq V^{Amer}(S,t), \tag{2.4}$$

since a Bermudean option implies more rights than a European and less rights than an American option.

Bermudean options are often used to approximate the value of American options. The values of a series of Bermudean options with an increasing number of (equally distributed) exercise times converge to the value of an American option, see, e.g., [51].

2.3 Path-Dependent Options

Path-dependent options are financial derivatives whose value not only depends on the price of the underlying at the exercise time but on all prices of the underlying between the starting time (usually, the current time) and the exercise time. In the following, we shortly illustrate three popular examples of path-dependent options, namely Asian options, barrier options and lookback options. Note that path-dependent options are usually of European type, i.e., they can be exercised only at the exercise time T.

2.3.1 Asian Options

The idea behind Asian options is that for standard European options a strong up- or downward movement of the underlying asset shortly before the exercise date has an unwantedly large influence on the value of the option. Therefore, in these options, the strike price is not compared with the value of the underlying asset at the exercise date but with its average value over the whole life time of the option. We discern here between discrete and continuous averages as well as arithmetic and geometric means.

Discrete Averages

In the case of a discrete averaging over finitely many time points $t_j, 1 \leq j \leq M$ where again $T = \max_{1 \leq j \leq M} t_j$, an Asian option is defined by a payoff function of the following type.

Definition 2.3.1 (Discrete Average Asian Option) *The payoff of a discrete average Asian call option is given by*

$$V(S,T) = \left(\frac{1}{M} \sum_{j=1}^{M} S(t_j) - K \right)^+ \tag{2.5}$$

in the case of a discrete arithmetic mean and by

$$V(S,T) = \left(\left(\prod_{j=1}^{M} S(t_j) \right)^{1/M} - K \right)^+ \tag{2.6}$$

for a discrete geometric mean. For put options the roles of the average and the strike are reversed.

Continuous Averages

Instead of a very large number of averaging time steps, the corresponding continuous means can be used which leads to continuous average Asian options.

Definition 2.3.2 (Continuous Average Asian Option) *The payoff of a continuous arithmetic average Asian call option is given by*

$$V(S,T) = \left(\frac{1}{T} \int_0^T S(t)\ dt - K \right)^+ \tag{2.7}$$

while for a continuous geometric average Asian call option it is given by

$$V(S,T) = \left(e^{\left(\frac{1}{T} \int_0^T \ln(S(t))\ dt \right)} - K \right)^+ . \tag{2.8}$$

Again, for put options the roles of the average and the strike are reversed.

2.3.2 Barrier Options

Another often traded example for path-dependent options are barrier options. For barrier options the option expires worthlessly as soon as the underlying reaches a certain level (barrier). In knock-out options the option expires as soon as the underlying exceeds (up-out) or falls below the barrier (down-out). Knock-in options are worthless until the underlying exceeds (up-in) or falls below (down-in) the barrier. Barrier options are frequently traded since the additional risk reduces their price in comparison to standard options.

As an example, we consider the payoff of a down-out call option.

Definition 2.3.3 (Down-Out Barrier Call Option) *The payoff of a down-out barrier call option with barrier H is given by*

$$V(S,T) = \begin{cases} (S(T)-K)^+ & \text{if } S(t) > H \text{ for } 0 \le t \le T \\ 0 & \text{else} \end{cases} . \tag{2.9}$$

The payoffs of the other types of barrier options have a similar structure, though. Note that in barrier options, the barrier is usually observed continuously. In the discrete variant, the barrier would only be checked at a prescribed set of times $t_j, 1 \le j \le M$, similar to discrete average Asian options.

2.3.3 Lookback Options

In the design of lookback options, the same consideration as for Asian options is done, i.e. that sudden changes in the price of the underlying at the end of the exercise period too strongly influence the option price. In Asian options the temporal mean of the price of the underlying is taken. In lookback options, instead the maximum or the minimum of the prices is taken. This way, the holder of the option can obtain the maximum profit with respect to the development of the asset price. As a disadvantage, lookback options are relatively expensive in comparison to the other considered types of options.

One discerns between lookback options with fixed and variable strike price.

Definition 2.3.4 (Lookback Option) *For a fixed strike price, the payoff of a lookback call option function reads*

$$V(S,T) = \left(\max_{0 \le t \le T} S(t) - K \right)^+ \tag{2.10}$$

while for a variable strike the payoff is given by

$$V(S,T) = \left(S(T) - \min_{0 \le t \le T} S(t) \right)^+ . \tag{2.11}$$

For lookback put options the subtractions are reversed.

Again, the continuous observations can be replaced by discrete ones at time points $t_j, 1 \le j \le M$, for example, if only the daily closing prices enter the maximum or minimum. This reduces, to a certain extent, the price of lookback options again.

2.4 Multi-Asset Options

Up to now, we have only considered options on single underlyings. In contrast, multi-asset options are written on two or more underlyings (usually assets). The multi-asset options we consider here are of European type in the sense that their payoff only depends on (all) the asset values at the exercise time T. Note that there also exist American-type and path-dependent multi-asset options.

We assume that there are n assets involved in total. The price of the i-th asset varying with time t is denoted by $S_i(t), 1 \leq i \leq n$. All asset prices at the end of the exercise time $t = T$ are collected in the vector $\mathbf{S} = (S_1(T), \ldots, S_n(T))$.

2.4.1 Basket Options

The payoff of a basket option is determined by the average of the asset prices at time T which is compared to the strike price K. For simplicity, we assume that all assets in the basket are given the same weight.

Definition 2.4.1 (Basket Option) *For an arithmetic average basket call option, the payoff reads*

$$V(\mathbf{S}, T) = \left(\frac{1}{n} \sum_{i=1}^{n} S_i(T) - K \right)^{+}, \tag{2.12}$$

while for a geometric average, the payoff is given by

$$V(\mathbf{S}, T) = \left(\left(\prod_{i=1}^{n} S_i(T) \right)^{1/n} - K \right)^{+}. \tag{2.13}$$

For basket put options, the roles of the average and the strike are reversed.

Weighted averages are also often used, especially in the arithmetic average. Thereby, in the summation, each asset price is multiplied with a weight $c_i, 1 \leq i \leq n$ with $\sum_{i=1}^{n} c_i = 1$ indicating the importance of the asset in the basket.

2.4.2 Performance-Dependent Options

Performance-dependent options are a special class of multi-asset options which we consider in more detail here.

Motivation

Companies make big efforts to bind their staff to them for longer periods of time in order to prevent a permanent change of executives in important positions. Besides high wages, such efforts are long-term incentive and bonus schemes. One widespread form of such schemes consists in giving the participants a conditional award of shares. If the participant stays with the company for at least a prescribed time period he or she will receive a certain number of company shares at the end of the period. Typically, the exact amount of shares is determined by a performance criterion such as the company's gain over the period or its ranking among comparable firms (the peer group). This way, such bonus schemes induce uncertain future costs for the company. Especially for the shareholders of the company, the fair value of such bonus programs would be an interesting figure.

An upper bound on this fair value would be the maximum number of possibly needed shares at end of the bonus scheme. A better upper bound would be the value of standard call options on the maximum number of possibly needed shares. Both bounds significantly overestimate the true value of the bonus program since the performance criterion is not taken into account.

The appropriate financial instruments to derive this fair value are so-called performance-dependent options, see, e.g. [80]. Such options simply include the performance criterion in their contract. Using these options, the company would be able to purchase exactly the number of required shares at the end of the scheme. This way, the fair price of the bonus program is given by the value of the corresponding performance-dependent options. Let us remark here that performance-dependent options can, when traded, also be used for pure performance speculation purposes.

Payoff profile

Performance-dependent options are financial derivatives whose payoff depends on the performance of one asset in comparison to other assets at the end of a given period. For hedging purposes of a bonus scheme, the asset under consideration is the stock of the considered company while the other assets are the stocks of benchmark companies.

We again assume that there are n assets involved in total. The asset of the considered company gets assigned label 1 and the $n-1$ benchmark assets are labeled from 2 to n. In order to define the payoff of a performance-dependent option, we denote the relative price increase of stock i over the time interval $[0, T]$ by

$$\Delta S_i := S_i(T)/S_i(0). \tag{2.14}$$

The performance of the first asset in comparison to a given strike price K (typically, $K = S_1(0)$) and in comparison to the benchmark assets at time T is saved in a ranking vector $\mathbf{Rank}(\mathbf{S}) \in \{+, -\}^n$ which is defined as follows.

Definition 2.4.2 (Ranking vector) *The ranking vector* $\mathbf{Rank}(\mathbf{S})$ *is defined by*

$$Rank_1 := \begin{cases} + & \text{if } S_1(T) \geq K, \\ - & \text{else} \end{cases} \quad \text{and} \quad Rank_i := \begin{cases} + & \text{if } \Delta S_1 \geq \Delta S_i, \\ - & \text{else} \end{cases} \tag{2.15}$$

for $i = 2, \ldots, n$.

This means, if the first asset outperforms benchmark asset i we denote this by a plus sign in the i-th component of the ranking vector Rank, otherwise, there is a minus sign. For each possible ranking $\mathbf{R} \in \{+,-\}^n$, a bonus factor $a_{\mathbf{R}} \in \mathbb{R}^+$ defines the payoff of the performance-dependent option.

Let us remark that it is important to distinguish between a possible ranking denoted $\mathbf{R}$ and the realized ranking which is induced by $\mathbf{S}$ and is denoted by $\mathbf{Rank}$ here.

Now, we are able to define the payoff of a performance-dependent option.

Definition 2.4.3 (Performance-Dependent Option) *The payoff of a performance-dependent option at time T is defined by*

$$V(\mathbf{S},T) := a_{\mathbf{Rank}}\,(S_1(T)-K)^+ \tag{2.16}$$

We always define $a_{\mathbf{R}} = 0$ if $R_1 = -$ such that the payoff can be written as

$$V(\mathbf{S},T) = a_{\mathbf{Rank}}\,(S_1(T)-K). \tag{2.17}$$

Example payoff profiles

In the following, we illustrate some possible choices for the bonus factors $a_{\mathbf{R}}$.

Example 2.4.4 *Performance-independent option:*

$$a_{\mathbf{R}} = \begin{cases} 1 & \text{if } R_1 = + \\ 0 & \text{else.} \end{cases} \tag{2.18}$$

In this case, we recover a plain vanilla European call option on the stock S_1.

Example 2.4.5 *Linear ranking-dependent option:*

$$a_{\mathbf{R}} = \begin{cases} m/(n-1) & \text{if } R_1 = + \\ 0 & \text{else.} \end{cases} \tag{2.19}$$

Here, m denotes the number of outperformed benchmark assets. The payoff only depends on the rank of the considered company in the benchmark. If the company ranks first, there is a full payoff $(S_1(T)-K)^+$. If it ranks last, the payoff is zero. In between, the payoff increases linearly with the number of outperformed benchmark assets.

Example 2.4.6 *Outperformance option:*

$$a_{\mathbf{R}} = \begin{cases} 1 & \text{if } \mathbf{R} = (+,\dots,+) \\ 0 & \text{else.} \end{cases} \tag{2.20}$$

A payoff only occurs if $S_1(T) \geq K$ and if all benchmark assets are outperformed.

Example 2.4.7 *Linear ranking-dependent option combined with an outperformance condition:*

$$a_{\mathbf{R}} = \begin{cases} m/(n-1) & \text{if } R_1 = + \text{ and } R_2 = + \\ 0 & \text{else.} \end{cases} \tag{2.21}$$

The bonus depends linearly on the number m of outperformed benchmark companies like in Example 2.4.5. However, the bonus is only payed if company two is outperformed. Company two could, e.g., be the main competitor of the considered company.

Let us remark here that several differences between the pricing of standard derivatives and the pricing of executive stock options which are not addressed here are thoroughly discussed in [71, 72]. In these papers, only performance-independent executive stock options are considered, though.

2.5 Interest Rate Derivatives

Finally, we take a look at an interest rate derivative, the so-called collateralized mortgage obligation (CMO) problem. The CMO problem attained some interest several years ago as a benchmark problem in computational finance [15, 100]. Let us remark that special interest rate derivatives also arise during the asset/liability management of insurance contracts, see [45].

2.5.1 CMO Problem

A typical collateralized mortgage obligation problem consists of several tranches which derive their cash flows from an underlying pool of mortgages [15, 100]. The problem is to estimate the expected value of the sum of present values of future cash flows for each of the tranches. We consider a pool of mortgages with a maturity of τ years where cash flows are obtained monthly yielding $M = 12 \cdot \tau$ time steps.

Definition 2.5.1 (CMO Problem) *In the CMO problem, the present value v of the future cash flows is given by*

$$v(\xi_1, \ldots, \xi_d) := \sum_{k=1}^{M} u_k m_k \tag{2.22}$$

with

$$\begin{aligned}
u_k &:= \prod_{j=0}^{k-1}(1+i_j)^{-1},\\
m_k &:= cr_k((1-w_k)+w_kc_k),\\
r_k &:= \prod_{j=1}^{k-1}(1-w_j),\\
c_k &:= \sum_{j=0}^{d-k}(1+i_0)^{-j},\\
i_k &:= K_0^k e^{\xi_1+\ldots+\xi_k} i_0,\\
w_k &:= K_1+K_2\arctan(K_3 i_k+K_4).
\end{aligned}$$

The variables u_k, m_k, w_k, r_k, and i_k are the discount factor, the cash flow, the prepaying mortgages, the remaining mortgages and the interest rate for month k, respectively.

The number of prepaying mortgages w_k in month k depends in a nonlinear way on the current interest rate i_k which is modelled by the arctan function. The constant $K_0 := e^{-\sigma^2/2}$ is chosen to normalize the log–normal distribution, i.e. $E(i_k) = i_0$. The initial interest rate i_0, the monthly payment c, and K_1, K_2, K_3, K_4 are further constants of the model.

2.6 Greeks

The so-called Greeks or Greek letters are the partial derivatives of the option price with respect to variables and parameters which have an influence on the option price.

The most important Greeks for single-asset options indicate the sensitivity of the option price to changes in the price of the underlying and time.

- **Delta** $\Delta = \frac{\partial V}{\partial S}$: Delta measures the sensitivity of the option price with changes in the value of the underlying. It is often used to derive hedging strategies.
- **Gamma** $\Gamma = \frac{\partial^2 V}{\partial S^2}$: Gamma measures the sensitivity of Delta to changes in the value of the underlying. It is important when second order effects have to be controlled.
- **Theta** $\Theta = \frac{\partial V}{\partial t}$: Theta indicates how the option price will evolve in time. For hedging strategies, it is important to know if the value of the option is likely to change significantly in the near future, which is indicated by a large Theta.

Furthermore, Greeks with respect to model parameters (see section 3) are often considered.

- **Rho** $\rho = \frac{\partial V}{\partial r}$: If the interest rate is not assumed to be constant, Rho measures the sensitivity of the option price with changes in the interest rate.
- **Vega** $\Lambda = \frac{\partial V}{\partial \sigma}$: Vega measures the sensitivity of the option price with respect to the volatility σ of the underlying.

For multi-asset options, similar Greeks can be defined, for example the Delta with respect to each underlying. But since the number of different Greeks can become quite large their importance in practice is limited.

Chapter 3

Stochastic Market Models

3.1 Introduction

In the following, we will take a look at often used models for the future development of single as well as multiple interacting asset prices, in particular so-called Black-Scholes models. In the univariate case, we will consider two methods for the determination of the most important parameter in this model, the volatility. First, we have to secure a few important market assumptions.

3.2 Market Assumptions

The following assumptions on the market are usually made:

- there are no transaction costs or taxes,
- the interest rates for loaning and lending are equal and constant for all parties,
- all parties have access to all information,
- securities and credits are available at any time and in any quantity.
- short sales are permitted,
- the individual trade does not influence the price,
- there are no arbitrage opportunities (i.e. there is no riskless profit).

The first few assumptions are made only for simplification purposes and can later be suspended or suitably modelled. Especially the last assumption of absence of arbitrage is of central importance for the fair valuation of financial derivatives, though.

3.3 Single-Asset Models

One of the most basic stochastic models for stocks was developed by Bachelier about 1900. This model is still used today also for other types of securities. It is the foundation of the pioneering works of Black, Scholes and Merton [7] on option pricing.

3.3.1 Black-Scholes Model

In the Black-Scholes model, the future development of the underlying is modelled by means of a geometric Brownian motion and follows a linear stochastic differential equation (SDE).

Definition 3.3.1 (Black-Scholes Model) *The Black-Scholes model for a single underlying asset is given by the SDE*

$$dS(t) = \mu S(t)\ dt + \sigma S(t)\ dW(t), \tag{3.1}$$

where μ represents the constant drift, σ the constant volatility and $W(t)$ a one-dimensional Wiener process (standard Brownian motion).

A Wiener process is a Markov process with properties $W(0) = 0$ and $W(t) \sim N(0, t)$ for $t > 0$. Thereby, $N(0, t)$ is the Gaussian normal distribution with mean 0 and variance t. Above notation is just an abbreviated form for the Itô integral equation

$$S(t) = S(0) + \int_0^t \mu S(u)\ du + \int_0^t \sigma S(u)\ dW(u). \tag{3.2}$$

For this integral equation there exists a closed-form solution as

$$S(t) = S(0)e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W(t)}, \tag{3.3}$$

which can be shown via Itôs lemma.

For option pricing, the stochastic process has to be transformed into its risk-neutral form. In the Black-Scholes model, only the drift μ has to replaced by the riskless interest rate r. This way, the explicit solution becomes

$$S(t) = S(0)e^{(r - \frac{1}{2}\sigma^2)t + \sigma W(t)}. \tag{3.4}$$

Dividing by $S(0)$ and taking the logarithm of both sides results in

$$\ln(S(t)/S(0)) = (r - \frac{1}{2}\sigma^2)t + \sigma W(t). \tag{3.5}$$

This way, one can see that the value increment $S(t)/S(0)$ is normally distributed with mean 0 and variance t and thus $S(t)$ is lognormally distributed. For the expectation and variance of S at time t we therefore have

$$E(S(t)) = S(0) \cdot e^{rt} \tag{3.6}$$

and

$$\begin{aligned} Var(S(t)) &= E(S^2(t)) - (E(S(t)))^2 = S^2(0)e^{(2r+\sigma^2)t} - (S(0)\cdot e^{rt})^2 \\ &= S^2(0)e^{2rt}(e^{\sigma^2 t} - 1). \end{aligned} \tag{3.7}$$

3.3.2 Further Single-Asset Models

The Black-Scholes model is by no means the only stochastic model which is used to describe the future development of assets (see, e.g., [91]). One point of criticism is that the Black-Scholes model does not properly reflect the dependence of the option price on the volatility. This lead to the idea that the volatility should follow its own stochastic process.

Definition 3.3.2 (Stochastic Volatility Model) *In the stochastic volatility model the asset price dynamics are given by the system of SDEs*

$$dS(t) = \mu S(t)\ dt + \sigma(t)S(t)\ dW(t) \tag{3.8}$$

$$d\sigma(t) = a\sigma(t)\ dt + b\sigma(t)\ d\tilde{W}(t) \tag{3.9}$$

with some constants a, b and with $W(t)$ and $\tilde{W}(t)$ being two Wiener process with correlation $\rho\, dt$.

Stochastic volatility models have the disadvantage that three more parameters (a, b and ρ) have to be estimated from market data. Also, they are more difficult to discretize and simulate than the Black-Scholes model since no closed-form solution of the system is known.

Another criticism point of the Black-Scholes model is that it underestimates extreme up- and downward movements of many assets, such as stocks. This problem can be removed by using more heavy-tailed distributions for the random increments. Popular examples are jump-diffusion models where extreme events are modelled by jumps of the underlying. To this end, additional jump term is added to the Black-Scholes model.

Definition 3.3.3 (Jump-Diffusion Model) *In a jump-diffusion model, the asset price follows the SDE*

$$dS = \mu S dt + \sigma S dW + \eta S dN \tag{3.10}$$

where N is a Poisson process with intensity λ, i.e.

$$dN = \begin{cases} 0 & \textit{with probability } 1 - \lambda dt \\ 1 & \textit{with probability } \lambda dt \end{cases}$$

and η is an impulse function which generates a jump from S to $S(1+\eta)$.

Many forms of η such as normal, singular or hypersingular distributions have been proposed in the literature. Jump-diffusion models, however, have as disadvantage that they result in an incomplete which makes option pricing by martingale methods much more difficult.

3.3.3 Parameter Estimation

In the Black-Scholes model, two parameters occur, the drift μ and the volatility σ which have to be determined from market data. As we have seen, the drift does not occur in the risk-neutral form of the stochastic differential equation, the volatility plays a very important role, however. We will now consider two methods for volatility estimation.

Historical volatility

One possibility for the determination of the volatility consists in the observation of past prices of the underlying. This historical volatility corresponds to the variance of the logarithmic prices over past times. Let t_k, $0 \le k \le n$, be $n+1$ points in time and $S(t_k)$ the prices of the underlying at these times. Since the prices are lognormally distributed in the Black-Scholes model, the the historical volatility can be computed by

$$\sigma^2 = \frac{1}{n-1} \sum_{j=1}^{n} (\ln(S(t_j)/S(t_{j-1})) - \bar{S}) \tag{3.11}$$

where

$$\bar{S} = \frac{1}{n} \sum_{j=1}^{n} \ln(S(t_j)/S(t_{j-1})). \tag{3.12}$$

The implementation of this formula at first sight requires two for-loops. However, a numerically stable evaluation using one loop only can be obtained with Algorithm 3.3.4 (see, e.g. [113]).

Algorithm 3.3.4 (Historical Volatility)

```
$\alpha = \ln(S(t_1)/S(t_0))$
$\beta = 0$
$for\ \ j = 2, \dots, n$
    $\gamma = \ln(S(t_j)/S(t_{j-1})) - \alpha$
    $\alpha = \alpha + \gamma/j$
    $\beta = \beta + \gamma^2(j-1)/j$
$\sigma = \sqrt{\beta/(M-1)}$
```

A further advantage of Algorithm 3.3.4 is the possibility to process market data (e.g. tick data) online without intermediate storage.

Implied volatility

Alternatively, the volatility can be computed from the market price of other options on the same underlying. This method is often used since in the Black-Scholes model actually the future and not the past volatility has to be used. The volatility implied by the marked is for trading purposes even more important than the option price itself. If an algorithm for approximation of option prices with varying volatility and its Vega Λ is known, the implied volatility can be computed by iterative zero finding, e.g., using the Newton-Raphson method

$$\sigma_{j+1} = \sigma_j - \frac{V(\sigma_j) - V}{\Lambda(\sigma_j)}. \tag{3.13}$$

starting with an estimated volatility of σ_0. Here, $V(\sigma_j)$ is the option price for the iterate σ_j, $\Lambda(\sigma_j)$ the corresponding Vega and V the market price of the option. This way, the Newton-Raphson for the computation of the implied volatility can be described as in Algorithm 3.3.5.

Algorithm 3.3.5 (Implied Volatility)

```
$\sigma = \sigma_0$
*for* $it = 1, \ldots, MAXIT$
    $P = V(\sigma_j)$
    *if* $(P - V < TOL)$ *break*
    $\sigma = \sigma + (P - V)/\Lambda(\sigma_j)$
```

The Newton-Raphson method is particularly effective for the computation of the implied volatility if a closed-form solution for the option price and its Vega derivative is known, see section 5.2.1.

3.4 Multi-Asset Models

Now, we consider some generalizations of the Black-Scholes model for several interacting assets, see, e.g. [63, 74, 81]. To this end, again systems of stochastic differential equations are used. Here, we discern between two cases. In the so-called full model, the number of stochastic processes equals the number of assets while in the so-called reduced model, the number of stochastic processes is smaller. In both cases, the resulting markets are complete, only if the number of stochastic processes is larger than the number of assets, the market would become incomplete [91].

3.4.1 Full Black-Scholes Model

We start with the full Black-Scholes model where the number of stochastic processes equals the number of assets n.

Definition 3.4.1 (Full Black-Scholes Model) *In the full multivariate Black-Scholes model, the asset price dynamics of n assets is given by the system of SDEs*

$$dS_i(t) = S_i(t)\left(\mu_i dt + \sum_{j=1}^{n} \sigma_{ij} dW_j(t)\right) \tag{3.14}$$

for $i = 1, \dots, n$, where μ_i denotes the drift of the i-th stock, σ the $n \times n$ volatility matrix of the stock price movements and $W_j(t), 1 \le j \le n$, Brownian motions.

The matrix $\sigma\sigma^T$ is assumed to be positive definite. The explicit solution of the system of SDEs (3.14) is given by

$$S_i(T) = S_i(\mathbf{X}) = S_i(0)\exp\left(\mu_i T - \bar{\sigma}_i + \sqrt{T}\sum_{j=1}^{n} \sigma_{ij} X_j\right) \tag{3.15}$$

for $i = 1, \dots, n$ with

$$\bar{\sigma}_i := \frac{1}{2}\sum_{j=1}^{n} \sigma_{ij}^2 T \tag{3.16}$$

and $\mathbf{X} = (X_1, \dots, X_n)$ being a $N(\mathbf{0}, \mathbf{I})$-normally distributed random vector.

The full Black-Scholes model is typically used if the number of assets is small. The entries of the volatility matrix can be estimated efficiently based on historical data. To this end, the covariance of the logarithmic prices is estimated in a similar way as in Algorithm 3.3.4.

3.4.2 Reduced Black-Scholes Model

For a larger number of assets, however, the parameter estimation problem can become more and more ill-conditioned resulting in eigenvalues of $\sigma\sigma^T$ which are close to zero. In this case, so-called reduced Black-Scholes models are typically used. There, it is assumed that the asset price movements are driven by $d < n$ stochastic processes.

Definition 3.4.2 (Reduced Black-Scholes Model) *The price dynamics of n assets is given in the reduced multivariate Black-Scholes model by a system of $d < n$ SDEs*

$$dS_i(t) = S_i(t)\left(\mu_i dt + \sum_{j=1}^{d} \sigma_{ij} dW_j(t)\right) \tag{3.17}$$

for $i = 1, \dots, n$. Here, μ_i denotes the drift of the i-th stock, σ the $n \times d$ volatility matrix of the stock price movements and $W_j(t)$ the corresponding Wiener processes.

Again, the matrix $\sigma\sigma^T$ is assumed to be positive definite. By Itô's formula, the explicit solution of the system of SDEs is given by

$$S_i(T) = S_i(\mathbf{X}) = S_i(0)\exp\left(\mu_i T - \bar{\sigma}_i + \sqrt{T}\sum_{j=1}^{d} \sigma_{ij} X_j\right) \tag{3.18}$$

for $i = 1, \ldots, n$ with

$$\bar{\sigma}_i := \frac{1}{2} \sum_{j=1}^{d} \sigma_{ij}^2 \, T. \tag{3.19}$$

The entries of the volatility matrix can again be estimated based on historical data, sometimes starting with a $n \times n$ volatility matrix for the full model. If the assets are all part of a stock index, a reduction can be achieved, for instance, by grouping assets in the same area of business. The matrix entry σ_{ij} then reflects the correlation of stock i with business area j. Such a grouping can often be obtained without much loss of information e.g. using principal component analysis (PCA), as was confirmed empirically [84, 86, 87].

Chapter 4

Pricing Approaches

4.1 Introduction

The prices of financial derivatives depend on the expected future development of the underlying assets. This development is presumed to be given by a stochastic differential equation or system of equations some of which were illustrated in the previous chapter. Under these model and market assumptions, formulas for the fair prices of financial derivatives can be mathematically derived which is the subject of this chapter.

This fair price is usually given as an expectation or the solution of a partial differential equation. The connection between these representations shows the Feynman-Kac theorem see, e.g., [66]. In both cases, the price of the derivative can be computed after suitable discretization (in space and time) and solution of the resulting discrete problem (see Figure 4.1). In the first case, an integration problem has to be computed, in the second case a large linear system has to be solved. For a fast and accurate computation of derivative prices, special numerical methods have to be used in these discretization and solution steps.

In the following, we do not follow the PDE approach but consider only the martingale approach corresponding to the left branch in the tree of Figure 4.1. Note, however, that also in the PDE approach in some cases (especially for multi-asset options) the curse of dimension is encountered. In this case, also sparse grid methods can be applied, see [88, 108] for basket options.

4.2 Pricing Principles

The following three main principles from the mathematical theory of derivatives pricing are important here, see [52]:

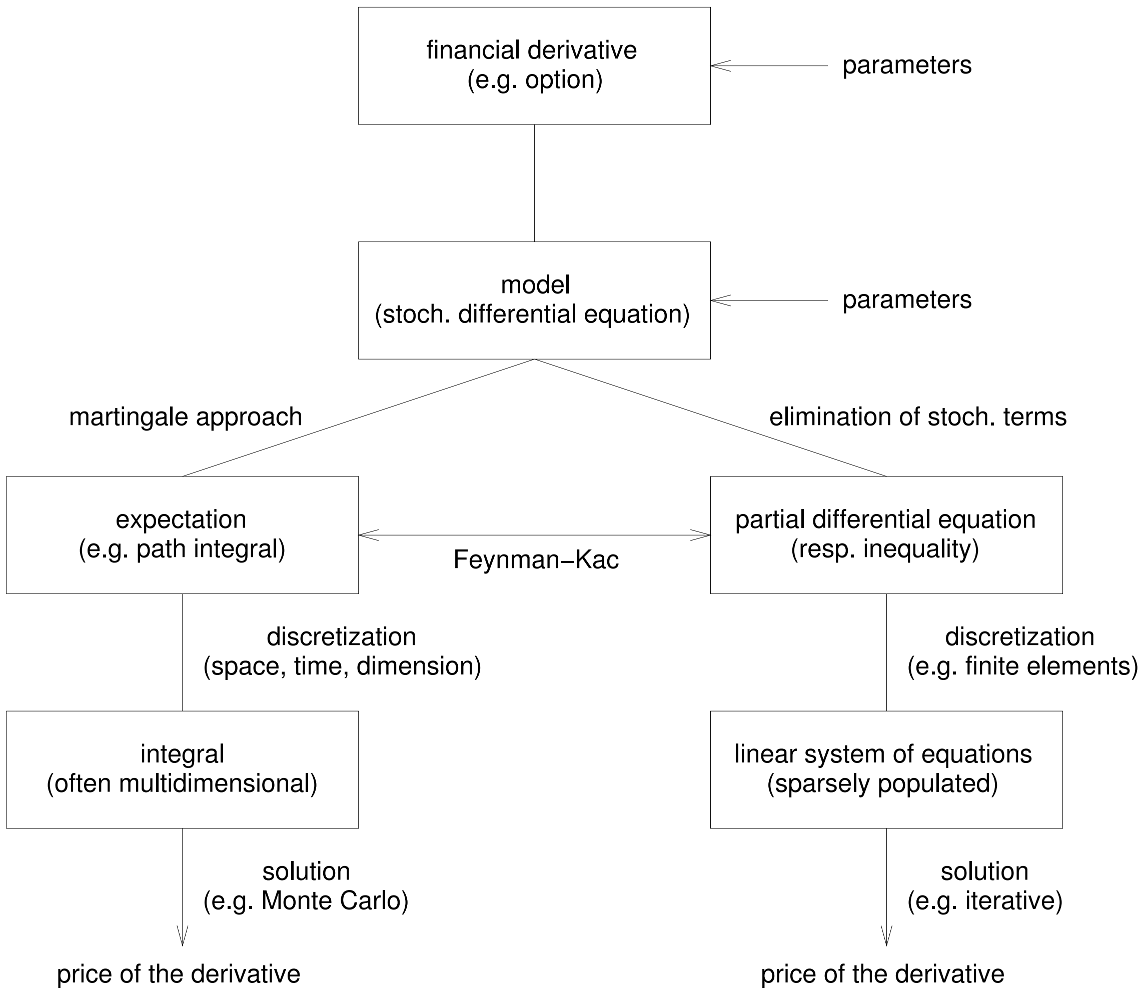


Figure 4.1: Overview and organization of the various methods for the valuation of financial derivatives.

1. If a derivative security can be perfectly replicated (hedged) through trading in other assets, then the price of the derivative security is the cost of the replicating trading strategy.

2. Discounted asset prices are martingales under a probability measure associated with the choice of numeraire. Prices are expectations of discounted payoffs under such a martingale measure.

3. In a complete market, any payoff (satisfying modest regularity conditions) can be synthesized through a trading strategy, and the martingale measure associated with a numeraire is unique. In an incomplete market there are derivative securities that cannot be perfectly hedged; the price of such a derivative is not completely determined by the prices of other assets.

The first principle tells us what the price of a derivative security ought to be but shows us not how this price can be evaluated. The second principle tells us how to represent prices as expectations. The third principle states under what conditions the price of a derivative security is determined by the prices of other assets so that the first and second principles apply.

4.3 Martingale Approach

The martingale approach is one of the main principles for option pricing. It says that the fair price of an option is the discounted expectation of the payoff under the risk-neutral probability distribution of the underlying economic factors.

4.3.1 Standard Options

The martingale representation of fair financial derivative prices has been found much later than the pioneering works of Black, Scholes and Merton which are based on the PDE representation.

Theorem 4.3.1 (Fair Value of Financial Derivatives) *The fair value of a financial derivative which can be exercised at the set of exercise times $\mathcal{T}$ is given by*

$$V(S,0) = \sup_{t\in\mathcal{T}} \; e^{-rt} E^*[V(S,t)] \tag{4.1}$$

where E^* is the expectation under the equivalent martingale measure.

Proof: see, e.g., [63]. It uses a change of numeraire, Itô's lemma and Girsanov's theorem. The complete proof is beyond the scope of this thesis.

Let us remark that this representation of fair prices is quite general and can be applied to a large class of financial derivatives with different underlying stochastic models and payoff functions. We will, for now, stay in the Black-Scholes world, though.

As already mentioned, for the Black-Scholes model, in the equivalent martingale measure the drift μ is replaced by the riskless interest rate r. For European options the set of exercise times $\mathcal{T} = \{T\}$. This way, the martingale representation of the fair value is explicit since $V(S,T)$ is the payoff of the option which is known as part of the option contract.

Theorem 4.3.2 (Fair Value of European Options) *The fair value of a European call option under the Black-Scholes model is given by*

$$V(S,0) = e^{-rT} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \left(S(0) e^{(r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T}x} - K \right)^+ dx \; . \tag{4.2}$$

For a European put option, the difference is simply reversed.

Proof (c.f., [63]): From Theorem 4.3.1 we have

$$V(S,0) = e^{-rT} E^*[V(S,T)] \; .$$

Since $W(t) \sim N(0,t)$, the expectation can be written as an integral with respect to a standard normal distribution

$$V(S,0) = e^{-rT} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} V(S,T) \; dx \; .$$

Plugging in the explicit solution (3.4) for $S(T)$, and using the scaling property of the normal distribution

$$N(0,t) = \sqrt{t} N(0,1) \tag{4.3}$$

we get

$$V(S,0) = e^{-rT} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} V(S(0) e^{(r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T}x}, T) \; dx \; .$$

From the definition of the payoff in (2.1) for European call options and in (2.2) for European put options we get the assertion. □

For Bermudean options, the set of exercise times $\mathcal{T} = \{t_j\}_{j=1}^n$, and for American options, $\mathcal{T} = \{t \leq T\}$. For these types of options, the representation of Theorem 4.3.1 is only implicit since the fair value of the option now depends on the fair values of the option in the future which are not known beforehand. Nevertheless, numerical methods can be also directly applied to representation (4.1), see, e.g. [51, 128], they are much more involved, though.

4.3.2 Path-Dependent Options

In a sense, path-dependent options are more similar to European options than to Bermudean or American options, since their fair value can be explicitly stated and is given by

$$V(S,0) = e^{-rT} E^{*}[V(S,T)] \tag{4.4}$$

Since the payoff $V(S,T)$ depends on the asset prices $S(t)$ at specific times $t < T$ but not on the option prices $V(S,t)$ for $t < T$, this representation is explicit. Because the payoff depends on the path of $S(t)$, we first need a suitable representation of the path in order to explicitly write down the fair value as a (multivariate) integral. We will encounter such representations later in section 7.6. After a path discretization, the dimension of the resulting integration problem is equal to the number of time steps.

4.3.3 Multi-Asset Options

The multi-asset options we have considered are of European type in the sense that they can be exercised only at the exercise time T. The full and reduced multivariate Black-Scholes models induce a complete market which gives the existence of a unique equivalent martingale measure, see, e.g., [74]. Under the equivalent martingale measure, all drifts μ_i in (3.15) and (3.18) are replaced by the riskless interest rate r for each asset. This way, we have the following representation for the fair value of multi-asset options.

Theorem 4.3.3 (Fair Value of Multi-Asset Options) *The fair value of a (European style) multi-asset option is under the full or the reduced multivariate Black-Scholes model given by*

$$V(\mathbf{S},0) = e^{-rT} \int_{R^d} \varphi(\mathbf{x})\ V(\mathbf{S},T)\ d\mathbf{x}. \tag{4.5}$$

where $\varphi(\mathbf{x}) := \varphi_{\mathbf{0},\mathbf{I}}(\mathbf{x})$ is the multivariate normal distribution with mean $\mathbf{0}$ and covariance matrix $\mathbf{I}$ and the asset prices $S_i(T)$ are given by (3.18).

Proof: We start with the martingale representation

$$V(\mathbf{S},0) = e^{-rT} E^{*}[V(\mathbf{S},T)]$$

where E^{*} is the expectation under the equivalent martingale measure. Plugging in the density function φ of the underlying random vector $\mathbf{X}$ (note that $\mathbf{S} = \mathbf{S}(\mathbf{X})$), we get the assertion. $\square$

Note that here, $d \leq n$, which incorporates both the full and the reduced Black-Scholes models.

Now we take a closer look at the representation of the fair value of performance-dependent options in the martingale approach.

Theorem 4.3.4 (Fair Value of Performance-Dependent Options) *The fair value of a performance-dependent option with payoff (2.17) is under the full or the reduced multivariate Black-Scholes model given by*

$$V(\mathbf{S},0) = e^{-rT} \int_{R^d} \sum_{\mathbf{R}\in\{+,-\}^n} a_{\mathbf{R}}(S_1(T) - K)\, \chi_{\mathbf{R}}(\mathbf{S})\varphi(\mathbf{x})\; d\mathbf{x} \tag{4.6}$$

where the characteristic function $\chi_{\mathbf{R}}(\mathbf{S})$ *is defined by*

$$\chi_{\mathbf{R}}(\mathbf{S}) = \begin{cases} 1 & \textit{if } \ \mathbf{Rank}(\mathbf{S}) = \mathbf{R}, \\ 0 & \textit{else.} \end{cases} . \tag{4.7}$$

Proof: We use the martingale representation of Theorem 4.3.3 and identify the different rankings via the characteristic function. □

Note that the expectation runs over all possible rankings $\mathbf{R}$ and the rankings $\mathbf{R}$ as well as asset prices $\mathbf{S}$ are functions of $\mathbf{x}$. Recall that the covariance matrix σ is present in the asset prices $\mathbf{S}$ which are given by (3.18).

Chapter 5

Valuation Formulas

5.1 Introduction

In this chapter, we take a look at valuation formulas, i.e., explicit solutions to derivative pricing problems under the Black-Scholes model assumptions. Explicit solutions exist only in rare cases. Due to their nature, however, they are very important for the validation of computer implementations of numerical methods.

Note that an explicit solution does not imply an easy computation, as we will see in the case of performance-dependent options. However, the usage of a valuation formula usually gives rise to more efficient computational algorithms.

5.2 European Options

The valuation formula of European options under the Black-Scholes model assumptions is known as Black-Scholes formula. Due to its simplicity, the Black-Scholes formula is of great practical importance for the trading of options.

Theorem 5.2.1 (Black-Scholes Formula) *In the Black-Scholes model, the fair price of a European call option is given by*

$$V(S,0) = S(0)N(d_1) - Ke^{-rT}N(d_2) \tag{5.1}$$

and of a European put option by

$$V(S,0) = Ke^{-rT}N(-d_2) - S(0)N(-d_1) \tag{5.2}$$

with

$$d_1 = \frac{\ln(S(0)/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \tag{5.3}$$

and

$$d_2 = d_1 - \sigma\sqrt{T} \tag{5.4}$$

Thereby, $N(x)$ *is the cumulative normal distribution with mean 0 and variance 1*

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}}\, dy. \tag{5.5}$$

Proof (c.f., [63]): We start with the representation of the fair value of a European call option (4.2)

$$V(S(0),0) = e^{-rT} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \left(S(0)e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}x} - K \right)^{+} dx. \tag{5.6}$$

The Black-Scholes formula can now be derived as the exact solution of this integral. To this end let χ *be the solution of the equation* $S(0)e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}\chi} - K = 0$*, i.e.*

$$\chi = \frac{\ln\frac{K}{S(0)} - (r - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \tag{5.7}$$

then we have

$$V(S(0),0) = e^{-rT} \int_{\chi}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \left(S(0)e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}x} - K \right) dx. \tag{5.8}$$

The first summand of this integrand can be computed by

$$\begin{aligned} e^{-rT}\int_{\chi}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} S(0)e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}x}dx &= S(0)\int_{\chi}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\sigma\sqrt{T}-x)^2}dx \\ &= S(0)N(\sigma\sqrt{T}-\chi) \end{aligned} \tag{5.9}$$

and for the second one we have correspondingly

$$e^{-rT} \int_{\chi}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} K dx = K e^{-rT} N(-\chi) \tag{5.10}$$

which yields the Black-Scholes formula for European call options. For European put options the derivation is analogous. □

Note that there is a variety of other approaches to derive this formula including the original one of Black and Scholes as the explicit solution of the Black-Scholes PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0. \tag{5.11}$$

Cumulative Normal Distribution

For the evaluation of the Black-Scholes formula the computation of the cumulative normal distribution is necessary. For this purpose there are fast approximation methods available which are based on piecewise polynomial interpolation. One in practice well-established method is the Moro algorithm [90] which partitions the domain into the three strips $[0, 1.87]$, $[1.87, 6]$, and $[6, \infty]$. For $x < 0$ one computes $1 - N(-x)$. The Moro algorithm is able to compute the cumulative normal distribution with an accuracy of 8 digits. It is given in Algorithm 5.2.2.

Algorithm 5.2.2 (Computation of the Cumulative Normal Distribution)

```
$A0 = 0.398942270991$   $A1 = 0.020133760596$   $A2 = 0.002946756074$
$B1 = 0.217134277847$   $B2 = 0.018576112465$   $B3 = 0.000643163695$
$C0 = 1.398247031184$   $C1 = -0.360040248231$   $C2 = 0.022719786588$
$D0 = 1.460954518699$   $D1 = -0.305459640162$   $D2 = 0.038611796258$
$D3 = -0.003787400686$
if $x \leq 1.87$
  $x2 = x * x$
  $N = 0.5 + x * (A0 + (A1 + A2 * x2) * x2)/(1.0 + (B1 + (B2 + B3 * x2) * x2) * x2)$
else if $x < 6$
  $N = 1.0 - ((C0 + (C1 + C2 * x) * x)/(D0 + (D1 + (D2 + D3 * x) * x) * x)^{16}$
else $N = 1.0$
```

5.3 Path-Dependent Options

In the following, we take a look at valuation formulas for path-dependent options, in particular, Asian options, barrier options and lookback options.

5.3.1 Asian Options

In the case of a discrete geometric mean, the fair price of an Asian option can be derived as a generalized Black-Scholes formula [134]. Under the Black-Scholes model assumptions, the fair value of a discrete geometric average Asian option is given by

$$\begin{aligned} V(S,0) &= S(0) \cdot A \cdot N(d + \sigma\sqrt{T_1}) - Ke^{-rT} N(d) \qquad (5.12) \\ A &= e^{-r(T-T_2)-\sigma^2(T_2-T_1)/2} \\ d &= \frac{\ln(S(0)/K) + (r - \frac{1}{2}\sigma^2)T_2}{\sigma\sqrt{T_1}} \\ T_1 &= T - \frac{n(n-1)(4M+1)}{6M^2}\Delta t \\ T_2 &= T - \frac{(M-1)}{2}\Delta t \end{aligned}$$

For a discrete arithmetic mean or other averaging methods, no closed-form solution can be given. A generalization of the Black-Scholes formula also exists for the continuous geometric mean.

Under the Black-Scholes model assumptions, the fair value of a continuous geometric average Asian option is given by

$$V(S(0),0) = Se^{-\frac{1}{2}(rT+\frac{1}{6}\sigma^2)}N(d+\sigma\sqrt{T/3}) - Ke^{-rT}N(d) \tag{5.13}$$

where

$$d = \frac{\ln(S(0)/K) + \frac{1}{2}(r - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T/3}} \tag{5.14}$$

Again no closed-form solution can be given for the continuous arithmetic mean or a different mean.

5.3.2 Barrier Options

For the prices of barrier options, also corresponding Black-Scholes formulas can be given, see [134]. We only take a look at the known down-out call option. In the Black-Scholes model, the fair value of a down-out call option is given by

$$\begin{aligned} V(S,T) &= V_{bs}(S,\bar{H}) - Z \cdot V_{bs}(H^2/S,\bar{H}) \\ &+ (\bar{H} - K)e^{-rT}(N(d(S,\bar{H})) - Z \cdot N(d(H^2/S,\bar{H}))) \end{aligned} \tag{5.15}$$

with

$$Z = \left(\frac{H}{S}\right)^{\frac{2r}{\sigma^2}-\frac{1}{2}} \tag{5.16}$$

and

$$\bar{H} = \max\{H, K\}. \tag{5.17}$$

Here, $V_{bs}(S,K)$ is the Black-Scholes price for a European call option with current asset price S and exercise price K

For more complex barrier options (e.g. for barrier options of American type or for floating barriers) no closed-form solution is available.

5.3.3 Lookback Options

For both fixed and floating strike lookback options there are generalized Black-Scholes formulas [134]. Under the Black-Scholes model assumptions, the fair value of a fixed strike lookback option is given by

$$V(S,T) = e^{-rT}(S-K) + V_{bs}(S,S) + \frac{S\sigma^2}{2r}\left(N(d_{bs}(S,S)+\sigma\sqrt{T}) - e^{-rT}N(-d_{bs}(S,S))\right) \tag{5.18}$$

and of a variable strike lookback option by

$$V(S,T) = V_{bs}(S,S) + \frac{S\sigma^2}{2r}\left(e^{-rT}N(d_{bs}(S,S)+\sigma\sqrt{T}) - N(-d_{bs}(S,S))\right) \tag{5.19}$$

Since both valuation formulas are similar, one sees that there is no large difference in the characteristics of lookback options with fixed and variable strike. The fixed strike variant is more popular though. Again, there are no closed form solutions for more complex lookback options.

5.4 Performance-Dependent Options

We will now derive similar valuation formulas for performance-dependent options. To this end, we aim to deduce analytical expressions for the solution of Theorem 4.3.4. For the full model case, a valuation formula can be derived in a straightforward way [50]. For reduced models, the derivation of a valuation formula is more involved [48, 49], as we will see.

Note that various other multi-asset option pricing problems not discussed in this section allow closed form solutions, see, e.g., [134, 16]. A valuation approach for American-style performance-dependent options using a fairly general Lévy model for the underlying securities is presented in [29]. There, a least-squares Monte Carlo scheme is used for the numerical solution of the model, but only in the case of one benchmark process. Thus, the problem of high-dimensionality does not arise.

5.4.1 Full Model Valuation Formula

We for now assume that the number of stochastic processes d equals the number of assets n. Looking at Theorem 4.3.4, we see that the fair price of a performance-dependent can be obtained by computing a d-dimensional integral. The integral can, at least at first sight, not be solved analytically and therefore requires numerical approaches for its solution. The integrand, however, is discontinuous induced by the jumps of the bonus factors $a_{\mathbf{R}}$ (see the examples in section 2.4.2). Therefore, numerical integration methods will perform poorly and only Monte Carlo integration can be used without penalty. Thus, high accuracy solutions will be hard to obtain.

In the following, we derive a representation of the integral in terms of multivariate normal distributions. We nevertheless distinguish between d and n in order to be able to reuse some of the results also for the reduced model case.

Let us first recall that the multivariate normal distribution with mean zero, limits $\mathbf{b} = (b_1, \ldots, b_d)$ and $d \times d$ covariance matrix $\mathbf{C}$ is defined by

$$\Phi(\mathbf{C}, \mathbf{b}) := \int_{-\infty}^{b_1} \cdots \int_{-\infty}^{b_d} \varphi_{\mathbf{0},\mathbf{C}}(\mathbf{x}) \; dx_d \ldots dx_1 \tag{5.20}$$

with the Gauss kernel

$$\varphi_{\mu,\mathbf{C}}(\mathbf{x}) := \frac{1}{(2\pi)^{d/2}(\det \mathbf{C})^{1/2}}\, e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{C}^{-1}(\mathbf{x}-\mu)}. \tag{5.21}$$

In order to prove our valuation formula we need the following two lemmas which relate the payoff conditions to multivariate normal distributions.

Lemma 5.4.1 *Let* $\mathbf{b}, \mathbf{q} \in \mathbb{R}^d$ *and* $\mathbf{A} \in \mathbb{R}^{d\times d}$ *with full rank, then*

$$\int_{\mathbf{Ax}\geq\mathbf{b}} e^{\mathbf{q}^T\mathbf{x}}\varphi(\mathbf{x})d\mathbf{x} = e^{\frac{1}{2}\mathbf{q}^T\mathbf{q}}\Phi(\mathbf{A}\mathbf{A}^T, \mathbf{A}\mathbf{q}-\mathbf{b}). \tag{5.22}$$

We use $\int_{\mathbf{Ax}\geq\mathbf{b}}$ *as abbreviation for the integration over the set* $\{\mathbf{x}\in\mathbb{R}^d : \mathbf{Ax}\geq\mathbf{b}\}$.

Proof: A straightforward computation shows

$$e^{\mathbf{q}^T\mathbf{x}}\varphi(\mathbf{x}) = e^{\frac{1}{2}\mathbf{q}^T\mathbf{q}}\varphi_{\mathbf{q},\mathbf{I}}(\mathbf{x}) \tag{5.23}$$

for all $\mathbf{x} \in \mathbb{R}^d$. Using the substitution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} + \mathbf{q}$ we obtain

$$\begin{aligned}\int_{\mathbf{Ax}\geq\mathbf{b}} e^{\mathbf{q}^T\mathbf{x}}\varphi(\mathbf{x})d\mathbf{x} &= e^{\frac{1}{2}\mathbf{q}^T\mathbf{q}}\int_{\mathbf{Ax}\geq\mathbf{b}} \varphi_{\mathbf{q},\mathbf{I}}(\mathbf{x})d\mathbf{x}\\ &= e^{\frac{1}{2}\mathbf{q}^T\mathbf{q}}\int_{\mathbf{y}\geq\mathbf{b}-\mathbf{Aq}} \varphi_{0,\mathbf{AA^T}}(\mathbf{y})\ d\mathbf{y}\end{aligned} \tag{5.24}$$

and thus the assertion. □

For the second Lemma, we first need to define a comparison relation $\geq_\mathbf{R}$ of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ with respect to the ranking $\mathbf{R}$:

$$\mathbf{x} \geq_\mathbf{R} \mathbf{y} \ :\Leftrightarrow\ R_i(x_i - y_i) \geq 0 \ \text{ for } 1 \leq i \leq n. \tag{5.25}$$

Thus, the comparison relation $\geq_\mathbf{R}$ is the usual component-wise comparison where the direction depends on the sign of the corresponding entry of the ranking vector $\mathbf{R}$.

Lemma 5.4.2 *We have* $\mathbf{Rank}(\mathbf{S}) = \mathbf{R}$ *exactly if* $\mathbf{AX} \geq_\mathbf{R} \mathbf{b}$ *with*

$$\mathbf{A} := \sqrt{T}\begin{pmatrix} \sigma_{11} & \dots & \sigma_{1d} \\ \sigma_{11}-\sigma_{21} & \dots & \sigma_{1d}-\sigma_{2d} \\ \vdots & & \vdots \\ \sigma_{11}-\sigma_{n1} & \dots & \sigma_{1d}-\sigma_{nd}\end{pmatrix} \ \textit{and}\ \mathbf{b} := \begin{pmatrix} \ln\frac{K}{S_1(0)} - rT + \bar\sigma_1 \\ \bar\sigma_1 - \bar\sigma_2 \\ \vdots \\ \bar\sigma_1 - \bar\sigma_n\end{pmatrix} \tag{5.26}$$

where $\mathbf{A} \in \mathbb{R}^{n\times d}$, $\mathbf{X} \in \mathbb{R}^d$ *and* $\mathbf{b} \in \mathbb{R}^n$.

Proof: Using (3.15) we see that $\text{Rank}_1 = +$ is equivalent to

$$S_1(T) \geq K \quad \Longleftrightarrow \quad \sqrt{T}\sum_{j=1}^{d} \sigma_{1j}X_j \geq \ln\frac{K}{S_1(0)} - rT + \bar{\sigma}_1 \tag{5.27}$$

which yields the first row of the system $\mathbf{AX} \geq_{\mathbf{R}} \mathbf{b}$. Moreover, for $i = 2, \ldots, n$ the outperformance criterion $\text{Rank}_i = +$ can be written as

$$\frac{S_1(T)}{S_1(0)} \geq \frac{S_i(T)}{S_i(0)} \quad \Longleftrightarrow \quad \sqrt{T}\sum_{j=1}^{d}(\sigma_{1j} - \sigma_{ij})X_j \geq \bar{\sigma}_1 - \bar{\sigma}_i \tag{5.28}$$

which yields rows 2 to n of the system. □

Now we can state the following pricing formula which, in a slightly more special setting, is originally due to Korn [80].

Theorem 5.4.3 (Valuation formula for Performance-Dependent Options) *The price of a performance-dependent option with payoff (2.17) is for the full Black-Scholes model given by*

$$V(\mathbf{S}, 0) = \sum_{\mathbf{R}\in\{+,-\}^n} a_{\mathbf{R}} \left(S_1(0)\, \Phi(\mathbf{A_R A_R^{\mathit{T}}}, -\mathbf{d_R}) - e^{-rT}\, K\Phi(\mathbf{A_R A_R^{\mathit{T}}}, -\mathbf{b_R})\right) \tag{5.29}$$

where the vectors $\mathbf{b_R}$, $\mathbf{d_R}$ and the matrix $\mathbf{A_R}$ are defined by $(\mathbf{b_R})_i := R_i\mathbf{b}_i$, $(\mathbf{d_R})_i := R_i\mathbf{d}_i$ and $(\mathbf{A_R})_{ij} := \mathbf{R}_i\mathbf{A}_{ij}$. Thereby, $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$ are defined as in Lemma 5.4.2 and the vector $\mathbf{d} \in \mathbb{R}^n$ is defined by $\mathbf{d} := \mathbf{b} - \sqrt{T}\mathbf{A}\sigma_1$ with σ_1^T being the first row of the volatility matrix σ.

Proof: The characteristic function $\chi_{\mathbf{R}}(\mathbf{S})$ in Theorem 4.3.4 can be eliminated using Lemma 5.4.2 and we get

$$V(\mathbf{S}, 0) = e^{-rT} \sum_{\mathbf{R}\in\{+,-\}^n} a_{\mathbf{R}} \int_{\mathbf{Ax}\geq_{\mathbf{R}}\mathbf{b}} (S_1(T) - K)\varphi(\mathbf{x})d\mathbf{x}. \tag{5.30}$$

By (3.15), the integral term can be written as

$$S_1(0)e^{rT-\bar{\sigma}_1} \int_{\mathbf{Ax}\geq_{\mathbf{R}}\mathbf{b}} e^{\sqrt{T}\sigma_1^T\mathbf{x}}\varphi(\mathbf{x})d\mathbf{x} - K\int_{\mathbf{Ax}\geq_{\mathbf{R}}\mathbf{b}} \varphi(\mathbf{x})d\mathbf{x}. \tag{5.31}$$

Application of Lemma 5.4.1 with $\mathbf{q} = \sqrt{T}\sigma_\mathbf{1}$ shows that the first integral equals

$$e^{\frac{1}{2}\mathbf{q}^T\mathbf{q}}\int_{\mathbf{y}\geq_{\mathbf{R}}\mathbf{b}-\mathbf{Aq}} \varphi_{0,\mathbf{AA^T}}(\mathbf{y})\; d\mathbf{y} = e^{\bar{\sigma}_1}\int_{\mathbf{y}\geq\mathbf{d_R}} \varphi_{0,\mathbf{A_R A_R^T}}(\mathbf{y})\; d\mathbf{y} = e^{\bar{\sigma}_1}\Phi(\mathbf{A_R A_R^{\mathit{T}}}, -\mathbf{d_R}). \tag{5.32}$$

By a further application of Lemma 5.4.1 with $\mathbf{q} = \mathbf{0}$ we obtain that the second integral equals $K\Phi(\mathbf{A_R A_R^{\mathit{T}}}, -\mathbf{b_R})$ and thus the assertion holds. □

Note that this decomposition not only provides the option price as a sum of normal distributions but can also be used to show which rankings appear with which probabilities under the model assumptions.

5.4.2 Reduced Model Valuation Formula

The pricing formula in Theorem 5.4.3 allows a stable and efficient valuation of performance-dependent options in the case of moderate-sized benchmarks. For a large number n of benchmark assets, one is, however, confronted with the following problems:

- In total, 2^n rankings have to be considered and thus a with n exponentially growing number of cumulative normal distributions has to be computed.
- For each normal distribution, an n-dimensional integration problem has to be solved which gets increasingly more difficult with rising n.
- In larger benchmarks, stock prices are typically highly correlated. As a consequence, some of the eigenvalues of the covariance matrix σ will be very small which makes the integration problems ill-conditioned.
- There is a large number $(n(n-1)/2)$ of free model parameters in the volatility matrix which are difficult to estimate robustly for large n.

In conclusion, the pricing formula in Theorem 5.4.3 can only be applied to small benchmarks, although it is very useful in this case. In this section, we aim to derive a similar pricing formula for reduced models which incorporate less processes than companies ($d < n$). This way, substantially fewer rankings have to be considered and much lower-dimensional integrals have to be computed which allows the pricing of performance-dependent options even for large benchmarks.

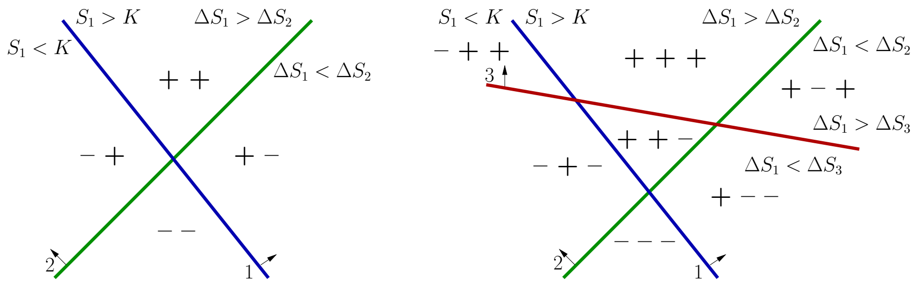
Geometric view

Lemma 5.4.2 and thus representation (5.30) of the option price remains also valid in the reduced model. Note, however, that $\mathbf{A}$ is now an $(n \times d)$-matrix which prevents the direct application of Lemma 5.4.1. At this point, a geometrical point of view is advantageous to illustrate the effect of performance comparisons in the reduced model.

The matrix $\mathbf{A}$ and the vector $\mathbf{b}$ define a set of n hyperplanes in the space $\mathbb{R}^d$. The dissection of $\mathbb{R}^d$ into different domains or cells is called an hyperplane arrangement and denoted by $\mathcal{A} = \mathcal{A}_{n,d}$. Each cell in the hyperplane arrangement $\mathcal{A}$ is a (possibly open) polyhedron P which is uniquely represented by a ranking vector $\mathbf{R} \in \{+,-\}^n$. Each element of the ranking vector indicates on which side of the corresponding hyperplane the polyhedral cell is located. We thus have the representation of the polyhedron as the set

$$P = \left\{ \mathbf{x} \in \mathbb{R}^d : \mathbf{A}\mathbf{x} \geq_{\mathbf{R}} \mathbf{b} \right\}. \tag{5.33}$$

Figure 5.1 illustrates two two-dimensional hyperplane arrangements, one for a full model with two assets and one for a reduced model with three assets. We see that in the reduced



$S_1 > K$
$S_1 < K$
$\Delta S_1 > \Delta S_2$
$\Delta S_1 < \Delta S_2$
$+\,+$
$-\,+$
$+\,-$
$-\,-$
2
1
$S_1 < K$
$S_1 > K$
$\Delta S_1 > \Delta S_2$
$\Delta S_1 < \Delta S_2$
$-\,+\,+$
$+\,+\,+$
3
$+\,-\,+$
$\Delta S_1 > \Delta S_3$
$+\,+\,-$
$-\,+\,-$
$\Delta S_1 < \Delta S_3$
$+\,-\,-$
$-\,-\,-$
2
1

Figure 5.1: Polyhedral cells and ranking vectors for two hyperplane arrangements with $d = 2, n = 2$ (left) and $d = 2, n = 3$ (right).

model fewer than the expected $2^3 = 8$ polyhedral cells arise. Indeed, it can be shown, see, e.g., [25], that the number of cells $c_{n,d}$ of the hyperplane arrangement $\mathcal{A}$ is bounded from above by

$$c_{n,d} = \sum_{i=0}^{d} \binom{n}{d-i}. \tag{5.34}$$

To illustrate this effect, note that in a full model with 30 benchmark assets, 1.1 billion cells arise while in a reduced model with 30 benchmark assets whose prices are driven by $d = 5$ underlying processes only about 170 thousand cells appear.

By identifying all cells in the hyperplane arrangement, we can significantly reduce the number of integrals to be computed. This way, the representation (5.30) of the option price can be rewritten as

$$V(\mathbf{S}, 0) = e^{-rT} \sum_{P \in \mathcal{A}} a_{\mathbf{R}} \int_P (S_1(T) - K)\varphi(\mathbf{x}) d\mathbf{x}. \tag{5.35}$$

By integrating the payoff function over each cell of the hyperplane arrangement separately, the option value can be determined as a sum over all integral values weighted with the corresponding bonus factors. Note that only smooth integrands appear in this approach.

Tools From Computational Geometry

Two problems remain with formula (5.35), however. First, it is not easy to see which ranking vectors and corresponding polyhedra appear in the hyperplane arrangement and which do not. Second, the integration region is now a general polyhedron and, therefore, involved integration rules are required. To resolve these difficulties we need some more utilities from computational geometry summarized in the following two Lemmas.

To state the first Lemma we have to choose a set of linearly independent directions $\mathbf{e}_1, \ldots, \mathbf{e}_d \in \mathbb{R}^d$ to impose an order on all points in $\mathbb{R}^d$. We assume in the following that no hyperplane is parallel to any of the directions. Moreover, we assume that the hyperplane arrangement is non-degenerate which means that exactly d hyperplanes intersect

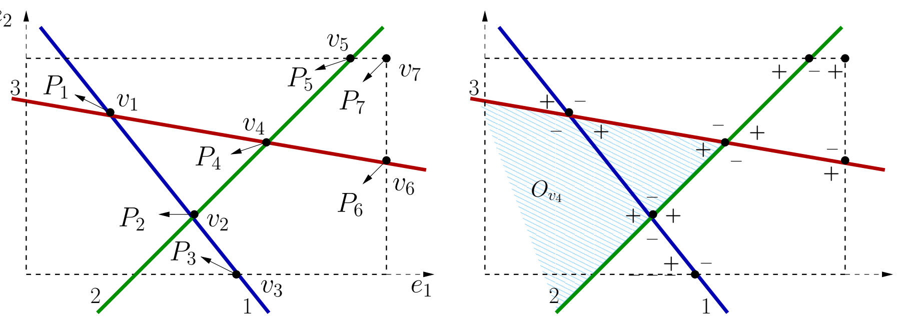


Figure 5.2: Illustration of the mapping between intersection points $\{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ and polyhedral cells $P_j := P_{\mathbf{v}_j}$ for the right arrangement from Figure 5.1 (left) and corresponding reflection signs $s_{\mathbf{v},\mathbf{w}}$ as well as the orthant $O_{\mathbf{v}_4}$ (right).

in each vertex. In the unlikely case that these conditions are not met, they can be ensured by slightly perturbing some of entries of the volatility matrix. Using the directions $\mathbf{e}_i$, an artificial bounding box which encompasses all vertices can be defined. This bounding box is only needed for the localization of the polyhedral cells in the following Lemma and does not implicate any approximation.

Lemma 5.4.4 *Let the set $\mathcal{V}$ consist of all interior vertices, of the largest intersection points of the hyperplanes with the bounding box and of the largest corner point of the bounding box. Furthermore, let $P_{\mathbf{v}} \in \mathcal{A}$ be the polyhedron which is adjacent to the vertex $\mathbf{v} \in \mathcal{V}$ and which contains no other vertex which is larger than $\mathbf{v}$ with respect to the direction vectors. Then the mapping $\mathbf{v} \mapsto P_{\mathbf{v}}$ is one-to-one and onto.*

The proof of Lemma 5.4.4 can be found in the next chapter. For the two dimensional example with three hyperplanes in Figure 5.1 the mapping between intersection points and polyhedral cells is illustrated in Figure 5.2 (left). Each vertex from the set $\mathcal{V} := \{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ is mapped to the polyhedral cell indicated by the corresponding arrow. Using Lemma 5.4.4, an easy to implement optimal order $O(c_{n,d})$ algorithm which enumerates all cells in an hyperplane arrangement can be constructed.

Note that by Lemma 5.4.4 each vertex $\mathbf{v} \in \mathcal{V}$ corresponds to a unique cell $P_{\mathbf{v}} \in \mathcal{A}$ and thus to a ranking vector $\mathbf{R}$. We can, therefore, also assign bonus factors to vertices by setting $a_{\mathbf{v}} := a_{\mathbf{R}}$. Next, we assign each vertex $\mathbf{v}$ an associated orthant $O_{\mathbf{v}}$. An orthant is defined as an open region in $\mathbb{R}^d$ which is bounded by $k \leq d$ hyperplanes. To find the orthant associated with the vertex $\mathbf{v}$, we look at k backward (with respect to the directions $\mathbf{e}_i$) points by moving $\mathbf{v}$ backwards on each of the k intersecting hyperplanes. The unique orthant which contains $\mathbf{v}$ and all backward points is denoted by $O_{\mathbf{v}}$.

For illustration, the orthant $O_{\mathbf{v}_4}$ is displayed in Figure 5.2 (right). Note that vertices which are located on the boundary correspond to orthants with $k < d$ intersecting hyperplanes.

For example, $O_{\mathbf{v}_3}$ is defined by all points which are below hyperplane one.

By definition, there exists a $(k \times d)$-submatrix $\mathbf{A}_\mathbf{v}$ of $\mathbf{A}$ and a k-subvector $\mathbf{b}_\mathbf{v}$ of $\mathbf{b}$ such that the orthant $O_\mathbf{v}$ can be characterized as the set

$$O_\mathbf{v} = \left\{ \mathbf{x} \in \mathbb{R}^d : \mathbf{A}_\mathbf{v}\mathbf{x} \geq_\mathbf{R} \mathbf{b}_\mathbf{v} \right\}, \tag{5.36}$$

where $\mathbf{R}$ is the ranking vector which corresponds to $\mathbf{v}$.

Furthermore, given two vertices $\mathbf{v}, \mathbf{w} \in \mathcal{V}$, we define the reflection sign $s_{\mathbf{v},\mathbf{w}} := (-1)^{r_{\mathbf{v},\mathbf{w}}}$ where $r_{\mathbf{v},\mathbf{w}}$ is the number of reflections on hyperplanes needed to map $O_\mathbf{w}$ onto $P_\mathbf{v}$. The reflection signs $s_{\mathbf{v},\mathbf{w}}$ with $\mathbf{v} \in \{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ and $\mathbf{w} \in P_\mathbf{v}$ arising in the two dimensional arrangement in Figure 5.1 (right) are displayed in Figure 5.2 (right). For instance, the three reflection signs in the cell $P_{\mathbf{v}_4}$ are given by $s_{\mathbf{v}_4,\mathbf{v}_1} = +$, $s_{\mathbf{v}_4,\mathbf{v}_2} = -$ and $s_{\mathbf{v}_4,\mathbf{v}_4} = +$. Finally, let $\mathcal{V}_\mathbf{v}$ denote the set of all vertices of the polyhedron $P_\mathbf{v}$.

Lemma 5.4.5 *It is possible to algebraically decompose any cell of a hyperplane arrangement into a signed sum of orthant cells by*

$$\chi(P_\mathbf{v}) = \sum_{\mathbf{w} \in \mathcal{V}_\mathbf{v}} s_{\mathbf{v},\mathbf{w}} \chi(O_\mathbf{w}) \tag{5.37}$$

where χ is the characteristic function of a set. Moreover, all cells of a hyperplane arrangement can be decomposed into a signed sum of orthants using exactly one orthant per cell.

The first part of Lemma 5.4.5 is originally due to Lawrence [85]. The second part follows from the one-to-one correspondence between orthants $O_\mathbf{v}$ and cells $P_\mathbf{v}$. It can be found in detail in the next chapter.

Note that such an orthant decomposition is not unique. A different decomposition of a polyhedron into a sum of orthants is, e.g., presented in [26].

Example 5.4.6 *To give an example, the decomposition of all cells within the hyperplane arrangement from Figure 5.2 is given by*

$$\begin{array}{rcl}
\chi(P_1) &=& \chi(O_1) \\
\chi(P_2) &=& \chi(O_2) - \chi(O_1) \\
\chi(P_3) &=& \chi(O_3) - \chi(O_2) \\
\chi(P_4) &=& \chi(O_4) - \chi(O_2) + \chi(O_1) \\
\chi(P_5) &=& \chi(O_5) - \chi(O_4) - \chi(O_1) \\
\chi(P_6) &=& \chi(O_6) - \chi(O_4) - \chi(O_3) + \chi(O_2) \\
\chi(P_7) &=& \chi(O_7) - \chi(O_6) - \chi(O_5) + \chi(O_4)
\end{array} \tag{5.38}$$

where we used the abbreviations $P_j := P_{\mathbf{v}_j}$ and $O_j := O_{\mathbf{v}_j}$.

Pricing Formula

Now, we are finally able to give a pricing formula for performance-dependent options also in the reduced model case.

Theorem 5.4.7 (Valuation Formula for Performance-Dependent Options) *The price of a performance-dependent option with payoff (2.17) is for the reduced Black-Scholes model in the case $d \leq n$ given by*

$$V(\mathbf{S},0) = \sum_{\mathbf{v}\in\mathcal{V}} c_{\mathbf{v}} \left(S_1(0)\Phi(\mathbf{A}_{\mathbf{v}}\mathbf{A}_{\mathbf{v}}^T, -\mathbf{d}_{\mathbf{v}}) - e^{-rT}K\Phi(\mathbf{A}_{\mathbf{v}}\mathbf{A}_{\mathbf{v}}^T, -\mathbf{b}_{\mathbf{v}})\right) \tag{5.39}$$

with $\mathbf{A}_{\mathbf{v}}, \mathbf{b}_{\mathbf{v}}$ as in (5.36) and $\mathbf{d}_{\mathbf{v}}$ being the corresponding subvector of $\mathbf{d}$. The weights $c_{\mathbf{v}}$ are given by

$$c_{\mathbf{v}} := \sum_{\mathbf{w}\in\mathcal{V}:\ \mathbf{v}\in P_{\mathbf{w}}} s_{\mathbf{v},\mathbf{w}} a_{\mathbf{w}}. \tag{5.40}$$

Proof: By Lemma 5.4.4 we see that the integral representation (5.35) is equivalent to a summation over all vertices $\mathbf{v} \in \mathcal{V}$, i.e.

$$V(\mathbf{S},0) = e^{-rT} \sum_{\mathbf{v}\in\mathcal{V}} a_{\mathbf{v}} \int_{P_{\mathbf{v}}} (S_1(T) - K)\varphi(\mathbf{x})d\mathbf{x}. \tag{5.41}$$

By Lemma 5.4.5 we can decompose the polyhedron $P_{\mathbf{v}}$ into a signed sum of orthants and obtain

$$V(\mathbf{S},0) = e^{-rT} \sum_{\mathbf{v}\in\mathcal{V}} a_{\mathbf{v}} \sum_{\mathbf{w}\in\mathcal{V}_{\mathbf{v}}} s_{\mathbf{v},\mathbf{w}} \int_{O_{\mathbf{w}}} (S_1(T) - K)\varphi(\mathbf{x})d\mathbf{x}. \tag{5.42}$$

By the second part of Lemma 5.4.5 we know that only $c_{n,d}$ different integrals appear in the above sum. Rearranging the terms leads to

$$V(\mathbf{S},0) = e^{-rT} \sum_{\mathbf{v}\in\mathcal{V}} c_{\mathbf{v}} \int_{O_{\mathbf{v}}} (S_1(T) - K)\varphi(\mathbf{x})d\mathbf{x}. \tag{5.43}$$

Since now the integration domains $O_{\mathbf{v}}$ are orthants, Lemma 5.4.1 can be applied exactly as in the proof of Theorem 5.4.3 which finally implies the Theorem. □

By the non-degeneracy condition there are at most 2^d cells adjacent to each vertex which bounds the number of terms in the definition of $c_{\mathbf{v}}$. Moreover, the number of vertices in $\mathcal{V}$ equals $c_{n,d}$ which yields the number of integrals which have to be computed in the worst case.

Example 5.4.8 *Consider the bonus scheme from Example 2.4.5 with $n = 3$, $d = 2$ and the hyperplane arrangement from Figure 5.2. Then, the bonus factors $a_j := a_{\mathbf{v}_j}$ are given by*

$$a_1 = 0,\ a_2 = 0,\ a_3 = 0,\ a_4 = \frac{1}{2},\ a_5 = 1,\ a_6 = 0,\ a_7 = \frac{1}{2}. \tag{5.44}$$

Following the steps in the proof of Theorem 5.4.7 and employing the decomposition from Example 5.4.6 we see that the price of this option satisfies

$$\begin{aligned} V(\mathbf{S},0) &= e^{-rT}\left(\frac{1}{2}I(P_4)+I(P_5)+\frac{1}{2}I(P_7)\right) \\ &= e^{-rT}\left(-\frac{1}{2}I(O_1)-\frac{1}{2}I(O_2)+\frac{1}{2}I(O_5)-\frac{1}{2}I(O_6)+\frac{1}{2}I(O_7)\right) \end{aligned} \tag{5.45}$$

where we define

$$I(B) := \int_B (S_1(T)-K)\varphi(\mathbf{x})d\mathbf{x}. \tag{5.46}$$

Special Cases

Let us first remark that, if the payoff function has a special structure, many weights $c_{\mathbf{v}}$ are zero in the formula from Theorem 5.4.7. This way, the corresponding normal distributions do not have to be computed. This is, for example, true for the outperformance option of Example 2.4.6.

In addition, if the vertex $\mathbf{v}$ is located on the artificial boundary, see for example vertex $\mathbf{v}_3$ in Figure 5.2, the corresponding orthant is defined by $k < d$ intersecting hyperplanes. As a consequence, only a k-dimensional normal distribution instead of a d-dimensional one has to be computed. Consider, for example, a bonus scheme which is defined by the bonus factors

$$a_{\mathbf{R}} = \begin{cases} \sum\limits_{\{i:R_i=+\}} \bar{a}_i & \text{if } \mathbf{R}_1 = + \\ 0 & else \end{cases} \tag{5.47}$$

for some given $\bar{a}_i \in \mathbb{R}$, where the sum goes over all $i \in \{2,\ldots,n\}$ where $R_i = +$. Example 2.4.5 is a special case of such a scheme with $\bar{a}_i \equiv 1/(n-1)$. The pricing formula for such a scheme only contains vertices which are located on at least $d-2$ boundary hyperplanes. Thus, independently of d and n, at most two-dimensional normal distributions have to be evaluated. Moreover, the number of two-dimensional normal distributions is bounded by $n-1$. This behaviour is most easily understood if the payoff function of the bonus scheme (5.47) is rewritten in the equivalent form

$$V(\mathbf{S},T) = \sum_{i=2}^{n} \bar{a}_i\,(S_1(T)-K)^+\,\chi_{\Delta S_1(T)\geq \Delta S_i(T)} \tag{5.48}$$

which shows that only the two-dimensional joint distributions of the random variables $S_1(T)$ and $S_i(T)$ are required for $i = 2,\ldots,n$. Equipping the basic scheme (5.47) by outperformance conditions one can see that each additional outperformance condition increases the maximum dimension of normal distributions arising in our pricing formula by one up to the nominal dimension d, see the Examples in section 9.4.2.

Note that these special cases are automatically recognized by our algorithm and only the minimum number of integrals with the corresponding minimal dimensions are computed.

Greeks

An additional advantage of the formulas from Theorem 5.4.3 and 5.4.7 compared to a standard Monte-Carlo pricing approach is given by the fact that option price sensitivities can be obtained by analytical differentiation. To give an example, the Greek letter Delta $\partial V/\partial S_1$ satisfies

$$\frac{\partial V}{\partial S_1} = \sum_{\mathbf{v}\in H_1} c_{\mathbf{v}} \left(\Phi(\mathbf{A_v}\mathbf{A_v}^T, -\mathbf{d_v}) + \Phi(\mathbf{C_v}, -\mathbf{e_v}) - e^{-rT}\frac{K}{S_1}\Phi(\mathbf{C_v}, -\mathbf{e_v}) \right)$$

where H_1 denotes the subset of $\mathcal{V}$ containing all vertices on hyperplane one. The matrix $\mathbf{C_v}$ is defined by $(\mathbf{C_v})_{i,j} := (\mathbf{A_v}\mathbf{A_v}^T)_{i+1,j+1}$ for $i, j = 1, \ldots, d-1$. and the vectors $\mathbf{e_v}$ and $\mathbf{f_v}$ are given by $\mathbf{e_v} := ((b_v)_1 + (b_v)_2, (b_v)_3, \ldots, (b_v)_d)^T$ and $\mathbf{f_v} := ((d_v)_1 + (d_v)_2, (d_v)_3, \ldots, (d_v)_d)^T$. The computation of the Greek letters can thus be integrated in the valuation algorithm without much additional effort. Instead, employing standard approaches the derivatives can only be approximated by finite differences which usually results in a much slower convergence rate.

Chapter 6

Hyperplane Arrangements

6.1 Introduction

In this chapter we take a closer look at hyperplane arrangements which played an important role for the derivation of valuation formulas for performance-dependent options. Thereby, we use a novel paradigm, a one-to-one correspondence between a certain set of intersection points and cells. This paradigm allows the development of very efficient algorithms. The first algorithm enumerates all cells in a hyperplane arrangement. The second one performs an orthant decomposition of a hyperplane arrangement using exactly one orthant per cell. Both algorithms are not difficult to implement and run in optimal order complexity.

Hyperplane arrangements are one of the most fundamental concepts in geometry and topology. They are the structure defined by a set of n hyperplanes in d-dimensional space. This structure becomes interesting when the number of hyperplanes is larger than the space dimension. Its topological properties have been studied thoroughly in many publications, for a summary see, e.g., [25, 97].

Hyperplane arrangements have not only been investigated from a theoretical point of view but have also been used as a computational tool. Applications include polyhedral volume computation [11, 85], integration over polyhedral domains [26], path planning in robotics [116], pattern recognition [54], higher order Voronoi diagrams [28] and computational finance [49].

Especially if the number of hyperplanes or the space dimension is large, algorithms which can handle hyperplane arrangements efficiently are difficult to realize. Although many approaches exist, none of them are fully satisfactory (see [11] for the polyhedral volume calculation problem) and the collection of available software is limited [98]. Furthermore, the cells arising in the hyperplane arrangement are general convex polyhedra which can have many faces and vertices. From a computational point of view, in many applications one would like to deal with simple building blocks such as hypercubes ($2d$ faces), simplices

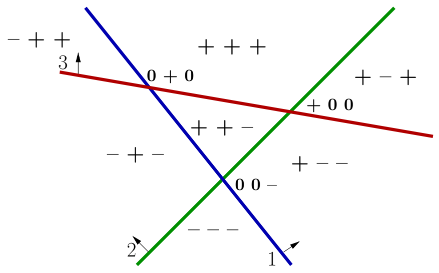


Figure 6.1: Example hyperplane arrangement $\mathcal{A}_{3,2}$. Shown are the position vectors of the 7 cells and the 3 vertices.

($d+1$ faces) or orthants (d or less faces) and not general polyhedra. Thereby, the number of building blocks should be as small as possible.

In this chapter, we address these two issues. First, we illustrate an efficient and easy to implement algorithm for the enumeration of all cells in a hyperplane arrangement. This algorithm uses a one-to-one correspondence of certain (interior and boundary) intersection points and cells. Second, we propose a novel method for the algebraic decomposition of a hyperplane arrangement into orthants. This method directly uses Lawrence's signed decomposition lemma. The number of required orthants is minimal in this approach since exactly one orthant per cell is used. Both algorithms run in optimal order complexity.

6.2 Definitions

The linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and a vector $\mathbf{b} \in \mathbb{R}^n$ defines a set of n hyperplanes

$$H_i := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{a}_i \cdot \mathbf{x} = b_i\} \tag{6.1}$$

in the space $\mathbb{R}^d$ where $\mathbf{a}_i$ denotes the i-th row of the matrix $\mathbf{A}$. The dissection of the space into different domains or cells is called a hyperplane arrangement and is denoted by $\mathcal{A}_{n,d}$. Each cell in the hyperplane arrangement $\mathcal{A}_{n,d}$ is a convex and possibly unbounded polyhedron P which is uniquely represented by a *position vector* $\mathbf{p} \in \{+,-\}^n$. Each element of the position vector indicates on which side of the corresponding hyperplane the polyhedral cell is located. The position vectors of an example hyperplane arrangement with three planes (lines) in dimension two are shown in Figure 6.1.

Moreover, each face and each vertex of a hyperplane arrangement can be characterized by a position vector $\mathbf{p} \in \{+,0,-\}^n$. If the entry p_i is zero, then the corresponding face or vertex is located on hyperplane i. In Figure 5.1, also the three arising vertices are labeled with their position vectors. We denote the set of all vertices in the hyperplane arrangement $\mathcal{A}_{n,d}$ by $\mathcal{V}_{n,d}$.

A hyperplane arrangement is called *non-degenerate* if any d hyperplanes intersect in a unique vertex and if any $d+1$ hyperplanes possess no common points. This way, the

n	$2^n = c_{n,n}$	$c_{n,20}$	$c_{n,10}$	$c_{n,5}$	$c_{n,3}$
2	4	-	-	-	-
4	16	-	-	-	15
8	256	-	-	219	93
16	65536	-	58651	6885	697
30	1.1e9	1.0e9	5.3e7	1.7e5	4526

Table 6.1: Number of cells $c_{n,d}$ in a non-degenerate hyperplane arrangement for varying n and d.

codimension of a face is given by the number of zeroes in the position vector. In particular, a vertex is characterized by d zeroes.

In a non-degenerate hyperplane arrangement there are exactly $\binom{n}{d}$ vertices. Furthermore, the number $c_{n,d}$ of cells is given by

$$c_{n,d} = \sum_{i=0}^{d} \binom{n}{d-i}, \tag{6.2}$$

see [25]. Note that non-degenerate arrangements maximize the number of vertices and cells. In Table 6.1 we show the number of cells in a non-degenerate hyperplane arrangement for various n and d. For large n and small d, we have $c_{n,d} \ll 2^n$. For constant d, the number of vertices and cells grows like $O(n^d)$ and thus constitute worst case examples for algorithms whose complexity is increasing with the number of cells.

In the following, we always assume that the non-degeneracy condition is satisfied. In the case this condition is not met, it can be ensured by slightly perturbing some entries of the matrix $\mathbf{A}$. For complexity reasons, this approach might not be desirable especially for highly degenerate arrangements. We are positive, though, that many of the following concepts and algorithms can be extended to handle degeneracies efficiently, although we do not address them in this here.

6.3 Enumeration

The enumeration of all existing cells in a hyperplane arrangement is not an easy task and is well investigated in the literature. In [27], an incremental approach based on the zone theorem is proposed to construct hyperplane arrangements in optimal $O(n^d)$ operations. The algorithm not only enumerates all cells but also constructs the complete incidence graph. There are no running times reported, however, and the implementation of the algorithm is complicated and demanding. In [115], an output-sensitive algorithm based on reverse search [2] was developed. Its essential component is the solution of several linear optimization problems for which existing software packages can be used. This way, the implementation of the algorithm is much easier than for algorithm [27]. Its complexity is given by $O(d \cdot lp(n,d) \cdot n^d)$ where $lp(n,d)$ denotes the time which is needed to solve an

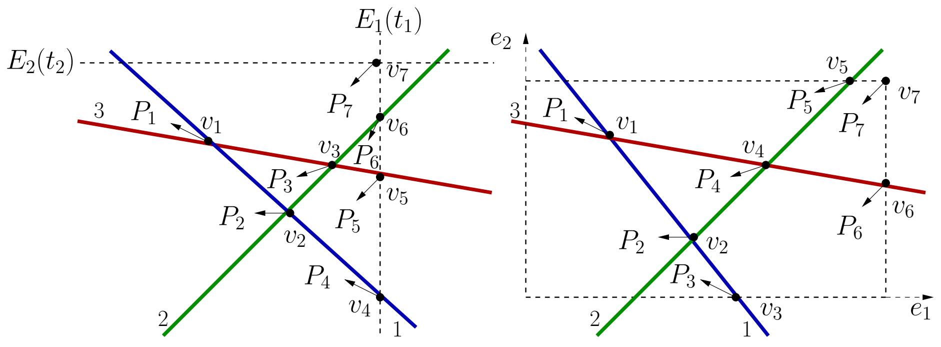


Figure 6.2: Two mappings between intersection points $\{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ and cells $\{P_1, \ldots, P_7\}$ for the arrangement from Figure 5.1.

$n \times d$ linear program.

6.3.1 Simple Cell Enumeration Algorithm

To illustrate the main problems which arise in cell enumeration, let us start with a very simple algorithm. It is based on the property that a cell P exists in $\mathcal{A}_{n,d}$ if and only if there exists a vertex $\mathbf{v}$ with a matching position code. Two position vectors $\mathbf{p}$ and $\mathbf{q}$ are said to *match* if

$$p_i = q_i \text{ or } p_i = 0 \text{ or } q_i = 0 \text{ for all } i = 1, \ldots, n \tag{6.3}$$

holds.

In Figure 5.1, we see that each cell matches all its vertices and each vertex matches all its adjacent cells.

Algorithm 6.3.1 first determines all vertices of the hyperplane arrangement by intersecting all possible subsets of d hyperplanes from the n given hyperplanes. The computation of each vertex position requires the solution of a $d \times d$ linear system which in practice requires $O(d^3)$ operations. The position code of each vertex contains 0 for all hyperplanes in the subset and $+$ or $-$ for the other $n - d$ hyperplanes depending on which side of the hyperplane the vertex is on. Each sign can be determined from an inner product which requires $O(d)$ operations. The algorithm then runs over all vertices in the arrangement and determines the position codes of the 2^d adjacent cells of each vertex. The position code of each adjacent cell has either $+$ or $-$ for each 0 in the position code of the vertex. In order to avoid duplicates, already found cells have to be stored, e.g., in a hash table for which a fast $O(n)$ access is possible. An implementation of Algorithm 6.3.1 typically requires less than 50 lines of code.

The computational complexity of Algorithm 6.3.1 is given by

$$O\left(\binom{n}{d}\left(d^3 + (n-d)\,d + n\,2^d\right)\right). \tag{6.4}$$

For fixed d, this complexity is of order $O(n^{d+1})$ which can be seen as optimal since there are

$O(n^d)$ cells and already outputting the position code of each cell requires $O(n)$ operations. For variable d, the algorithm is not optimal, though. It also requires a significant amount ($O(n^d)$) of storage. Nevertheless, it is surprisingly competitive and was in our experiments much faster than the output-sensitive algorithm [115] for many practical relevant choices of d and n (see section 5). Without significant additional costs, it also provides the number of vertices and the bounding hyperplanes of each cell.

Algorithm 6.3.1 (Simple Cell Enumeration Algorithm)

```
Input: Hyperplane arrangement $\mathcal{A}_{n,d}$ defined by $\mathbf{A}$ and $\mathbf{b}$.
Output: List of the position vectors $\mathbf{p}$ of all cells in the arrangement.
 a) for each vertex $\mathbf{v} \in \mathcal{A}_{n,d}$
    a.1) compute the position vector $\mathbf{p_v}$
    a.2) find the position codes $\mathbf{p}_P$ of all adjacent cells $P$ to $\mathbf{v}$
    a.3) for each adjacent cell $P$ which has not been stored
            store and output $P$
```

6.3.2 Correspondence between Intersection Points and Cells

The problem with Algorithm 6.3.1 is that each cell is found many times, once for each of its vertices. The running time and space complexities would be much better if each cell could be assigned to a unique vertex and vice versa and thus only found once. We now aim to establish such a correspondence.

To this end, we select a set of linearly independent *directions* $\mathbf{e}_1, \ldots, \mathbf{e}_d \in \mathbb{R}^d$. Using these directions we impose an order on all points in $\mathbb{R}^d$. A point is smaller than another point if it lies behind it with respect to direction $\mathbf{e}_1$. If both points are on equal level with respect to $\mathbf{e}_1$, their positions are compared with respect to $\mathbf{e}_2$, and so on. We assume in the following that the directions are chosen such that no hyperplane is parallel to any of the directions. For efficiency and simplicity, the d unit vectors are selected as directions if possible. Now, we can define a mapping between vertices from a given set $\mathcal{V}$ (which will be determined later) and cells as follows.

Definition 6.3.2 *The mapping $\pi : \mathbf{v} \mapsto P$ assigns a vertex $\mathbf{v} \in \mathcal{V}$ the cell P which is adjacent to $\mathbf{v}$ and which contains no other vertex from $\mathcal{V}$ which is larger than $\mathbf{v}$.*

In Figure 6.2, left we see this mapping illustrated for the three vertices from the vertex set $\mathcal{V}_{n,d}$ labeled v_1, v_2 and v_3 for the hyperplane arrangement from Figure 5.1. The mapping π captures only a subset $\{P_1, P_2, P_3\}$ of the cells in the arrangement. This is to be expected since the number of vertices in $\mathcal{V}_{n,d}$ is significantly smaller than the number of cells. We therefore need some additional vertices which we now define as the intersection points of the hyperplanes in the arrangement with some additional hyperplanes.

Definition 6.3.3 *Let the d hyperplanes $E_i(t)$, $1 \leq i \leq d$, be defined by the equations $\mathbf{e}_i \cdot \mathbf{x} = t$. The intersection of the hyperplane arrangement $\mathcal{A}_{n,d}$ with the first i hyperplanes $E_1(t_1), \ldots, E_i(t_i)$ is denoted by $\mathcal{A}_{n,d-i}$. Thereby, t_i is chosen so large such that all vertices of the hyperplane arrangement $\mathcal{A}_{n,d-i+1}$ are below $E_i(t_i)$.*

In Figure 5.2, left, we see two possible hyperplanes $E_1(t_1)$ and $E_2(t_2)$ indicated by dashed lines. Note that the hyperplanes E_i do not imply any approximation but are only used in a symbolic way. With these definitions, we can now establish a one-to-one correspondence between the cells in $\mathcal{A}_{n,d}$ and a special set of intersection points.

Lemma 6.3.4 *Let the set $\bar{\mathcal{V}}_{n,d}$ consist of the intersection points of any k different hyperplanes H_i, $i \in \{1, \ldots, n\}$, with the first $d-k$ hyperplanes $E_j(t_j)$, $j = 1, \ldots, d-k$, where $k = 0, \ldots, d$. Then, the mapping $\pi : \bar{\mathcal{V}}_{n,d} \mapsto \mathcal{A}_{n,d}$ is one-to-one and onto.*

Proof: The proof uses a sweep plane argument and induction over d similar to the proof of Lemma 1.2 in [25].

Without loss of generality we assume that no two vertices of the arrangement share the same x_1-coordinate. First, we sweep the hyperplane $E_1(t)$ through the arrangement by running t from $-\infty$ to t_1. Each time $E_1(t)$ passes through a vertex $\mathbf{v}$, one more cell P comes to lie below $E_1(t)$ and we have $\pi(\mathbf{v}) = P$. When we arrive at t_1, all vertices in $\mathcal{V}_{n,d}$ and $\binom{n}{d}$ cells lie behind $E_1(t_1)$ and we have a one-to-one correspondence between these cells and vertices.

Now, all remaining cells are intersected by $E_1(t_1)$. The intersection of $\mathcal{A}_{n,d}$ with $E_1(t_1)$ defines the $(d-1)$-dimensional arrangement $\mathcal{A}_{n,d-1}$. This arrangement is now swept by the second hyperplane $E_2(t)$ with t running from $-\infty$ to t_2 which results in a one-to-one correspondence of $\binom{n}{d-1}$ cells with the set of all vertices in $\mathcal{A}_{n,d-1}$ denoted by $\mathcal{V}_{n,d-1}$. The mapping π then maps the vertices in $\mathcal{V}_{n,d-1}$ to a subset of the cells in $\mathcal{A}_{n,d-1}$. These cells can in turn be identified with the corresponding cells in $\mathcal{A}_{n,d}$.

Proceeding this argument inductively with the hyperplanes $E_3, \ldots, E_d$, we obtain a one-to-one correspondence of all cells in $\mathcal{A}_{n,d}$ with the set

$$\bar{\mathcal{V}}_{n,d} = \bigcup_{k=0}^{d} \mathcal{V}_{n,k} \tag{6.5}$$

where $\mathcal{V}_{n,k}$ is the set of all intersection points of k different hyperplanes H_i of $\mathcal{A}_{n,d}$ with the first $d-k$ hyperplanes $E_j(t_j)$. □

Figure 5.2, left, shows all intersection points and their corresponding cells for the example arrangement of Figure 5.1. The intersection points form the ordered set $\bar{\mathcal{V}}_{3,2} := \{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ which is sorted with respect to the directions $\mathbf{e}_1$ and $\mathbf{e}_2$. The mapping between intersection points and the corresponding polyhedral cells is indicated by arrows. For example, v_5 is mapped to the cell below it since all the other vertices of this cell

(v_2, v_3, v_4) are smaller than v_5 while the cell above it contains one vertex (v_6) which is larger. It is now easy to see that the number of cells in $\mathcal{A}_{n,d}$ is given by $c_{n,d}$ which is the cardinality of the set $\bar{\mathcal{V}}_{n,d}$.

6.3.3 Intersection Points with a Box

In theory, using Lemma 6.3.4 it is possible to enumerate the cells of $\mathcal{A}_{n,d}$ by the determination of the set of vertices $\bar{\mathcal{V}}_{n,d}$. From a practical point of view, however, one is confronted with the problem that, depending on the specific arrangement, the coordinates t_i can easily become very large. This can go so far that they cannot be stored anymore using standard floating point arithmetic or intersection computations become numerically unstable. Since the choice of t_i depends on the choice of t_{i-1}, this problem becomes more and more severe with rising dimension d.

To circumvent this difficulty, we will now show that also an equivalent set $\tilde{\mathcal{V}}_{n,d}$ of intersection points can be used to find all cells in a hyperplane arrangement. All these intersection points are located inside or on the boundary of an artificial bounding box which encompasses all vertices of the hyperplane arrangement. Again, just like the hyperplanes E_i, this bounding box does not imply any approximation but is used symbolically.

Definition 6.3.5 *Let the bounding box C be defined by*

$$C := \bigcap_{i=1}^{d} I_i \quad \textit{where} \quad I_i := \{s_i \le \mathbf{e}_i \cdot \mathbf{x} \le t_i\} \tag{6.6}$$

and where s_i and t_i are chosen such that all vertices in $\mathcal{V}_{n,d}$ are located within C. Furthermore, we denote by $\mathcal{C}_k$, $k = 0, \ldots, d$, the sets of k-dimensional faces of C.

This way, $\mathcal{C}_0$ consists of all vertices, $\mathcal{C}_1$ of all edges and $\mathcal{C}_{d-1}$ of all sides of C while the set $\mathcal{C}_d$ has C itself as its only element. If the $\mathbf{e}_i$ are given by the d unit vectors, t_i can be chosen as the maximal x_i coordinate of all vertices in $\mathcal{V}_{n,d}$ plus ε and s_i as the corresponding minimal coordinate minus ε.

Now, we can use the intersection points of the hyperplanes with the bounding box C for the determination of the cells. We may only take the largest intersection points, though, which is constituted in the following Lemma.

Lemma 6.3.6 *Let the set $\tilde{\mathcal{V}}_{n,d}$ consist of the largest intersection points of any k different hyperplanes H_i, $i \in \{1, \ldots, n\}$, with $\mathcal{C}_{d-k}$ where $k = 0, \ldots, d$. Then, the mapping $\pi : \tilde{\mathcal{V}}_{n,d} \mapsto \mathcal{A}_{n,d}$ is one-to-one and onto.*

Proof: We show a one-to-one and onto mapping between $\bar{\mathcal{V}}_{n,d}$ and $\tilde{\mathcal{V}}_{n,d}$ which preserves π. The assertion then follows from Lemma 6.3.4. We know that each vertex $\mathbf{v} \in \bar{\mathcal{V}}_{n,k}$

corresponds to a cell $P = \pi(\mathbf{v})$. Let us denote by $\mathbf{w}$ the maximum intersection point from $\tilde{\mathcal{V}}_{n,d}$ located within or on the boundary of P. We can now define $\pi(\mathbf{w}) = P$ and thus we have $\pi(\mathbf{v}) = \pi(\mathbf{w})$. Since the maximum intersection point of a cell is unique in a non-degenerate arrangement, the assertion follows. $\square$

In Figure 5.2, right, we show the mapping of intersection points and cells for our two-dimensional example. We have seven maximal intersection points: three involve two hyperplanes ($k = 2$), three involve one hyperplane and a bounding box side ($k = 1$) and one involves two bounding box sides ($k = 0$). The intersection points form the ordered set $\tilde{\mathcal{V}}_{3,2} := \{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ (note that especially $v_3 < v_4$). The mapping between intersection points and the corresponding polyhedral cells is again indicated by arrows. Note that the order of cells is not the same as in Figure 5.2, left.

6.3.4 Cell Enumeration Algorithm

We can now describe a numerically stable algorithm which uses the correspondence of intersection points and cells in a hyperplane arrangement from Lemma 6.3.6. It is also well suited as a starting point for the decomposition of a hyperplane arrangement into orthants as discussed in section 6.4.

In the first step of Algorithm 6.3.7, all largest intersection points are computed. To this end, each set of k hyperplanes, $0 \leq k \leq d$, has to be intersected with a set of $d - k$ bounding box sides. The largest of these intersection points is added to the set $\tilde{\mathcal{V}}_{n,d}$. Now, for each of these intersection points $\mathbf{v}$, the corresponding polyhedral cell $P = \pi(\mathbf{v})$ has to be determined. Thereby, each of the d zeroes in the position vector $p_{\mathbf{v}}$ has to be replaced by the sign corresponding to P. To this end, in the second step of the algorithm, first the position vector $p_{\mathbf{v}}$ is computed (see Algorithm 6.3.1). Then, all d edges going through the vertex are determined. On each edge, the vertex is moved slightly backwards. The entry of the position vector of this backward point which corresponds to the hyperplane the edge is not part of yields the corresponding sign of the position vector of P. Once the vertex has been moved backwards on all edges, the position vector of the cell is complete.

We will now discuss the complexity of Algorithm 6.3.7. In the first step, all intersection points are determined. For $0 \leq k \leq d$ there are $\binom{n}{k}$ possible combinations of hyperplanes and $2^{d-k}\binom{d}{k}$ valid combinations of bounding box sides. Since the computation of each intersection costs $O(d^3)$ operations, the total complexity of this step is

$$O\left(\sum_{k=0}^{d} \binom{n}{k} 2^{d-k} \binom{d}{k} d^3\right), \tag{6.7}$$

which for fixed d equals $O(n^d)$. In the second step of the algorithm, the determination of the position vector costs $O(nd)$ operations for the inner products. Then, for each edge a $d \times d$ system has to be solved for the computation of the backward point and $O(nd)$ operations have to be carried out to determine its position vector. The complexity of the second step is therefore given by

$$O(c_{n,d}(nd + d(d^3 + nd))) = O(c_{n,d}(d^4 + nd^2)). \tag{6.8}$$

Since $c_{n,d} = O(n^d)$ for fixed d, the total complexity of Algorithm 6.3.7 is given by

$$O(n^{d+1}). \tag{6.9}$$

which is again optimal. The order constant is significantly smaller than for Algorithm 6.3.1, though.

Let us remark that not all intersections with the bounding box faces have to be computed to determine the maximum intersection point. Thus, there is still room for the improvement of Algorithm 6.3.7 especially concerning its complexity with respect to d.

Algorithm 6.3.7 (Efficient Cell Enumeration)

```
Input: Hyperplane arrangement $\mathcal{A}_{n,d}$ defined by $\mathbf{A}$ and $\mathbf{b}$.
Output: List of the position vectors $\mathbf{p}$ of all cells in the arrangement.
  a) compute the set of all intersection points $\tilde{\mathcal{V}}_{n,d}$ from Lemma 6.3.6
  b) for each $\mathbf{v} \in \tilde{\mathcal{V}}_{\mathbf{n,d}}$
        b.1) compute the position vector $\mathbf{p_v}$ of $\mathbf{v}$
             set $\mathbf{p} = \mathbf{p_v}$
        b.2) let $H_i$ denote the numbers of the $d$ hyperplanes intersecting $\mathbf{v}$
             for $i = 1, \ldots, d$
                find a backward point $\mathbf{b}$ on the edge through $\mathbf{v}$ which
                   is not on hyperplane $H_i$
                compute position vector $\mathbf{q}$ of $\mathbf{b}$
                set $p_{H_i} = q_{H_i}$
             output $\mathbf{p}$
```

6.4 Orthant Decomposition

In applications, the ability to efficiently enumerate all cells of a hyperplane arrangement is often only the first step. Typically, on each cell some kind of computations have to be carried out such as the integration of a function or the solution of a linear program. Often, the complex structure of the cells (the polyhedra can have many sides and vertices) make computations difficult and involved. Thus, it is desirable to have simple building blocks, like hypercubes, simplices or orthants. In many approaches, each cell in the arrangement is decomposed into smaller elements (e.g., by triangulation or trapezoidal decomposition, see, e.g.,[62]) with the disadvantage that the number of elements can rise substantially.

On the other hand, sometimes it is advantageous to represent each cell as the intersection or difference of larger elements. For example, if we have to integrate a function f within a polyhedral cell P and we can represent the cell as the difference $P = B \setminus A$ of two simpler cells A and B (having fewer vertices) we can use the relation

$$\int_P f(\mathbf{x})\ d\mathbf{x} = \int_{B\setminus A} f(\mathbf{x})\ d\mathbf{x} = \int_B f(\mathbf{x})\ d\mathbf{x} - \int_A f(\mathbf{x})\ d\mathbf{x}, \tag{6.10}$$

provided we can extend f onto B. If we can continue this process with B and A by again replacing them with simpler cells, it is possible to compute the complicated integral over P by a series of integrals over simpler elements. This is especially important when numerical integration is needed since efficient quadrature formulas are only available for simple integration regions [121]. For an application of this technique in computational finance see [49].

In this section, we discuss the algebraic decomposition of a hyperplane arrangement into orthants. A (generalized) *orthant* (sometimes called cone) is defined as the intersection of k not necessarily orthogonal half-planes in $\mathbb{R}^d$ where $k \leq d$. We will show that it is possible to decompose all cells in a non-degenerate hyperplane arrangement using exactly one orthant per cell. We will also give an efficient algorithm for the computation of this decomposition.

6.4.1 Signed Polyhedral Decomposition

In the following, we illustrate a special orthant decomposition which uses the nice one-to-one correspondence between vertices and polyhedral cells of Lemma 6.3.6 and which is easy to realize. Let us remark here that an orthant decomposition is not unique. A different decomposition of a polyhedron into a sum of orthants is, e.g., presented in [26]. In a non-degenerate hyperplane arrangement there exist exactly $2^d\binom{n}{d}$ different orthants of dimension $k = d$ alone (around each of the $\binom{n}{d}$ interior vertices there are 2^d adjacent orthants). Thus, there are many potential orthant candidates to choose from.

First, we have to discuss the signed decomposition of a single convex polyhedron into orthants. Thereby, we use the following orthants.

Definition 6.4.1 *The orthant $O_{\mathbf{v}}$ corresponding to the intersection point $\mathbf{v} \in \tilde{\mathcal{V}}_{n,d}$ is defined as the unique polyhedron defined by the k hyperplanes through $\mathbf{v}$ which intersects the cell $P_{\mathbf{v}} = \pi(\mathbf{v})$.*

For illustration, the orthant $O_{\mathbf{v}_4}$ is displayed in Figure 6.3. Note that vertices which are located on the boundary of C correspond to orthants with $k < d$ intersecting hyperplanes. For example, $O_{\mathbf{v}_3}$ is defined by all points which are below hyperplane one. For the signed decomposition, we now require the following reflection signs.

Definition 6.4.2 *Given two vertices $\mathbf{v}, \mathbf{w} \in \tilde{\mathcal{V}}_{n,d}$, we define the reflection sign $s_{\mathbf{v},\mathbf{w}}$ by*

$$s_{\mathbf{v},\mathbf{w}} := (-1)^{r_{\mathbf{v},\mathbf{w}}} \tag{6.11}$$

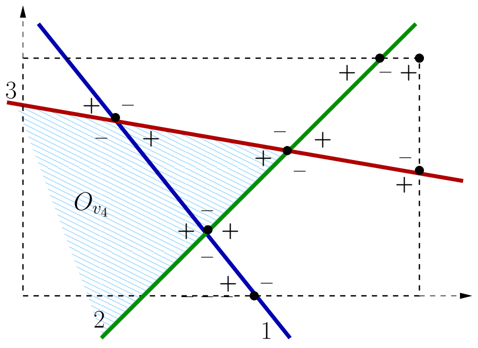


Figure 6.3: The orthant $O_{\mathbf{v}_4}$ and all reflection signs $s_{\mathbf{v},\mathbf{w}}$.

where $r_{\mathbf{v},\mathbf{w}}$ is the number of reflections on hyperplanes needed to map the orthant $O_{\mathbf{w}}$ onto the polyhedral cell $P_{\mathbf{v}}$.

The reflection signs $s_{\mathbf{v},\mathbf{w}}$ with $\mathbf{v}, \mathbf{w} \in \{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ arising in our two-dimensional example arrangement are displayed in Figure 6.3. For instance, the three reflection signs in the cell $P_{\mathbf{v}_4}$ are given by $s_{\mathbf{v}_4,\mathbf{v}_1} = +$, $s_{\mathbf{v}_4,\mathbf{v}_2} = -$ and $s_{\mathbf{v}_4,\mathbf{v}_4} = +$.

Now, we are able to reiterate the following signed decomposition Lemma which is originally due to Lawrence [85]:

Lemma 6.4.3 (Lawrence, 1991) *Let $\mathcal{V}_{\mathbf{v}}$ denote the set of all vertices of the polyhedron $P_{\mathbf{v}}$. It is possible to algebraically decompose the polyhedron into a signed sum of orthant cells by*

$$\chi(P_{\mathbf{v}}) = \sum_{\mathbf{w} \in \mathcal{V}_{\mathbf{v}}} s_{\mathbf{v},\mathbf{w}} \chi(O_{\mathbf{w}}) \tag{6.12}$$

where χ is the characteristic function of a set.

Some applications require the decomposition of all cells arising in a given hyperplane arrangement. Then, it is important to ensure that the overall number of required orthants is as small as possible as often time consuming operations, such as numerical integration, have to be performed on each orthant.

Based on the signed decomposition property of polyhedra and using the $c_{n,d}$ orthants $O_{\mathbf{v}}$ we can now realize an algebraic decomposition of a hyperplane arrangement into orthants which is optimal in the sense that the number of required orthants equals the number of cells which are decomposed.

Lemma 6.4.4 *Applying the signed decomposition of Lemma 6.4.3 to each cell in a hyperplane arrangement $\mathcal{A}_{n,d}$, all cells are decomposed into signed sums of orthants whereby exactly one orthant per cell is used.*

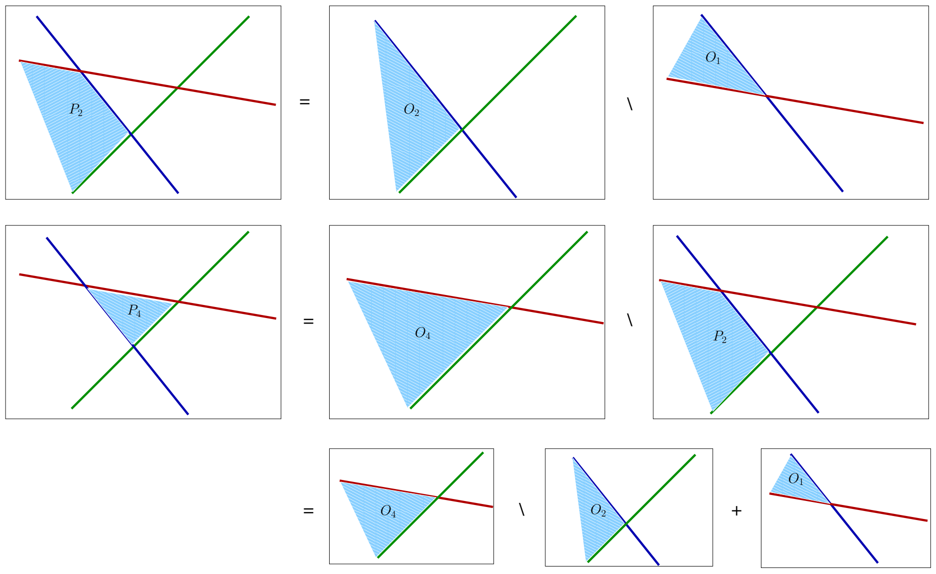


Figure 6.4: Decomposition of three cells using three orthants.

Proof: By Lemma 6.3.6 we have a one-to-one correspondence between the set $\tilde{\mathcal{V}}_{n,d}$ of intersection points with the cells P in $\mathcal{A}_{n,d}$. Each cell in $\mathcal{A}_{n,d}$ can be decomposed by Lemma 6.4.3 using a subset of the $c_{n,d}$ orthants $O_{\mathbf{v}}$. Thus, the complete hyperplane arrangement can be decomposed using the $c_{n,d}$ orthants $O_{\mathbf{v}}$ and, this way, exactly one orthant per cell. $\square$

To give an example, the decomposition of all cells of the hyperplane arrangement from Figure 5.2, right, is given by

$$\begin{aligned}
\chi(P_1) &= \chi(O_1) \\
\chi(P_2) &= \chi(O_2) - \chi(O_1) \\
\chi(P_3) &= \chi(O_3) - \chi(O_2) \\
\chi(P_4) &= \chi(O_4) - \chi(O_2) + \chi(O_1) \\
\chi(P_5) &= \chi(O_5) - \chi(O_4) - \chi(O_1) \\
\chi(P_6) &= \chi(O_6) - \chi(O_4) - \chi(O_3) + \chi(O_2) \\
\chi(P_7) &= \chi(O_7) - \chi(O_6) - \chi(O_5) + \chi(O_4)
\end{aligned} \tag{6.13}$$

where we used the abbreviations $P_j := P_{\mathbf{v}_j}$ and $O_j := O_{\mathbf{v}_j}$. We see that seven orthants are required for the decomposition of seven cells. In Figure 6.4, the decomposition of three polyhedral cells P_1, P_2, P_4 into the three orthants O_1, O_2, O_4 is illustrated. Note that the small orthant O_1 directly corresponds to the cell P_1.

6.4.2 Orthant Decomposition Algorithm

Now, we can give the complete algorithm for the orthant decomposition of a hyperplane arrangement (Algorithm 6.4.5). Again we start with the set of all intersection points $\tilde{\mathcal{V}}_{n,d}$.

For each intersection point $\mathbf{v}$, we first determine in step b.1) its associated polyhedron $P_{\mathbf{v}}$ and its associated orthant $O_{\mathbf{v}}$ using Algorithm 6.3.7. Note that the position vector of the orthant $O_{\mathbf{v}}$ is given as a subvector of the position vector of the polyhedron $P_{\mathbf{v}}$ using only the d signs corresponding to the d planes through the vertex $\mathbf{v}$. In the next step b.2), all vertices $\mathbf{w} \in \mathcal{V}_{\mathbf{v}} \subset \mathcal{V}_{n,d}$ are determined by a vertex enumeration algorithm. For each of these vertices, the reflection sign $s_{\mathbf{v},\mathbf{w}}$ is determined in the steps b.3) and b.4). Here, the exponent $r_{\mathbf{v},\mathbf{w}}$ of the reflection signs is given by the number of entries in the position vector of the orthant $O_{\mathbf{w}}$ which differ from the corresponding entries in the position vector of $P_{\mathbf{v}}$.

Algorithm 6.4.5 (Orthant Decomposition Algorithm)

```
Input: Hyperplane arrangement $\mathcal{A}_{n,d}$ defined by $\mathbf{A}$ and $\mathbf{b}$.
Output: Orthant-decomposition of each cell $P \in \mathcal{A}_{n,d}$.
  a) compute the set of all intersection points $\tilde{\mathcal{V}}_{n,d}$ using Lemma 6.3.6
  b) for each $\mathbf{v} \in \mathcal{V}$
        b.1) determine the associated polyhedron $P_{\mathbf{v}}$ and orthant $O_{\mathbf{v}}$
        b.2) determine the vertices $\mathcal{V}_{\mathbf{v}}$ of the polyhedron $P_{\mathbf{v}}$
        b.3) determine the reflection signs $s_{\mathbf{v},\mathbf{w}}$ for all $\mathbf{w} \in \mathcal{V}_{\mathbf{v}}$
        b.4) decompose $P_{\mathbf{v}}$ into the orthants $O_{\mathbf{w}}$ using Lemma 6.4.3
```

Let us also discuss the complexity of Algorithm 6.4.5. The steps a) and b.1) are essentially identical to Algorithm 6.3.7. For the computation of all vertices $\mathcal{V}_{\mathbf{v}}$ of the polyhedron $P_{\mathbf{v}}$ in step b.2) the pivoting algorithm [1] can be used. It requires $O(|\mathcal{V}_{\mathbf{v}}|nd)$ time and $O(nd)$ space. Steps b.3) and b.4) also require $O(|\mathcal{V}_{\mathbf{v}}|d)$ time. Since for fixed d the average number of vertices of a polyhedron is of order $O(n)$, the overall complexity of Algorithm 6.4.5 is given by

$$O(n^{d+2}). \tag{6.14}$$

6.5 Computational Results

In this section, we illustrate the efficiency of the Algorithms 6.3.1, 6.3.7 and 6.4.5. To this end, we measure the time which is needed by the algorithms to enumerate and decompose example hyperplane arrangements with up to 150,000 cells in up to dimension 7. Following [116], the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ which define the arrangement are chosen randomly. The entries of $\mathbf{A}$ and $\mathbf{b}$ are uniformly distributed in $[-16,384, 16,384]$ and $[-10,000, 10,000]$, respectively. All computations were performed on a dual Intel(R) Xeon(TM) CPU 3.06GHz computer.

The results are displayed in Table 6.2. For each example, the number of hyperplanes n, the dimension d and the number $c_{n,d}$ of cells are stated in the first three columns. The times required by Algorithm 6.3.1 and Algorithm 6.3.7 to enumerate all cells of the arrangement

n	d	$c_{n,d}$	Alg. 0	Alg. 1	Alg. [116]	Alg. 2
10	4	386	0.01	0.01	1.86	0.01
20	4	6,196	0.17	0.08	63	0.10
30	4	31,931	1.16	0.33	486	0.73
44	4	149,986	13.99	1.70	3,388	4.36
9	5	382	0.01	0.01	2.36	0.02
15	5	4,944	0.16	0.11	56	0.19
21	5	27,896	1.48	0.55	448	1.32
29	5	146596	11.13	2.71	3,249	8.11
8	6	247	0.01	0.01	1.62	0.05
13	6	4,096	0.14	0.29	54	0.46
18	6	31,180	2.19	1.35	611	3.54
23	6	145,499	5.17	4.88	3,714	19.42

Table 6.2: Running times in seconds for several example hyperplane arrangements. Alg. 0, Alg. 1 and Alg. [116] enumerate the arrangement, Alg. 2 enumerates and decomposes it.

are then given in columns four and five. For comparison, we report in the sixth column the running times taken from [116] for the same type of problems. These computations were conducted on a slower computer. On our computer, we expect the running times to be faster by a factor of 4-6. To our knowledge, these are the only running times which can be found in the literature. In column seven, we display the running times of Algorithm 6.4.5 for the orthant decomposition of the corresponding hyperplane arrangements.

One can see that all considered hyperplane arrangements are very quickly enumerated by Algorithm 6.3.7. In no example the enumeration takes more than five seconds. Note for comparison that the algorithm in [116] requires up to one hour for the same examples. Algorithm 6.3.1 is competitive especially for small n and large d and requires at most 14 seconds. The orthant decomposition using Algorithm 6.4.5 requires from 0.01 to about 20 seconds depending on the specific example.

The dependency of the running times of Algorithms 6.3.1, 6.3.7 and 6.4.5 on the number of cells and on the dimension is illustrated in Figure 6.5. One can see, that in all cases the time needed increases almost linearly with the number of cells in the arrangement. The constant with respect to n in the complexity of Algorithm 6.3.1 is worse than for Algorithm 6.3.7 which is indicated by a larger slope. As expected, the running times of all three algorithms depend exponentially on the dimension. We also see that the constant with respect to d in the complexity is for Algorithm 6.3.1 better than for Algorithm 6.3.7. The behaviour of Algorithm 6.4.5 with respect to n and d is similar to Algorithm 6.3.7.

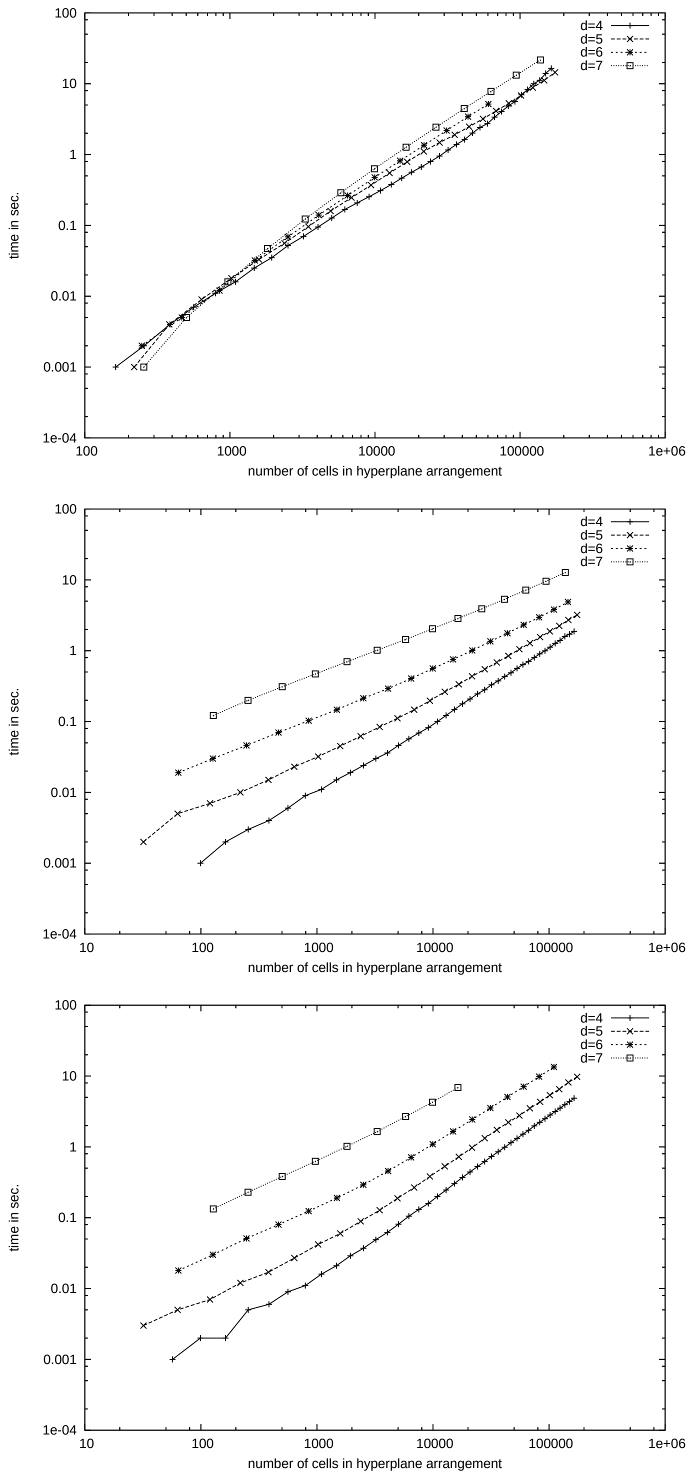


Figure 6.5: Time in seconds for the cell enumeration using Algorithm 6.3.1 (top) and Algorithm 6.3.7 (middle) as well as for the orthant decomposition using Algorithm 6.4.5 (bottom) of several hyperplane arrangements in dimensions $d = 4 - 7$.

Chapter 7

Simulation Methods

7.1 Introduction

If no closed-form solution for the price of a financial derivative is known (as is the case for the most types of options), numerical simulation methods have to be employed. Of these, Monte Carlo simulation is certainly the best-known and the most frequently used variant. Monte Carlo simulation is, however, no numerical but a statistical method. Convergence takes place only in the statistical average and no deterministic error bounds are possible. The convergence rate itself is small but independent of the dimension of the problem. This last property and its (relatively) simple implementation makes it an important tool in many financial applications, though.

Besides the Monte Carlo method we will also consider deterministic methods such as Quasi-Monte Carlo and numerical quadrature methods. These methods lead to a substantially faster convergence especially for smooth problems. However, the convergence rate of the methods more or less depends on the dimension of the problem.

First, we will consider tree approximation methods, the binomial method and the stochastic mesh method which can be seen as special discretizations of the expectation of Theorems 4.3.1 and 4.3.3.

7.2 Tree methods

We first consider a simple discrete time model for the future development of securities. This model was developed in by Cox, Ross and Rubinstein [18] and is thus called CRR model. Under these model assumptions fair prices for standard options can be derived. Under certain assumptions, the option price of the discrete CRR model converges against the price of the Black-Scholes model if the time intervals tend to zero.

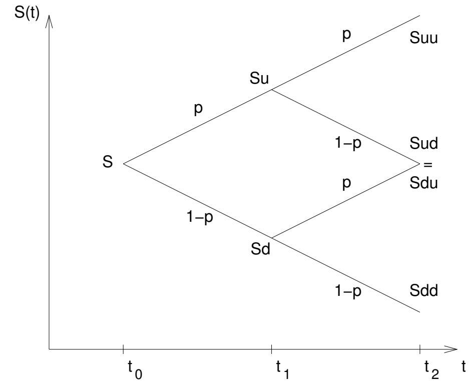


Figure 7.1: The first two steps of a binomial tree.

7.2.1 CRR model

We will now partition the time interval $[0, T]$ into $M + 1$ time steps t_j

$$t_j = j\Delta t \quad \text{for} \quad j = 0, \dots, M, \tag{7.1}$$

where $\Delta t = T/M$. In between two time steps the price of the security can move either up by a factor of u or down by a factor of d $(0 < d < u)$. Here, the probability of an upward movement is p and for a downward movement $1 - p$. Let $\xi_i \in \{u, d\}, 1 \leq i \leq M$ be these random factors, then the price of the security follows

$$S(t_j) = S(0) \prod_{i=1}^{j} \xi_j. \tag{7.2}$$

In this way, $S \cdot ud = S \cdot du$ and therefore the growth of the different outcomes is limited since after M time steps $S(t_M)$ can only attain $M + 1$ different values (see Figure 7.1.

This process is also called binomial process and the time discrete model also binomial model. For a suitable choice of u, d and p one attains convergence against the Wiener process for $\Delta t \to 0$ and therefore the binomial model can be understood as a discretization of the continuous model.

The free parameters u, d and p do not reflect an individual market expectation but are determined by three equations in such a way that a risk-neutral valuation follows. The first equation for this is

$$u \cdot d = 1. \tag{7.3}$$

which is chosen for a symmetry of up- and downward movements. The two remaining equations for the fixation of u, d and p result from a equalization of the expectations and variances of the discrete and the continuous model. In the discrete model, the expectation of the price S at time t_{j+1} is given by

$$E(S(t_{j+1})) = pS(t_j)u + (1 - p)S(t_j)d, \tag{7.4}$$

and the variance by

$$Var(S(t_{j+1})) = p(S(t_j)u)^2 + (1-p)(S(t_j)d)^2 - S^2(t_j)(pu + (1-p)d)^2. \tag{7.5}$$

In the continuous model we have according to (3.6) and (3.7)

$$E(S(t_{j+1})) = S(t_j) \cdot e^{r\Delta t} \tag{7.6}$$

and

$$Var(S(t_{j+1})) = S^2(t_j)e^{(2r+\sigma^2)\Delta t} - S(t_j) \cdot e^{r\Delta t} = S^2(t_j)e^{2r\Delta t}(e^{\sigma^2 \Delta t} - 1). \tag{7.7}$$

After the solution of the (nonlinear) system of equations we get the three parameters u, d and p as functions of σ, r and Δt according to

$$\begin{aligned} u &= \beta + \sqrt{\beta^2 - 1} && (7.8) \\ d &= 1/u = \beta - \sqrt{\beta^2 - 1} && (7.9) \\ p &= \frac{e^{r\Delta t} - d}{u - d} && (7.10) \end{aligned}$$

where $\beta = \frac{1}{2}(e^{-r\Delta t} + e^{(r+\sigma^2)\Delta t})$.

7.2.2 Binomial Method

The actual binomial method consists of two phases, the forward and the backward phase. In the forward phase the future security prices are initialized. To this end, the different outcomes are represented as a two-dimensional array S_{ij}, where $S_{00} = S(t_0)$ is the starting value and

$$S_{ij} = S(t_0)u^i d^{j-i} \tag{7.11}$$

for $1 \le j \le M$ and $0 \le i \le j$. This way, S_{ij} is the $i-$th possible outcome at time t_j. For European options it is sufficient to compute S_{ij} only for $j = M$ and $i = 0, \dots, j$ instead for all i and j. For American options the whole array has to be computed due to the early exercise right.

In the backward phase the option prices are computed and stored in a corresponding array V_{ij}. At time $T = t_M$ the value of the option V is known due to the payoff function and we have therefore

$$V_{iM} = (S_{iM} - K)^+ \tag{7.12}$$

for call options and correspondingly $V_{iM} = (K - S_{iM})^+$ for put options. Now, the values V_{ij} are computed backwards for each t_j from t_{j+1}. In the case of European options one computes

$$V_{ij} = e^{-r\Delta t} \cdot (pV_{i+1,j+1} + (1-p)V_{i,j+1}). \tag{7.13}$$

In the case of American options early exercise has to be checked and in the case of call options one computes

$$V_{ij} = \max\{(S_{ij} - K)^+, e^{-r\Delta t} \cdot (pV_{i+1,j+1} + (1-p)V_{i,j+1})\}. \tag{7.14}$$

For put options the corresponding formulas are used with $(K - S_{ij})^+$. This way, $V(S, 0) = V_{00}$ is the computed option price at time $t_0 = 0$. In summary, the binomial method for European and American options is shown in Algorithm 7.2.1.

Algorithm 7.2.1 (Binomial Method)

```
compute $u$, $d$, $p$ from (7.8)–(7.10)
$S_{00} = S(0)$
for $j = 1, \dots, M$
   for $i = 0, \dots, i$
      set $S_{ij} = S_{00} u^i d^{j-i}$
for $i = 0, \dots, j$
   compute $V_{iM}$ from (7.12)
for $j = M - 1, \dots, 0$
   for $i = 0, \dots, j$
   compute $V_{ij}$ from (7.13) resp. (7.14)
$V = V_{00}$
```

For the binomial method, the derivative of the option price with respect to the volatility is not easily computable. Here, either approximations of the derivative by difference quotients or derivative-free zero finding methods, such as the bisection method, can be applied.

7.3 Stochastic Meshes

Another more general way to simulate early exercise options is the stochastic mesh method due to Broadie and Glasserman [10]. The stochastic mesh method generates two expected values for the option price, one with a too high bias and one with a too low one. The biases of both expectations tends to zero if the number of simulations tends to ∞. The two expectations serve as confidence interval for the option price.

First, a random tree with B branches per node is constructed (see Figure 7.2) where the asset prices at times t_j are denoted by $S_j^{i_1 i_2 \dots i_j}$, $j = 1, 2, \dots M$ and $1 \leq i_1 \dots i_j \leq B$. These prices are generated by a random walk in a forward step. In a backward step now (in a similar way as in the binomial method) the high and low expected value is computed. The high expectation is called $\theta_{high,j}^{i_1 \dots i_j}$ and is recursively computed by

$$\theta_{high,j}^{i_1 \dots i_j} = \max \left\{ V(S_j^{i_1 i_2 \dots i_j, t_j}), \frac{1}{B} \sum_{i_{j+1}=1}^{B} e^{-r\Delta t} \theta_{high,j+1}^{i_1 \dots i_j i_{j+1}} \right\}. \tag{7.15}$$

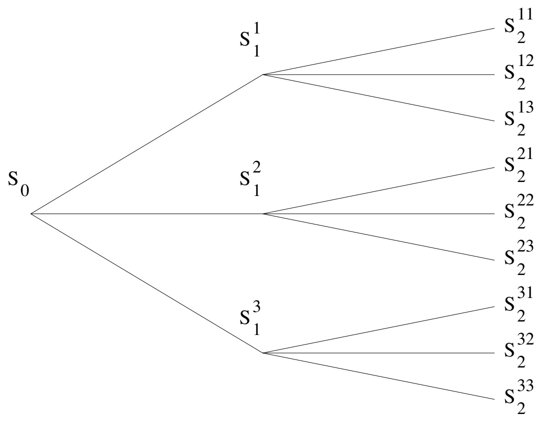


Figure 7.2: Simulation tree with three branches and two time steps.

The low expectation $\theta_{low,j}^{i_1\ldots i_j}$ is computed in two steps. First one sets

$$\eta_j^{i_1\ldots i_j k} = \begin{cases} V(S_j^{i_1\ldots i_j}) & \text{if } V(S_j^{i_1\ldots i_j}) \geq \frac{1}{B-1} \sum\limits_{\substack{i_{j+1}=1 \\ i_{j+1}\neq k}} e^{-r\Delta t}\theta_{low,j+1}^{i_1\ldots i_j i_{j+1}} \text{ for } k = 1\ldots B \\ e^{-r\Delta t}\theta_{low,j+1}^{i_1\ldots i_j k} & \text{else} \end{cases} \tag{7.16}$$

and then $\theta_{low,j}^{i_1\ldots i_j}$ is determined by

$$\theta_{low,j}^{i_1\ldots i_j} = \frac{1}{B}\sum_{k=1}^{B} \eta_j^{i_1\ldots i_j k}. \tag{7.17}$$

Thereby, both $\theta_{low,M}^{i_1\ldots i_M}$ and $\theta_{high,M}^{i_1\ldots i_M}$ is initialized to $V(S_M^{i_1\ldots i_M}, M)$ As an approximate option price then

$$V(S,0) = \frac{1}{2}(\theta_{high,0} + \theta_{low,0}) \tag{7.18}$$

can be used. At first sight, the stochastic mesh method looks considerably more complicated than the binomial method, which practically solves the same problem. The advantage of the stochastic mesh method lies in its easier generalizability to more complex options like path-dependent or multi-asset options.

7.4 Univariate Integration Methods

Numerical integration methods are a natural way to compute the expectations arising in derivative security pricing. We will now shortly take a look at standard univariate integration methods which will also play a role for the construction multivariate integration methods.

At first let us recall the definition of a quadrature formula for the solution of a univariate integration problem

$$I^{(1)}f = \int_0^1 f(x)\;dx \approx \sum_{i=1}^{N} w_i f(x_i). \tag{7.19}$$

with weights w_i and points x_i, $1 \leq i \leq N$. Now we want to assign a quadrature formula a level $l = 1, 2, \ldots$ and let the number of points of the quadrature formula N_l depend on the level, i.e.

$$Q_l^{(1)}f = \sum_{i=1}^{N_l} w_{li} f(x_{li}). \tag{7.20}$$

Thereby, the number of points is designed to roughly double in between levels, i.e.

$$N_l = O(2^l). \tag{7.21}$$

A series of quadrature formulas is called nested, if the set of points of $Q_l^{(1)}$ is a subset of the points of $Q_l^{(1)}$. Nested quadrature formulas are important in practice for error estimation purposes as well as the construction of multivariate quadrature formulas. In the following, we give a short review of nested univariate quadrature formulas for functions $f \in \mathcal{C}^r$ with

$$\mathcal{C}^r := \left\{ f : \Omega \longrightarrow \mathbb{R}, \left\| \frac{\partial^s f}{\partial x^s} \right\|_\infty < \infty,\ s \leq r \right\}, \tag{7.22}$$

As we will see later, it is of great importance that $N_1 = 1$. Therefore, in the following we always set

$$Q_1^{(1)}f = 2 \cdot f(0). \tag{7.23}$$

7.4.1 Trapezoidal Rule

The Newton–Cotes formulas [19] use equidistant abscissas and determine the corresponding weights by integration of the Lagrange polynomials through these points. The closed versions include the endpoints of the interval, whereas the open ones omit one or both of them. The formulas get numerically instable for large numbers of points, i.e. some of the weights will become negative.

Therefore, iterated versions of low degree formulas are most commonly used. A well known example is the iterated trapezoidal rule. Here we use

$$N_l = 2^l - 1 \tag{7.24}$$

and therefore have as the open iterated trapezoidal rule

$$Q_l^{(1)}f = \frac{1}{N_l+1}\left(\frac{3}{2}\cdot f\left(\frac{1}{N_l+1}\right) + \sum_{i=2}^{N_l-1} f\left(\frac{i}{N_l+1}\right) + \frac{3}{2}\cdot f\left(\frac{N_l}{N_l+1}\right)\right). \tag{7.25}$$

The error bounds are well known and for functions $f \in \mathcal{C}^2$ of the form

$$|E_l^1 f| = O(N_l^{-2}). \tag{7.26}$$

For $[-1, 1]$–periodic functions $f \in \mathcal{C}^r$, this bound improves to

$$|E_l^1 f| = O(2^{-lr}). \tag{7.27}$$

Instead of the trapezoidal rule of course also the Simpson rule or higher Newton-Cotes rules with corresponding boundary modifications (and higher convergence rates) can be used.

7.4.2 Clenshaw-Curtis Formulas

The Clenshaw-Curtis formulas [17] are numerically more stable and use the non-equidistant abscissas given as the zeros or the extreme points of the Chebyshev polynomials. The quadrature formulas are nested in case the extreme points are used. We select N_l like in the trapezoidal rule. The abscissas of the open Clenshaw-Curtis formulas (also called Filippi formulas) are then given by

(7.28)

and the weights by

$$w_{li} = \frac{2}{N_l + 1} \sin \frac{\pi i}{N_l + 1} \sum_{j=1}^{(N_l+1)/2} \frac{1}{2j - 1} \sin \frac{(2j-1)\pi i}{N_l + 1} \; . \tag{7.29}$$

The amount of work for the computation of the weights can be reduced to order $O(N_l \log N_l)$ using a variant of the FFT algorithm [38]. The polynomial degree of exactness is $N_l - 1$ and the error bounds for $f \in \mathcal{C}^r$ are therefore [19].

$$|E_l^1 f| = O(N_l^{-r}). \tag{7.30}$$

For only continuous functions the convergence rate is 1, for infinitely smooth functions, the convergence is exponential.

7.4.3 Gauss and Gauss–Patterson Formulas

Gauss formulas have the maximum possible polynomial degree of exactness of $2n - 1$. For the case of the unit weight function the abscissas are the zeroes of the Legendre polynomials and the weights are computed by integrating the associated Lagrange polynomials. However, these Gauss–Legendre formulas are in general not nested.

Kronrod [82] extended an n–point Gauss quadrature formula by $n + 1$ points such that the polynomial degree of exactness of the resulting $2n + 1$ formula is maximal. This way, quadrature formulas with degree $2n + \bar{n} + 2$ with

$$\bar{n} := \begin{cases} n, & \text{if } n \text{ odd} \\ n - 1, & \text{else} \end{cases} \; . \tag{7.31}$$

are obtained. For the Gauss–Legendre formula, the new abscissas are real, symmetric, inside the integration interval and interlace with the original points. Furthermore, all weights are positive. It turned out [89] that the new abscissas are the zeros of the Stieltjes polynomial F_{n+1} satisfying

$$\int_{-1}^{1} P_n(x)F_{n+1}(x)x^j \; dx = 0, \quad \text{for} \quad j = 0, 1, \ldots, n, \tag{7.32}$$

where $P_n(x)$ is the n–th Legendre polynomial. Therefore, F_{n+1} can be seen as the orthogonal polynomial with respect to the weight function $P_n(x)$ which is of varying sign. The polynomial F_{n+1} can be computed by expanding it in terms of Legendre [101] or Chebyshev [104] polynomials and solving the resulting linear system. The zeroes of F_{n+1} can then be calculated by a modified Newton method. Alternatively, the computation of the abscissas can be achieved by the solution of a partial inverse eigenvalue problem [53]. Finally, the weights are computed just like in Gauss formulas by integration of the Lagrange polynomials through the computed abscissas.

Patterson [101] iterated Kronrod's scheme recursively and obtained a sequence of nested quadrature formulas with maximal degree of exactness. He constructed a sequence of polynomials $G_k(x)$ of degree $2^{k-1}(n+1), \; k \geq 1$, satisfying

$$\int_{-1}^{1} P_n(x)(\prod_{i=1}^{k-1} G_i(x))G_k(x)x^j \; dx = 0 \quad \text{for} \quad j = 0, 1, \ldots, 2^{k-1}(n+1) - 1. \tag{7.33}$$

This way, $G_1(x) = F_{n+1}(x)$ and the G_j are orthogonal to all polynomials of degree less than $2^{k-1}(n+1)$ with respect to the variable signed weight function $P_n(x)(\prod_{i=1}^{j-1} G_i(x))$. The $2^k(n+1) - 1$ abscissas of the resulting quadrature formulas are the zeroes of P_n and all $G_j, 1 \leq j < k$. The abscissas and weights can be computed similar to the Kronrod case. This way, formulas of degree $(3 \cdot 2^{k-1} - 1)(n+1) + \bar{n}$ can be obtained – at least in theory.

However, Patterson extensions do not exist for all Gauss–Legendre formulas. For example, in the case of the 2–point Gauss–Legendre formula, only four extensions are possible [102]. But, starting with the 3–point formula, extensions exist for practicable k and all properties of Kronrod's scheme are preserved.

We set Q_2^1 equal to the 3–point Gauss–Legendre formula, and $Q_l^1, \; l \geq 3$, equal to its $(l-2)$nd Patterson extension. This way, $N_l = 2^l - 1$ and the polynomial degree of exactness is $3 \cdot 2^{l-1} - 1$ for $l \geq 2$. The error is therefore for $f \in \mathcal{C}^r$ again

$$|E_l^1 f| = O(2^{-lr}). \tag{7.34}$$

Of the considered nested quadrature formulas (with the restriction to periodic functions in the case of the trapezoidal rule) all achieve the optimal order of accuracy $O(2^{-lr})$. Among these, the Gauss–Patterson formulas achieve the highest possible polynomial exactness of nearly $(3/2)N_l$ compared to $n_1^l - 1$ for the Clenshaw–Curtis and Filippi formulas and 1 for the trapezoidal rule. From the results in [9] also follows that the Peano constants are smaller in comparison to the other formulas considered.

However, the existence of Patterson extensions is at the time not clear for large k, i.e. $k > 5$. Still, for Smolyak's construction the existing Patterson formulas are sufficient for moderate and high dimensional problems.

Note, that although the order of N_l is the same in all cases, the actual number of points in the trapezoidal and Clenshaw–Curtis formulas compared to the Filippi, Gauss–Legendre and Patterson formulas can differ by almost a factor of 2 for the same level l.

7.4.4 Domain Transformation

In financial derivative pricing problems, the integration domain is the whole real line (for higher-dimensional problem the whole real space), which is a problem for numerical integration methods which usually work on a finite interval. One possibility would be to cut off the integration domain at finite points whereby, of course, an additional error is made. Since the integrand decays very quickly towards $\pm\infty$ (because of the Gauss weight), this strategy is admissible but cutoff points are difficult to determine a priori. Another method is to use weighted quadrature formulas such as Gauss-Hermite formulas. These quadrature formulas are, in general, not nested though and difficult to adapt to new situations.

A better possibility is to use a suitable substitution of variables to transform the integral onto the unit interval $[0, 1]$. The obvious substitution is here $y = N^{-1}(x)$, i.e. to use the inverse (cumulative) normal distribution. This way we have in the example of a European call option

$$V(S,0) = e^{-rT} \int_0^1 \left(S(0) e^{(r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T} N^{-1}(x)} - K \right)^+ \, dx \tag{7.35}$$

and, alternatively, using the knowledge of the zero χ

$$V(S,0) = e^{-rT} \int_{N(\chi)}^1 S(0) e^{(r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T} N^{-1}(x)} - K \, dx. \tag{7.36}$$

The second representation has the advantage of a smooth integrand (C^∞), while in the first representation the integrand has a discontinuous first derivative (C^0). In both cases, the inverse normal distribution is required for which a fast approximate Moro method (Algorithm 7.4.1) is available.

Algorithm 7.4.1 (Inverse Normal Distribution)

```
$E0 = 2.50662823884$              $E1 = -18.61500062529$        $E2 = 41.39119773534$
$E3 = -25.44106049637$            $F0 = -8.47351093090$         $F1 = 23.08336743743$
$F2 = -21.06224101826$            $F3 = 3.13082909833$          $G0 = 0.3374754822726147$
$G1 = 0.9761690190917186$         $G2 = 0.1607979714918209$     $G3 = 0.0276438810333863$
$G4 = 0.0038405729373609$         $G5 = 0.0003951896511919$     $G6 = 0.0000321767881768$
$G7 = 0.0000002888167364$         $G8 = 0.0000003960315187$
$p = x - 0.5$
$if\ abs(p) < 0.42$
  $r = p * p$
  $Ninv = p * (((E3 * r + E2) * r + E1) * r + E0)/((((F3 * r + F2) * r + F1) * r + F0) * r + 1.0)$
$else$
  $if\ p < 0$
     $r = x$
  $else$
     $r = 1 - x$
  $r = \log(-\log(r))$
  $r = G0 + r * (G1 + r * (G2 + r * (G3 + r * (G4 + r * (G5 + r * (G6 + r * (G7 + r * G8)))))))$
  $if\ p < 0$
     $Ninv = -r$
  $else$
     $Ninv = r$
```

7.5 Multivariate Integration Methods

As we have seen in chapter 4, the pricing of path-dependent or multi-asset derivatives using the martingale approach requires the solution of multivariate integration problems. We will now indicate a few simple quadrature methods for the integration of a function f over the unit hypercube $[0,1]^d$ of the form

$$I^{(d)} f = \int_{[0,1]^d} f(\mathbf{x})\ d\mathbf{x} \approx \sum_{i=1}^{N} w_i f(\mathbf{x}_i) \tag{7.37}$$

with corresponding weights w_i and abscissas $\mathbf{x}_i$. In order to handle the whole real space as integration domain, the substitution using the inverse normal distribution of the previous section is applied to each direction separately.

7.5.1 Product Approach

In the product approach, simply the tensor product of univariate quadrature formulas with equal level l are used for the construction of multivariate quadrature formulas, i.e.

$$Q_l^{(d)} f = (Q_l^{(1)} \otimes \ldots \otimes Q_l^{(1)}) f. \tag{7.38}$$

The tensor product of d quadrature formulas $(Q_{l_1}^{(1)} \otimes \ldots \otimes Q_{l_d}^{(1)})f$ (here with different levels l_1 to l_d, since this more general method is required soon), is defined as the sum over all combinations

$$(Q_{l_1}^{(1)} \otimes \ldots \otimes Q_{l_d}^{(1)})f = \sum_{i_1=1}^{N_{l_1}} \ldots \sum_{i_d=1}^{N_{l_d}} w_{l_1 i_i} \cdot \ldots \cdot w_{l_d i_d} \cdot f(x_{l_1 i_1}, \ldots, x_{l_d i_d}). \tag{7.39}$$

Thus, the integrand is evaluated at the points of a product grid where the resulting multidimensional weights are the products of the corresponding one-dimensional weights.

When implementing the product approach, one encounters an unexpected difficulty. Since the dimension d is a variable parameter, the number of sums (and therefore the number of for-loops) is a priori unknown. Therefore, so-called drop algorithms are employed here which are also used e.g. for the enumeration of binary numbers.

Algorithm 7.5.1 (Product Quadrature)

```
for $j = 1 \ldots d$
  let $i_j = 1$
let $p = 1$
while $i_d \leq N_l$
  let $i_p = i_p + 1$
  if $i_p > N_l$
    let $i_p = 1$
    let $p = p + 1$
  else
    evaluate function at the point $x_{li_1}, \ldots, x_{li_d}$
    multiply the function value with the weight $w_{li_1} \ldots w_{li_d}$
    let $p = 1$
```

If as a univariate quadrature formula the Clenshaw-Curtis formula or Gauss formulas is used, a convergence rate of

$$\varepsilon(N) = O(N^{-r/d}). \tag{7.40}$$

is attained for integrands from C^r (i.e., functions with bounded total derivatives up to order r), see, e.g., [19, 121]. Here, the curse of dimension is visible in the convergence rate. The higher the dimension d, the slower the convergence.

7.5.2 Monte Carlo Methods

In the Monte Carlo method, the integrand is evaluated at (uniformly distributed) random points in the unit hypercube and the integral value is computed as the average of the function values at these points, i.e.

$$\int_{[0,1]^d} f(\mathbf{x})\, d\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i). \tag{7.41}$$

Due to the law of large numbers, the Monte Carlo method converges against the integral value in the statistical average if the number of points N goes to ∞. The convergence is slow, however, and given by

$$\varepsilon(N) = O(N^{-1/2}) \tag{7.42}$$

This means that 100 times more function evaluations are required in order to obtain one more digit accuracy.

7.5.3 Quasi-Monte Carlo Methods

In so-called Quasi-Monte Carlo methods, the integrand is not evaluated at random points but at deterministic ones. The same averaging as with Monte Carlo methods is applied, i.e.

$$\int_{[0,1]^d} f(\mathbf{x})\, d\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i). \tag{7.43}$$

Thereby, low discrepancy point sets are used. In the one-dimensional case, one of these point sets is the Van-der-Corput sequence. Here, the i-th point x_i is generated by writing the number i in basis p (where p is prime)

$$i = \sum_{k=0}^{j} d_k b^k, \tag{7.44}$$

where $d_k \in \{0, \dots, b-1\}$ are the digits of the number representation. Then, the point x_i is defined as the radical inverse (the reflection at the decimal point) of the number i

$$x_i = \sum_{k=0}^{j} d_k b^{-k-1}. \tag{7.45}$$

The first Van-der-Corput points in basis 3 are for example $0, \frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}, \frac{1}{27}$. The corresponding incremental algorithm for the generation of the sequence is:

Algorithm 7.5.2 (Van-der-Corput Sequence)

```
$x = 0$
$for\ i = 1 \dots N$
    $z = 1 - x$
    $v = 1/p$
    $while\ z < v + EPS$
        $v = v/p$
    $x = x + (p+1) * v - 1$
```

Thereby, EPS is a small number, e.g. 10^{-11}. In the multivariate case, different constructions have been proposed such as Halton, Faure, Sobol and Niederreiter sequences [93]. In the Halton sequence, for example, in each direction a Van-der-Corput sequence with a different prime basis is used. The first points of the two-dimensional Halton sequence with prime bases 2 and 3 are $(0,0), (\frac{1}{2}, \frac{1}{3}), (\frac{1}{4}, \frac{2}{3}), (\frac{3}{4}, \frac{1}{9}), (\frac{1}{8}, \frac{4}{9})$. Another type of quasi-Monte Carlo methods are so-called lattice rules [117].

Quasi-Monte Carlo methods have a deterministic convergence rate of

$$\varepsilon(N) = O(N^{-1}(\log N)^{d-1}) \tag{7.46}$$

which is (for fixed d) half an order better than for Monte Carlo methods.

7.6 Path Discretization

For path-dependent options such as Asian options, barrier options or lookback options it is first necessary to simulate the path of the asset.

7.6.1 Random Walk

This is in the simplest case possible using a random walk which can be written as

$$S(t_j + \Delta t) = S(t_j) e^{(r - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} z_j} \tag{7.47}$$

for $j = 0 \dots M-1$, where the z_j are $N(0,1)$ normally distributed random numbers.

In deterministic quadrature methods such as the product approach or Quasi-Monte Carlo methods, instead of random numbers the abscissas of the quadrature formula after transformation with the inverse normal distribution are used.

This way, the price of path-dependent options can be determined by Algorithm 7.6.1.

Algorithm 7.6.1 (Option Pricing by Simulation)

```
set $y = 0$
for $i = 1 \dots N$
   for $j = 1 \dots M$
      draw a $[0,1]$ uniformly distributed random number$x_i$
      transform the random number by the inverse normal distribution: $z_j = N^{-1}(x_j)$
      compute the asset price $S(t_j)$
   determine the option price $V(S,T)$ from the simulated prices
   set $y = y + V(S,T)$
the option price is then the arithmetic mean after discounting $V(S,0) = e^{-rt}y/N$
```

7.6.2 Brownian Bridge

Alternatively, the asset price path can be discretized hierarchically using the so-called Brownian bridge. Thereby, the asset prices are not generated incrementally using the prices in the previous time steps. Instead, a past and a future price is used, i.e.

$$S(t_j + k\Delta t) = \frac{S(t_j) + S(t_j + 2k\Delta t)}{2} e^{(r - \frac{1}{2}\sigma^2)k\Delta t + \sigma\sqrt{k\Delta t/2} z_j} \tag{7.48}$$

This way, first the price at time T is determined. Then, $S(T/2)$ is computed from $S(0)$ and $S(T)$. Afterwards, $S(T/4)$ from $S(0)$ and $S(T/2)$, and $S(3T/4)$ from $S(T/2)$ and $S(T)$, and so on. For simplicity, we want to assume that M is a power of two.

The advantage of this construction is that now the random numbers have a variance of different size. While the first refinement levels show a larger variance than for the random walk discretization, it becomes smaller for finer discretization levels. Of course the total variance is the same for both cases. Numerical methods, however, can use the decay of the variance and weight the time steps differently. For Quasi-Monte Carlo methods, the Brownian bridge construction usually leads to an acceleration of convergence. For example, the components of the Halton sequence have better equal distribution properties for small primes than for larger primes.

Chapter 8

Sparse Grids

Sparse grids are a fairly general method for the efficient numerical treatment of multivariate problems. Besides numerical integration, sparse grids have been applied to the solution of elliptic, parabolic and hyperbolic PDEs [12, 56, 60, 96, 133], SDEs [111], integral equations [58], eigenvalue problems [33], interpolation and approximation [20, 122], Fourier and wavelet analysis [61, 119], global optimization [94], data and image compression [44] and data mining [32, 34, 35, 36, 37].

It goes (at least) back to the Russian mathematician Smolyak [118] and is nowadays known under different names such as (discrete) blending method [55] or Boolean method [20]. It has been applied to numerical integration by several authors using the midpoint rule [4], the rectangle rule [99], the trapezoidal rule [8], the Clenshaw-Curtis rule [95] and the Patterson rule [46] as a one-dimensional basis. Further studies have been made concerning extrapolation methods [8], discrepancy measures [30] and complexity questions [129].

The main idea in sparse grids is a decomposition of the quadrature formula into a telescope sum. To the single terms of the telescope sum, a product approach is applied, however, of all possible combinations only a subset is selected. This selection is done by a balancing of work and accuracy. This way, the dependence of the convergence rate on the dimension is significantly reduced.

We will now briefly illustrate some basic properties about sparse grids and sparse grid quadrature formulas. More information on this subject can be found in the review papers [14, 46].

8.1 Regular Sparse Grids

In the following, we illustrate the sparse grid construction, give an algorithm for its implementation and state error bounds.

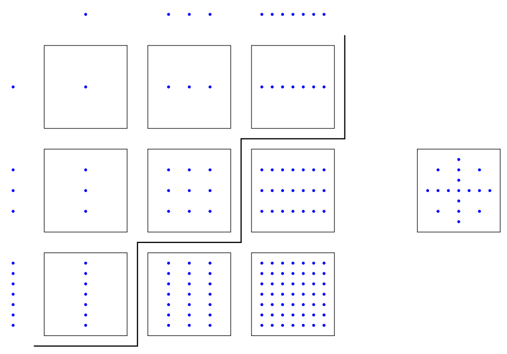


Figure 8.1: Left are the grid points of the trapezoidal sum for $l = 1, 2, 3$ in x- and y-direction as well as the corresponding product grids $\Delta_{k_1} \otimes \Delta_{k_2}$ for $1 \leq k_1, k_2 \leq 3$. Right is the corresponding sparse grid $Q_3^{(2)}$.

8.1.1 Basic Construction

The sparse grid construction starts with a series of one-dimensional quadrature formulas for a univariate function f,

$$Q_l^{(1)} f := \sum_{i=1}^{N_l} w_{li} \cdot f(x_{li}). \tag{8.1}$$

Now, to construct the telescope sum, define the difference formulas by

$$\begin{aligned} \Delta_k^{(1)} f &:= (Q_k^{(1)} - Q_{k-1}^{(1)}) f \quad \text{with} \\ Q_0^{(1)} f &:= 0. \end{aligned}$$

Here, the differences $\Delta_k^{(1)} f$ are again (univariate) quadrature formulas. In the case that the initial formulas $Q_k^{(1)} f$ are nested (such as for the quadrature formulas of section 7.4), the difference formulas use the same grid points, only with different weights. The new weights are nothing else but the differences of the weights of the initial formulas between successive levels.

The regular sparse grid method for d-dimensional functions f is then for a given level $l \in \mathbb{N}$ and $\mathbf{k} \in \mathbb{N}^d$

$$Q_l^{(d)} f := \sum_{|\mathbf{k}| \leq l+d-1} (\Delta_{k_1}^{(1)} \otimes \ldots \otimes \Delta_{k_d}^{(1)}) f. \tag{8.2}$$

Hereby, all possible tensor products of the difference formulas are considered. Of all these possibilities only those are used whose sum of indices is smaller than a constant (here $l + d - 1$). Let us remark that the product approach from section 7.5.1 is characterized by using all valid indices, i.e.

$$Q_l^{(d)} f = \sum_{\max\{k_1,\ldots,k_d\} \leq l} (\Delta_{k_1}^{(1)} \otimes \ldots \otimes \Delta_{k_d}^{(1)}) f. \tag{8.3}$$

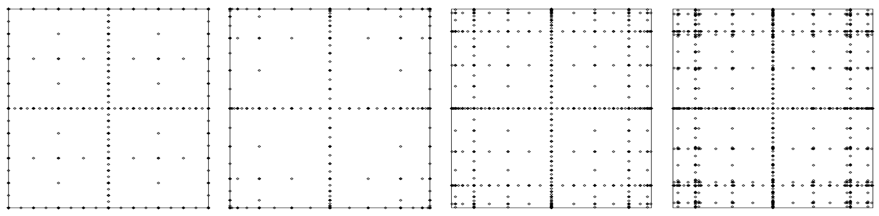


Figure 8.2: Examples for two-dimensional sparse grids based on the trapezoidal, Clenshaw-Curtis, Gauß-Patterson, and Gauß-Legendre rules ($l = 6$).

Visually, the product approach corresponds to a summation over a discrete hypercube with edge length l, i.e., $\max\{k_1, \ldots, k_d\} \leq l$ while the sparse grid method sums over the simplex with edge length l, i.e., $k_1 + \ldots + k_d \leq l + d - 1$. In Figure 8.1 the construction is shown using a two-dimensional example.

Figure 8.2 shows several two-dimensional classical sparse grids based on different univariate quadrature rules.

8.1.2 Implementation

While the programming of the single tensor products in formula (15) can be realized in the same way as for the product approach (see Algorithm 7.5.1) the programming of the sum over the simplex seems at first sight difficult for general d.

Algorithm 8.1.1 (Sparse Grid Quadrature)

```
for $j = 1 \ldots d$
  let $k_j = 1$
  let $\hat{k}_j = l$
let $q = 1$
let $k_1 = l - 1$
while $k_d \leq l$
  let $k_q = k_q + 1$
  if $k_q > \hat{k}_q$
    let $k_q = 1$
    let $q = q + 1$
  else
    for $r = 1 \ldots q - 1$
      let $\hat{k}_r = \hat{k}_q - k_q$
    let $k_1 = \hat{k}_1$
    integrate grid for $k_1 \ldots k_d$ using Algorithm 7.5.1
    let $q = 1$
```

This problem can be solved by an analogous drop algorithm. Thereby, an additional vector $\hat{k}$ which contains the currently valid maximal values and which is updated with time is used. The procedure is illustrated in Algorithm 8.1.1.

8.1.3 Error Bounds

We will now come to the integration error of sparse grid quadrature formulas. Let us therefore consider the class of functions $\mathcal{W}_d^r$ with bounded mixed derivatives of order r,

$$\mathcal{W}_d^r := \left\{ f : \Omega \longrightarrow \mathbb{R}, \ \left\| \frac{\partial^{|\mathbf{s}|_1} f}{\partial x_1^{s_1} \dots \partial x_d^{s_d}} \right\|_\infty < \infty, \ s_i \leq r \right\}. \tag{8.4}$$

Now, let us assume that the underlying one-dimensional quadrature formula satisfies the error bound

$$|I^{(1)} - Q_l^{(1)} f| = O((N_l)^{-r}). \tag{8.5}$$

for functions $f \in \mathcal{W}_1^r$. This bound holds, for example, for all interpolatory quadrature formulas with positive weights, such as the Clenshaw-Curtis, Gauß-Patterson and Gauß-Legendre formulas. Taking one such quadrature formula as one-dimensional basis and assuming $N_l = O(2^l)$, the error of the classical sparse grid quadrature formula is of order

$$|I^{(d)} - Q_l^{(d)} f| = O(N^{-r} (\log N)^{(d-1)(r+1)}), \tag{8.6}$$

for $f \in \mathcal{W}_d^r$ where N is the number of sparse grid points, see [95, 129]. We see that the convergence rate depends only weakly on the dimension but strongly on the smoothness r. Unfortunately, little is known about error bounds for more general sparse grid constructions and different function spaces than $\mathcal{W}_d^r$. See [103, 130] for recent developments in this direction.

8.2 Dimension-Adaptive Sparse Grids

Despite the large improvements of the Quasi-Monte Carlo and sparse grid methods over the Monte Carlo method, their convergence rates will suffer more and more with rising dimension due to their respective dependence on the dimension in the logarithmic terms. Therefore, one aim of recent numerical approaches has been to reduce the dimension of the integration problem without (too great) affection of the accuracy.

In some applications, the different dimensions of the integration problem are not equally important. For example, in path integrals the number of dimensions corresponds to the number of time-steps in the time discretization. Typically the first steps in the discretization are more important than the last steps since they determine the outcome more substantially. In other applications, although the dimensions seem to be of the same importance at first sight, the problem can be transformed into an equivalent one where the dimensions are not. Examples are the Brownian bridge discretization or the Karhunen-Loeve decomposition of stochastic processes.

Intuitively, problems where the different dimensions are not of equal importance might be easier to solve. Numerical methods could concentrate on the more important dimensions and spend more work for these dimensions than for the unimportant ones. Interestingly, also complexity theory reveals that integration problems with weighted dimensions can become tractable even if the unweighted problem is not [130]. Unfortunately, classical adaptive numerical integration algorithms [42, 124] cannot be applied to high-dimensional problems since the work overhead in order to find and adaptively refine in important dimensions would be too large.

To this end, a variety of algorithms have been developed which try to find and quantify important dimensions. Often, the starting point of these algorithms is Kolmogorov's superposition theorem [78, 79]. Here, a high-dimensional function is approximated by sums of lower-dimensional functions. A survey of this approach from the point of approximation theory is given in [75]. Further results can be found in [106, 114]. Analogous ideas are followed in statistics for regression problems and density estimation. Here, examples are so-called additive models [64], multivariate adaptive regression splines (MARS) [31], and the ANOVA decomposition [126, 132], see also [68]. Other interesting techniques for dimension reduction are presented in [65].

In case the importance of the dimensions is known a priori, techniques such as importance sampling can be applied in Monte Carlo methods [73]. For the Quasi-Monte Carlo method already a sorting of the dimensions according to their importance leads to a better convergence rate (yielding a reduction of the effective dimension). The reason for this is the better distributional behaviour of low discrepancy sequences in lower dimensions than in higher ones [15]. The sparse grid method, however, a priori treats all dimensions equally and thus gains no immediate advantage for problems where dimensions are of different importance.

The aim of this section is to develop a generalization of the conventional sparse grid approach [118] which is able to adaptively assess the dimensions according to their importance and thus reduces the dependence of the computational complexity on the dimension. The dimension-adaptive algorithm tries to find important dimensions automatically and adapts (places more integration points) in those dimensions. To achieve this efficiently, a data structure for a fast bookkeeping and searching of generalized sparse grid index sets is proposed as well.

8.2.1 Dimension-Adaptive Refinement

In order to be able to assess the dimensions differently, it is necessary to modify the original sparse grid construction [47]. Note that conventional adaptive sparse grid approaches [8, 12, 22] merely tackle a locally non-smooth behaviour of the integrand function and usually cannot be applied to high-dimensional problems.

The most straightforward way to generalize the conventional sparse grid with respect to differently important dimensions is to consider a different index set than the unit simplex $|\mathbf{k}|_1 \le l+d-1$. For example, one could consider the class of general simplices $\mathbf{a}\cdot\mathbf{k} \le l+d-1$ where $\mathbf{a} \in \mathbb{R}_+^d$ is a weight vector for the different dimensions [35, 46, 109]. A static strategy would be to analyze the problem and then to choose a suitable vector $\mathbf{a}$. Such a strategy has two drawbacks, though. First, it is hard to a-priori choose the optimal (or, at least, a good) weight vector $\mathbf{a}$, and second, the class of general simplices itself may be inadequate for the problem at hand (e.g. more or less points in mixed directions may be required).

Instead, we will allow more general index sets [67, 105, 130] in the summation of (8.8) and try to choose them properly. To this end, we will consider the selection of the whole index set as an optimization problem, i.e. as a binary knapsack problem [13, 57], which is closely related to best N-term approximation [21]. A self-adaptive algorithm can try to find the optimum index set in an iterative procedure. However, not all index sets are admissible in the generalized sparse grid construction and special care has to be taken during the selection of indices, as we will see.

In the following, we will take a look at the general sparse grid construction and at the required conditions on the index set. After that, we will present the basic iterative algorithm for the selection of an appropriate index set. Then, we will address the important issue of error estimation.

8.2.2 Generalized Sparse Grids

We will start with the admissibility condition on the index set for the generalized sparse grid construction. An index set $\mathcal{I}$ is called *admissible* if for all $\mathbf{k} \in \mathcal{I}$,

$$\mathbf{k} - \mathbf{e}_j \in \mathcal{I} \quad \text{for} \quad 1 \le j \le d,\ k_j > 1, \tag{8.7}$$

holds. Here, $\mathbf{e}_j$ is the j-th unit vector. In other words, an admissible index set contains for every index $\mathbf{k}$ all indices which have smaller entries than $\mathbf{k}$ in at least one dimension. Note that the admissibility condition on the index set ensures the validity of the telescope sum expansion of the general sparse grid quadrature formulas using the difference formulas $\Delta^1_{k_j}$.

Now we are able to define the *general sparse grid construction* [46]:

$$Q_{\mathcal{I}}^{(d)} f^{(d)} := \sum_{\mathbf{k}\in\mathcal{I}} (\Delta_{k_1} \otimes \ldots \otimes \Delta_{k_d}) f^{(d)}, \tag{8.8}$$

for an admissible index set $\mathcal{I} \in \mathbb{N}^d$.

Note that this general sparse grid construction includes conventional sparse grids ($\mathcal{I} = \{\mathbf{k} : |\mathbf{k}|_1 \le l+d-1\}$) as well as classical product formulas ($\mathcal{I} = \{\mathbf{k} : \max\{k_1, \ldots, k_d\} \le l\}$) as special cases. Unfortunately, little is known about error bounds of quadrature formulas associated to general index sets $\mathcal{I}$ (see [105, 130]). However, by a careful construction of the index sets $\mathcal{I}$ we can hope that the error for generalized sparse grid quadrature formulas is at least as good as in the case of conventional sparse grids. Furthermore, the algorithm allows for an adaptive detection of the important dimensions.

8.2.3 Basic Algorithm

Our goal is now to find an admissible index set such that the corresponding integration error ε is as small as possible for a given amount of work (function evaluations). The procedure starts with the one-element index set $\{\mathbf{1}\}, \mathbf{1} = (1, \dots 1)$ and adds indices successively such that

- the resulting index sets remain admissible, and
- possibly a large error reduction is achieved.

To this end, an estimated error $g_{\mathbf{k}}$ called *error indicator* is assigned to each index $\mathbf{k}$ which is computed from the differential integral

$$\Delta_{\mathbf{k}} f^{(d)} = (\Delta_{k_1} \otimes \cdots \otimes \Delta_{k_d}) f^{(d)} \tag{8.9}$$

and from further values attributed to the index $\mathbf{k}$ like the work involved for the computation of $\Delta_{\mathbf{k}} f$. Let us remark here that the exact integration error is unknown since the integrand itself is unknown. We will address error estimation afterwards.

In our algorithm always the index with the largest error indicator is added to the index set. Once an index is added, its forward neighbourhood is scanned for new admissible indices and their error indicators are computed. Here, the *forward neighbourhood* of an index $\mathbf{k}$ is defined as the d indices $\{\mathbf{k} + \mathbf{e}_j,\ 1 \le j \le d\}$. Conversely, the *backward neighbourhood* is defined by $\{\mathbf{k} - \mathbf{e}_j,\ 1 \le j \le d\}$. Altogether, we hope to heuristically build up an optimal index set in the sense of [13, 57] or [21] this way.

Recall that the computed total integral is just the sum over all differential integrals within the actual index set $\mathcal{I}$. Now as soon as the error indicator for a new index is computed, the index can in fact already be added to the index set since it does not make sense to exclude the just computed differential integral from the total integral. Therefore, when the error indicator of an index is computed, the index is put into the index set $\mathcal{I}$ (but its forward neighbours in turn are currently not considered).

To this end, we partition the current index set $\mathcal{I}$ into two disjoint sets, called *active* and *old* indices. The active index set $\mathcal{A}$ contains those indices of $\mathcal{I}$ whose error indicators have been computed but the error indicators of all their forward neighbours have not yet been considered. The old index set $\mathcal{O}$ contains all the other indices of the current index set $\mathcal{I}$. The error indicators associated with the indices in the set $\mathcal{A}$ act as an estimate $\eta = \sum_{\mathbf{i} \in \mathcal{A}} g_{\mathbf{i}}$ for the global error.

Now, in each iterative step of the dimension-adaptive algorithm the following actions are taken: The index with the largest associated error indicator is selected from the active index set and put into the old index set. Its associated error is subtracted from the

Algorithm 8.2.1 (Dimension-Adaptive Quadrature)

```
$\mathbf{i} := (1, \ldots, 1)$
$\mathcal{O} := \emptyset$
$\mathcal{A} := \{\mathbf{i}\}$
$r := \Delta_{\mathbf{i}} f$
$\eta := g_{\mathbf{i}}$
while ($\eta >$ TOL) do
    select $\mathbf{i}$ from $\mathcal{A}$ with largest $g_{\mathbf{i}}$
    $\mathcal{A} := \mathcal{A} \setminus \{\mathbf{i}\}$
    $\mathcal{O} := \mathcal{O} \cup \{\mathbf{i}\}$
    $\eta := \eta - g_{\mathbf{i}}$
    for $k := 1, \ldots, d$ do
        $\mathbf{j} := \mathbf{i} + \mathbf{e}_k$
        if $\mathbf{j} - \mathbf{e}_q \in \mathcal{O}$ for all $q = 1, \ldots, d$ then
            $\mathcal{A} := \mathcal{A} \cup \{\mathbf{j}\}$
            $s := \Delta_{\mathbf{j}} f$
            $r := r + s$
            $\eta := \eta + g_{\mathbf{j}}$
        endif
    endfor
endwhile
return $r$
```

Symbols:

$\mathcal{O}$	*old index set*
$\mathcal{A}$	*active index set*
$\Delta_{\mathbf{i}} f$	*integral increment* $\bigotimes_{k=1}^{d} \Delta_{i_k} f$
$g_{\mathbf{i}}$	*local error indicator*
η	*global error estimate* $\sum_{\mathbf{i} \in \mathcal{A}} g_{\mathbf{i}}$
$\mathbf{e}_k$	*k-th unit vector*
TOL	*error tolerance*
r	*computed integral value* $\sum_{\mathbf{i} \in \mathcal{O} \cup \mathcal{A}} \bigotimes_{k=1}^{d} \Delta_{i_k} f$

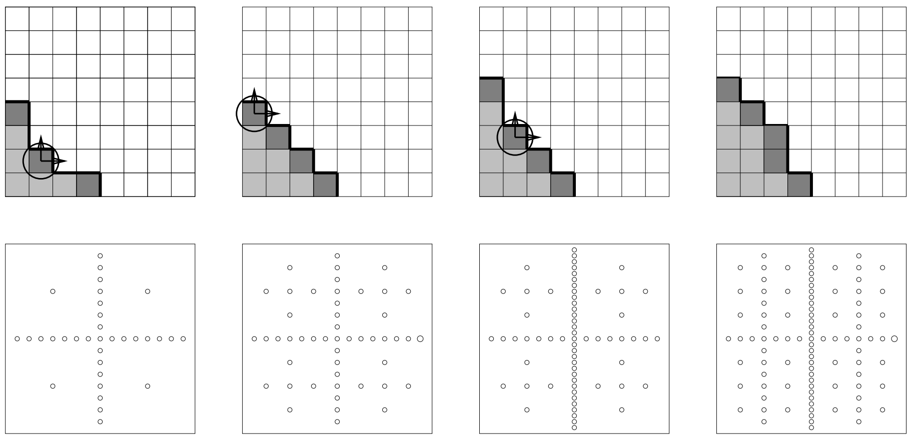


Figure 8.3: A few snapshots of the evolution of the dimension-adaptive algorithm. Shown are the sparse grid index sets (upper row) together with the corresponding sparse grids using the midpoint rule (lower row). Active indices are dark-shaded, old indices are light-shaded. The encircled active indices have the largest error indicators and are thus selected for insertion into the old index set.

global error estimate η. Also, the error indicators of the admissible forward neighbouring indices of this index are computed and their indices are put into the active index set. Accordingly, the corresponding values of the differential integral (8.9) are added to the current quadrature result and the corresponding values of the error indicators are added to the current global error estimate. If either the global error estimate falls below a given threshold or the work count exceeds a given maximal amount, the computation is stopped and the computed integral value is returned. Otherwise, the index with the now largest error is selected, and so on (see Figure 8.2.1).

A two-dimensional example for the operations of the algorithm is shown in Figure 8.3. Whenever an active index is selected and put into the old index set (in this example the indices $(2,2)$, $(1,4)$, and $(2,3)$) its two forward neighbours (indicated by arrows) are considered. If they are admissible, they are inserted in the active index set. In the example the forward neighbour $(2,4)$ of $(1,4)$ is not inserted since it is not admissible (its backward neighbour $(2,3)$ is in the active index set but not in the old index set).

8.2.4 Error Estimation

Error estimation is a crucial part of the algorithm. If the estimated error for a given index $\mathbf{k}$ happens to be very small, then there may be no future adaptive refinement in its forward neighbourhood. Now, this behaviour can be good or bad. If the errors of the forward neighbours of $\mathbf{k}$ are smaller or of the same magnitude as the error of $\mathbf{k}$, then the algorithm has stopped the adaption properly. But, it might be that one or more forward neighbours have a significantly larger error and thus the algorithm should refine there. Unfortunately, there is usually no way to know the actual magnitude beforehand (besides by a close a-priori analysis of the integrand function, which is usually not available). The problem could of course be fixed by actually looking at the forward neighbours and the computation of their error indicators. But, this just puts the problem off since we encounter the same difficulty again with the neighbours of the neighbours.

We will here attack this problem through an additional consideration of the involved work. The number of function evaluations required for the computation of the differential integral (and thus also for the error estimation) for a given index $\mathbf{k}$ is known beforehand. If we assume that the univariate quadrature formulas are nested, then the number of function evaluations $n_{\mathbf{k}}$ related to an index $\mathbf{k}$ is given by

$$n_{\mathbf{k}} := n_{k_1}^{(1)} \cdot \ldots \cdot n_{k_d}^{(1)}, \tag{8.10}$$

and thus can be computed directly from the index vector. Now, in order to avoid a too early stopping it makes sense to consider the forward neighbourhood of an index with a small error if the work involved is small – especially in comparison to the work for the index with the currently largest error. Let us therefore consider a generalized error indicator $g_{\mathbf{k}}$ which depends on both the differential integral and the number of function evaluations,

$$g_{\mathbf{k}} := q(|\Delta_{\mathbf{k}} f|, n_{\mathbf{k}}), \tag{8.11}$$

with a yet to be specified function q which relates these two numbers. Clearly, the function q should be increasing with the first and decreasing with the second argument.

As a possible choice for q we will consider the following class of generalized error estimators

$$g_{\mathbf{k}} = \max\left\{ w \frac{|\Delta_{\mathbf{k}} f|}{|\Delta_{\mathbf{1}} f|}, (1-w)\frac{n_{\mathbf{1}}}{n_{\mathbf{k}}} \right\} \tag{8.12}$$

where $w \in [0,1]$ relates the influence of the error in comparison to the work (we assume that $\Delta_{\mathbf{1}} f \neq 0$; this reference value can also be replaced by a suitable normalizing constant or the maximum of previously computed differential integrals). Let us remark that usually $n_{\mathbf{1}} = 1$.

By selection of $w = 1$ a greedy approach is taken which disregards the second argument i.e. when the function is known to be very smooth (e.g. strictly convex or concave) and thus the error estimates would decay with increasing indices anyway. Classical sparse grids are realized by $w = 0$ and in this case only the involved work is counted. Values of w in between will safeguard against both comparatively too high work and comparatively too small error.

Note that in general we have to assume that the integrand function fulfills a certain *saturation assumption*, compare also [3, 23, 125] for the case of adaptive finite elements. This means that the error indicators roughly decrease with the magnitude of their indices. This condition would not be true for example for functions with spikes on a very fine scale or large local discontinuities. Let us remark here that we believe it impossible to search such spikes or discontinuities in high-dimensional space unless the integrand function has

`unsigned char I[m][d]`	entries of all indices
`int A[m]`	active indices
`int O[m]`	old indices
`double G[m]`	error estimates
`int N[m][2*d]`	neighbours
`int ni`	number of elements in I
`int na`	number of elements in A
`int no`	number of elements in O

Figure 8.4: The data types and memory requirements for the dimension-adaptive algorithm.

special properties (for example, convexity). Note that such functions would practically not be integrable by Monte Carlo and Quasi-Monte Carlo methods as well.

Note furthermore that the global error estimate η typically underestimates the error. But, η and the true integration error ε are proportional to each other if the error indicators decrease with the magnitude of their indices. Therefore, the error tolerance `TOL` is only achieved up to a constant.

The illustrated dimension-adaptive algorithm and error estimation method is not unique. A set of different index refinement and error estimation schemes were developed and compared in [92].

8.2.5 Data Structures

The number of indices in the index sets can become very large for difficult (high-dimensional) problems. For the performance of the overall dimension-adaptive algorithm it is necessary to store the indices in such a way that the operations required by the algorithm can be performed efficiently.

In view of section 8.2.1 these operations are

- to insert and remove indices from the active index set $\mathcal{A}$,
- to insert indices into the old index set $\mathcal{O}$,
- to find the index in the active index set with the largest error,
- to check if an index is admissible.

In this section we will describe the data structures which allow a fast execution of these operations. We will use relative addressing for the storage of the indices, a heap for the active indices, a linear array for the old indices, and linked neighbour lists for the admissibility check.

Relative Addressing

In contrast to classical numerical algorithms the dimension d of the problem at hand is highly variable and cannot be neglected in the space and time complexity of the algorithm. In application problems this dimension can readily range up to 1000 and, for example, already a cubic dependence on the dimension can render an algorithm impractical.

This easily overlooked problem becomes visible when for example a multi-index of dimension d has to be copied to a different memory location or when two indices have to be checked for identity. A straightforward approach would require $O(d)$ operations (to copy or compare all the single elements). If these operations are performed within an inner loop of the algorithm, complexities multiply and the total dependence on d is increased.

Therefore, we use relative addressing here. We allocate one two-dimensional array `I` for all (active and old) indices which contains the elements of the current index set $\mathcal{I} = \mathcal{A} \cup \mathcal{O}$. This array has dimension $m \times d$ where m is the maximum number of generated indices. The size m can be chosen statically as the maximum amount of memory available (or that one is willing to spend). Alternatively, m can be determined dynamically and the whole array is reallocated (e.g. with size $2m$) when the current number of elements denoted by `ni` exceeds the available space. One byte per index element is sufficient for the storage. Indices which are newly generated (i.e. as a forward neighbour of a previously generated active index) are inserted successively. Indices are never moved within the array or removed from the array.

For the description of the active and old index sets ($\mathcal{A}$ and $\mathcal{O}$) we use one-dimensional arrays `A` and `O` of maximum size m, respectively. Each entry in these arrays is the position of the corresponding index in the array `I`. In addition, the current number of indices in `A` and `O` denoted by `na` and `no` are stored (see Figure 8.4). Now, when an index is copied from `A` to `O`, only the entry to `I` has to be copied and not all its d elements. This way, the total dependence on d of the algorithm is reduced.

Active Indices

So far we have not illustrated how the indices in `A` and `O` are stored. The required operations on the active and old index sets are quite different and therefore, we will arrange the two sets differently.

Let us first look at the set of active indices. The necessary operations are fast insertion and removal of indices. Furthermore, we have to be able to find the index with the largest associated error indicator. For the latter operation one clearly does not want to search through all the indices in order to find the current maximum every time (which would lead to a quadratic work complexity in the number of indices).

Let us first remark that we store the error indicators in an additional floating point array `G` of size m (with the same numbering as `I`, see Figure 8.4). We will here use a (at least in the computer science literature) well-known data structure called *heap* [112] which supports

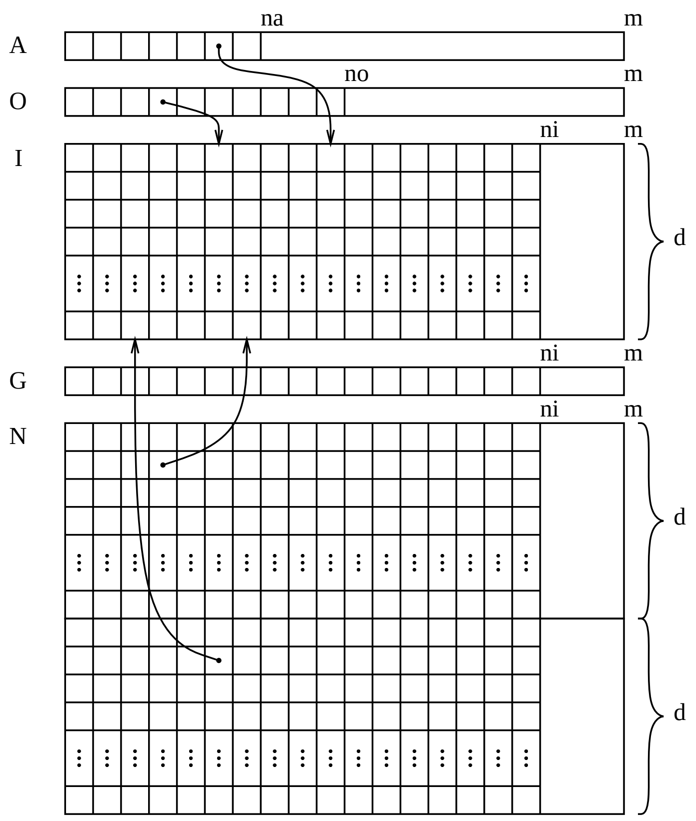


Figure 8.5: A schematic representation of the data structures. Shown are the arrays for the active and old indices `A` and `O`, the index elements `I`, the error estimates `G`, and the neighbours `N`.

the required operations in the most efficient way. A heap is an ordering of the indices in `A` such that the error indicator for an index at position p is greater than (or equal to) the error indicators of the indices at positions $2p$ and $2p+1$. This way, a binary tree hierarchy is formed on the set of indices where the index at the root (position 1) has the largest error indicator.

When the root index is removed (i.e. by putting it into the old index set), then the one of the two sons (at positions 2 and 3) with the larger error indicator is promoted as the new root. The vacancy that arises this way is filled with the son which possesses the larger error indicator and this scheme is repeated recursively until the bottom of the tree is reached.

Old Indices

Similarly, when a new index is inserted (i.e. as the forward neighbour of the just removed index) it is first placed at the last position in the tree (i.e. it is assigned the highest position

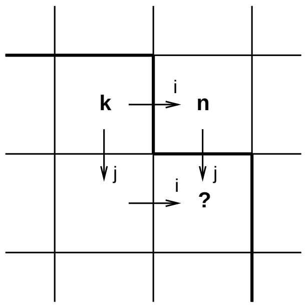


Figure 8.6: Index $\mathbf{n}$ has just been generated as the forward neighbour in direction i of index $\mathbf{k}$. The backward neighbour of $\mathbf{n}$ in direction $j \neq i$ can be found as the forward neighbour in direction i of the backward neighbour in direction j of $\mathbf{k}$.

`na+1`). Now, if the error indicator of the father of the new index is smaller than its own error indicator, then the two positions are swapped. This procedure is repeated recursively until the error indicator of the current father is larger than that of the new index. This way, insertion and removal are functions which can all be performed in $O(\log(\texttt{na}))$ operations.

The required operations on the old index set are the insertion of indices and the checking if an index is admissible. Since indices are never removed from the old index set, the indices are stored in order of occurrence and insertion is simply done at the end of `O` (at position `no+1`). The check for admissibility is more difficult, though, since it involves the location of the whole backward neighbourhood of a given index. To this end, we explicitly store all the neighbours. For every index in both the active and old index sets the positions in `I` of the d forward neighbours and the d backward neighbours are stored. This requires an array `N` of size $m \times 2d$ where the first d entries are the forward and the second d entries the backward neighbours (see Figure 8.4). Note that the indices in I themselves already require $m \cdot d$ bytes. Thus, the overhead for the new array is not large. Note also that indices in the active index have only backward neighbours.

Now, let us discuss how the neighbour array is filled. Let us assume that a new index is generated as the forward neighbour of an active index $\mathbf{k}$ in direction i. The backward neighbour of the new index in direction i is known (the previously active index $\mathbf{k}$), but the $d-1$ other backward neighbours are unknown. Let us consider the backward neighbour in direction $j \neq i$. This backward neighbour can be found as the forward neighbour in direction i of the backward neighbour in direction j of the previously active index (see Figure 8.6). Put differently,

$$\bar{p}_j(p_i(\mathbf{k})) = p_i(\bar{p}_j(\mathbf{k})), \tag{8.13}$$

where p_i is the forward neighbour in direction i and $\bar{p}_j$ is the backward neighbour in direction j. In turn, when the backward neighbour in direction j is found, the new index is stored as its forward neighbour in direction j. This way, all required forward neighbours can be found and stored in the data structure. In summary, the construction of the neighbour array is done in constant additional time.

A new index is admissible if all backwards neighbours are in the array `O`. Indices in `O` can be distinguished from indices in `N` e.g. by looking at the first forward neighbour. Recall that indices in N do not have any forward neighbours and thus a marker (e.g. -1) may be used to identify them without additional storage. In summary, the admissibility check can now be performed in $O(d)$ operations.

8.2.6 Complexities

We will now discuss the space and time complexities of the algorithm. Concerning the time complexity we will distinguish between the work involved for the computation of the integral and the work overhead for the bookkeeping of the indices.

The memory requirement of the data types of Figure 3 is $(9d + 16)m + 12$ bytes. Additionally, the nodes and weights of the univariate quadrature formulas have to be stored (if they cannot be computed on-the-fly). This storage, however, can usually be neglected. In our experience, 257 quadrature nodes have proved to be more than enough for typical high-dimensional problems. In summary, the required memory is $O(d \cdot m)$ bytes (with a constant of about 9).

The amount of work required for $\Delta_{\mathbf{k}} f$ is $c \cdot n_{\mathbf{k}}$ where c is the cost of a function evaluation (which is at least $O(d)$). However, since the total cost depends on the size and structure of the index set, which is unknown beforehand, bounds for the work required for the function evaluations can in general not be obtained. For the conventional sparse grid of level l, we know that this work is $O(2^l \cdot l^{d-1})$, but we hope that the work for a dimension-adapted grid is substantially smaller (especially concerning the dependence on d).

However, we can tell something about the work overhead for the bookkeeping of the indices. In view of Figure 8.2.1 we see that for each index which is put into `O` two *for* loops (over k and q) of size d are performed. In the outer loop, the new index is put into `A` which requires $O(\log \mathtt{na})$ operations. So, the worst case time complexity for bookkeeping is $O(d^2 + d \log \mathtt{na})$. Note that the average case complexity is smaller since the inner loop can be terminated early. In practice, the total overhead behaves like $O(d^2)$.

8.3 Derivative Pricing using Sparse Grids

As we have seen, many option pricing problems (e.g. path- and performance-dependent options) require the computation of multivariate integrals. The dimension of these integrals is determined by the number of independent stochastic factors (e.g. the number of time steps in the time discretization or the number of assets under consideration). The high dimension of these integrals can be treated with dimension-adaptive quadrature methods as illustrated in the previous section.

However, the integrand has typically discontinuous first derivatives, which heavily degrades the performance of quadrature formulas. We will here show an approach which can also

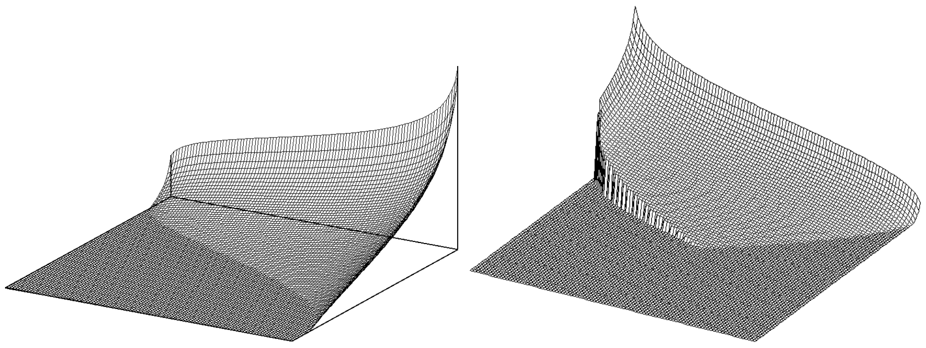


Figure 8.7: Integrands of the path- (left) and the performance-dependent option (right) for two-dimensional problems.

handle this problem efficiently. The main idea is to find lines are areas of discontinuity and to employ suitable transformations of the integration domain. This way, integration takes only place over the smooth parts of the integrand and the fast convergence of the sparse grid method can be regained.

8.3.1 Transformation

For options we have the following problem: the payoff function is not smooth due to the nature of the option. This is caused by the fact that the holder would not exercise the option if a purchase or sale of the underlying asset would lead to a loss. Of course, the discontinuity of the payoff function carries over to the integrand. Examples for such integrands in two dimensions after transformation to $[0,1]^2$ are shown in Figure 8.7. The integrand shows a kink (path-dependent option) with respect to a $(M \cdot N - 1)$-dimensional manifold or even a jump (performance-dependent option) at such a manifold. Since some (mixed) derivatives are not bounded at these manifolds, the smoothness requirements for the sparse grid method are clearly not fulfilled any more.

We will therefore decompose the integration domain into areas where the integrand function is smooth. Folds and jumps will be located at the boundary of these areas. The sparse grid quadrature formulas are mapped onto the areas with the help of suitable transformations. The total integral is then computed as sum of these separate integrals. This way, only smooth functions are integrated and the positive properties of the sparse grid methods are regained.

In order to find the kinks and jumps it suffices to compute the zeros of the integrand function. Using iterated integration, the zero finding is restricted to only one (the last) dimension. We therefore determine the zero $\hat{x}$ (Newton's or Brent's method for a kink and bisection for a jump) and transform the integrand with respect to the last dimension using the linear mapping $t(x) = x \cdot (1 - \hat{x}) + \hat{x}$ onto $[0,1]$. This way, the integration domain is topologically still a hypercube. This yields sparse grids shown in Figure 8.8.

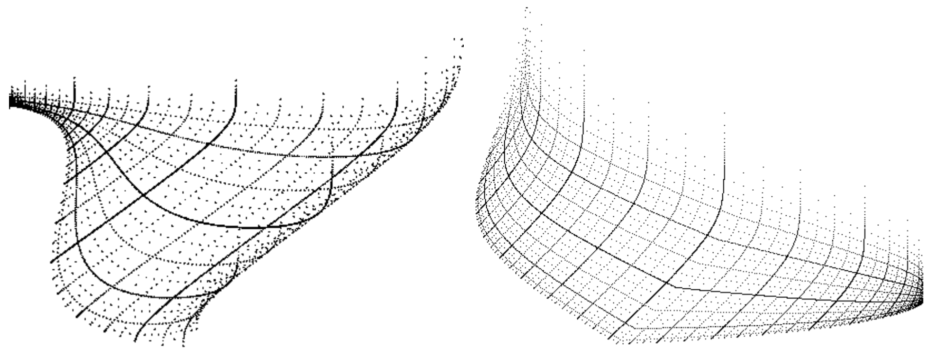


Figure 8.8: Sparse grids after zero finding and transformation for the path- (left) and the performance-dependent option (right).

8.3.2 Integration of Multivariate Normal Distributions

But also the integration of smooth multivariate normal distributions by sparse grids is involved. The efficient application of the formulas from Theorem 5.4.3 and 5.4.7 crucially depends on the availability of accurate and fast numerical methods for the evaluation of multivariate normal probabilities. For small dimensions $d \leq 3$ there is reliable and efficient software, see e.g. [24, 41, 110]. For larger dimensions, standard multivariate numerical integration software, like ADAPT [42] or DCUHRE [6], can be applied but their accuracy usually suffers from the fact that the infinite integration limits need to be transformed or cut off. Moreover, they do not take advantage of the special form of the integrand.

Instead, Genz [39] proposed a simple sequence of transformations of the multivariate normal distribution function which reduces the dimension by one and places the problem to the unit square. One obtains

$$\Phi(\mathbf{A}, \mathbf{b}) = e_1 \int_{[0,1]^{d-1}} \prod_{i=2}^{d} e_i(w_1, \ldots, w_{i-1}) \, d\mathbf{w} \tag{8.14}$$

with

$$e_i(w_1, \ldots, w_{i-1}) = \Phi((b_i - \sum_{j=1}^{i-1} c_{ij} \Phi^{-1}(w_j \, e_j))/c_{ii}) \tag{8.15}$$

where $\Phi(x)$ denotes the standard univariate normal distribution function and c_{ij} the entries of the Cholesky decomposition $\mathbf{CC}^T$ of the matrix $\mathbf{A}$.

This way, the convergence of standard numerical integration software can be significantly accelerated. Usually, the computation time can be further reduced if the variables are reordered such that the variables associated with the largest integration intervals are the innermost variables. The standard univariate normal distribution function and its inverse can efficiently and up to high accuracy be computed by the Moro [90] scheme.

Chapter 9

Numerical Results

In this chapter, we apply the various illustrated methods and algorithms of the previous chapters to application problems in computational finance. Thereby, we compute fair values of path-dependent, interest-rate and performance-dependent derivatives and compare the efficiency of different pricing approaches. All computations were performed on a dual Intel(R) Xeon(TM) CPU 3.06GHz processor.

9.1 Example Problem

We will start this section with a small but illustrative example. Let us consider the following integration problem (which is, albeit simplified, the core of many derivative security pricing problems):

$$I^d f = \int_{\mathbb{R}^d} e^{-\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b}} \, d\mathbf{x}. \tag{9.1}$$

We here choose as dimension $d = 10$, $\mathbf{A} = \mathbf{I}$ and $\mathbf{b}$ randomly in $[-1, 1]$. The integrand is transformed to $[0, 1]^d$ using the inverse normal distribution function (see section 7.4.4). In Figure 9.1 we plot the computing time vs. the number of correct digits for the product Gauss rule, Monte Carlo, Quasi-Monte Carlo (using the Halton sequence), and the classical sparse grid Gauss-Patterson method.

We observe convergence rates of 0.25, 0.5, 1.0, and 2.5 respectively. Perhaps more dramatically, a 5-digit accuracy would require 0.02 seconds for the sparse grid method in comparison to 300 years for the product, 100 seconds for the Monte Carlo, and still 1 second for the Quasi-Monte Carlo approach.

This example shows that already the classical sparse grid approach will outperform the other methods if the integrand is smooth and if the dimension of the problem is not too large.

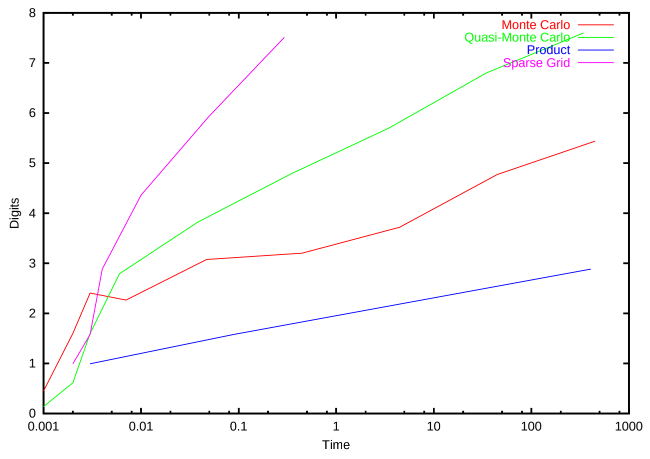


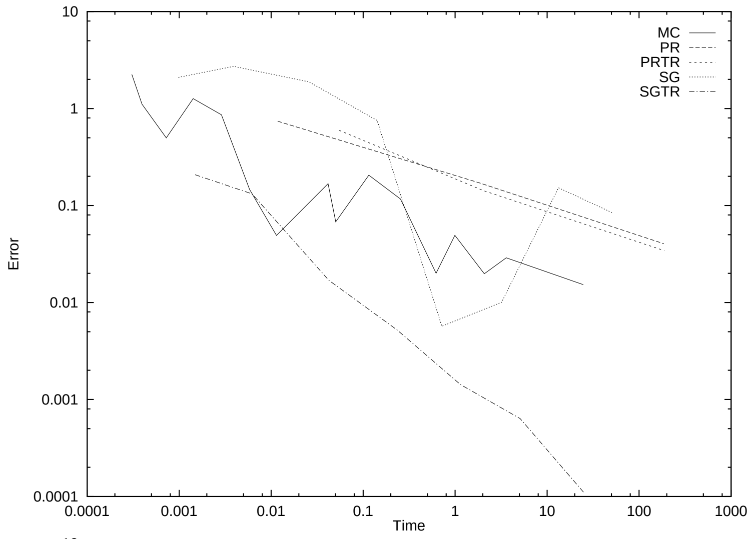
Figure 9.1: Performance comparison of different quadrature methods for a smooth exponential function.

9.2 Path-Dependent Derivatives

Next, we will consider two types of path-dependent derivatives, the geometric average Asian option of section 5.3.1 and the down-out barrier option of section 5.3.2. Both types of options have closed-form solutions. This enables us to exactly specify the quadrature errors. As already mentioned, slight changes in the type of option lead to problems where no longer a closed-form solution exists. Then, numerical integration is the only way to obtain approximate option prices.

In both cases, the parameters are $S(0) = 100, K = 100, \sigma = 0.2, r = 0.05$. The number of time steps is $M = 8$, the barrier for the down out option is set to $K = 90$. The asset prices are simulated using a random walk here.

In Figure 9.2, a comparison of different integration methods for the pricing of these options is shown. The methods are Monte Carlo (MC), classical product integration without (PR) and with transformation (PRTR), Quasi-Monte Carlo without (QM) and with transformation (QMTR) as well as classical sparse grid Gauß-Patterson quadrature without (SG) and with transformation (SGTR). We plot the computing time versus the quadrature error. Let us remark, that function evaluations would not be a fair work measure since the transformation requires additional zero finding. The superior convergence rate of the sparse grid method with transformation is clearly visible. Let us remark that both time and error are scaled logarithmically. Our new method is superior to Monte Carlo and Quasi-Monte Carlo since these approaches cannot use the higher smoothness of the integrand. The sparse grid method can exploit this smoothness optimally.



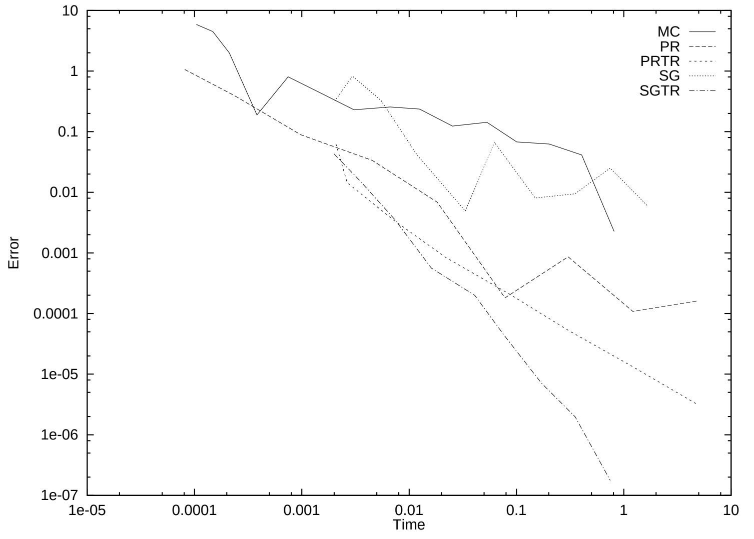


Figure 9.2: Computing time vs. integration error of the a geometric average Asian option (top) and a down-out barrier option (bottom) for various numerical integration methods.

9.3 CMO problem

Let us now consider the collateralized mortgage obligation problem. We assume that the pool of mortgages has a 21 1/3 year maturity and cash flows are obtained monthly. Then, the expected value requires the evaluation of the following integral for each tranche

$$\int_{\mathbb{R}^d} v(\xi_1, \ldots, \xi_d) \cdot g(\xi_1) \cdot \ldots \cdot g(\xi_d) \quad d\xi_1 \ldots d\xi_d, \tag{9.2}$$

with Gaussian weights $g(\xi_i) = (2\pi\sigma^2)^{-1/2} e^{-\xi_i^2/2\sigma^2}$. The dimension of the integral is $d = 256$ which simplifies the Brownian bridge construction.

The parameters are set to

$$(i_0, c, K_1, K_2, K_3, K_4, \sigma) := (0.007, 1.0, 0.01, -0.005, 10, 0.5, 0.0004). \tag{9.3}$$

The interest rates are either discretized using a random walk or the Brownian bridge construction. For the numerical computation, the integral over $\mathbb{R}^d$ is transformed to an unweighted integral on $[0,1]^d$ with the help of the inverse normal distribution. Note that this problem is much smoother than the previous path-dependent derivatives.

In Figures 9.3 and 9.4 we compare the conventional sparse grid method with the dimension-adaptive method for the random walk and the Brownian bridge discretization. The error is computed against an independent In Figure 9.3 we Quasi-Monte Carlo calculation. Note that also in this example the convergence rate of the conventional sparse grid approach is comparable to the Quasi-Monte Carlo method [46].

We see that again a weighting of the dimensions does not influence the convergence of the conventional sparse grid method. But for the dimension-adaptive method the amount of work is again substantially reduced (by several orders of magnitude) for the same accuracy when the Brownian bridge discretization is used and thus higher accuracies can be obtained. In this example the dimension-adaptive method also gives better results than the conventional sparse grid method for the random walk discretization. This implies that the conventional sparse grid spends too many points in mixed dimensions for this problem. The problem seems to be intrinsically lower-dimensional and nearly additive [15].

Classical sparse grid, random walk:

l	integral value	fcalls	indices
0	119.4059308399649950	1	1
1	119.2479112149794247	769	256
2	119.2204865071986433	296321	33152

Classical sparse grid, Brownian bridge:

l	integral value	fcalls	indices
0	119.4059308399649950	1	1
1	119.2484848592076929	769	256
2	119.2206858591296168	296321	33152

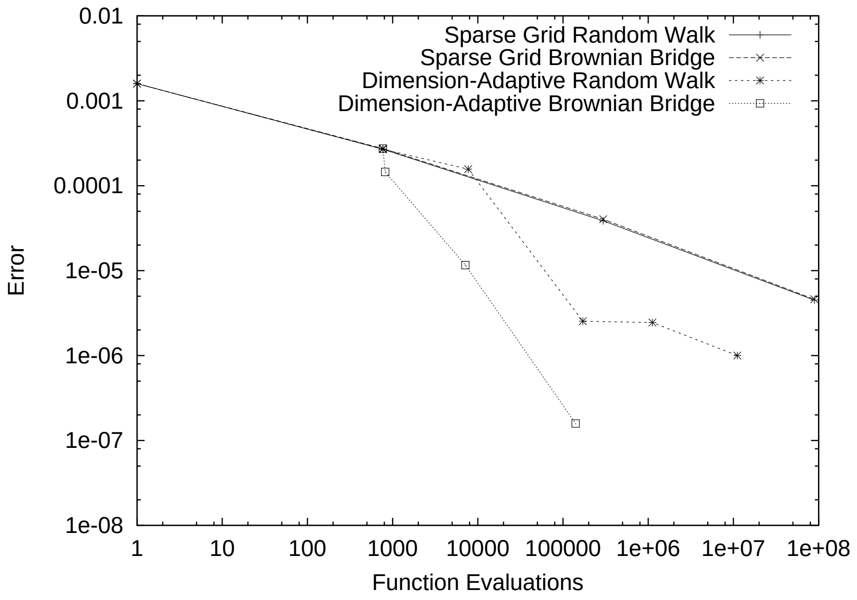
Dimension-adaptive sparse grid, random walk:

TOL	integral value	fcalls	indices
1e+1	119.4059308399649950	1	1
1e-0	119.2479112149794247	769	256
1e-1	119.2345402690575042	7711	1036
1e-2	119.2161776299815159	171009	19112
1e-3	119.2155830853408105	11217220	437623

Dimension-adaptive sparse grid, Brownian bridge:

TOL	integral value	fcalls	indices
1e+0	119.2484848592076929	769	256
1e-1	119.2331990732023428	817	262
1e-2	119.2172580210007311	7144	945
1e-3	119.2158938540621165	140552	15357
1e-4	119.2158644451883731	339244	32799
1e-5	119.2158825072985735	1224579	51844

Figure 9.3: A comparison of the conventional and dimension-adaptive sparse grids with random walk and Brownian bridge discretization of the CMO problem. We compare the level, respectively the error tolerance, the integral value, the number of function calls and the number of indices in the index sets.



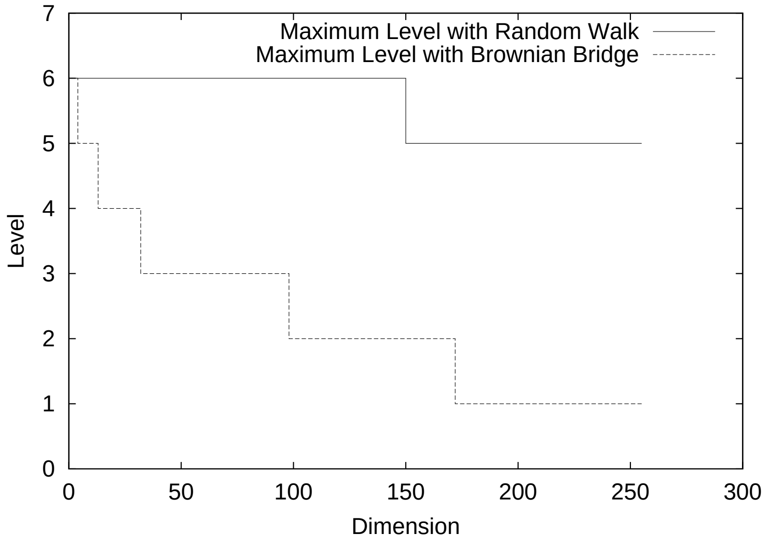


Figure 9.4: Computational results for the CMO problem ($d = 256$): integration error vs. number of function evaluations (left) and maximum level over all dimensions (sorted) for the dimension-adaptive algorithm with and without Brownian bridge discretization (right).

9.4 Performance-Dependent Options

Finally, we show pricing results for performance-dependent options in the full and reduced Black-Scholes model cases.

9.4.1 Full Model

We start with numerical examples to illustrate the use of the full model pricing formula of Theorem 5.4.3. In particular, we compare the efficiency of our algorithm to the standard pricing approach (denoted by STD) of quasi-Monte Carlo simulation of the expected payoff (4.3.4) based on Sobol point sets, see, e.g., Glasserman [52]. Monte Carlo instead of quasi-Monte Carlo simulation led to significantly less accurate results in all our experiments. We systematically compare the use of our pricing formula with

- Quasi-Monte Carlo integration based on Sobol point sets (QMC),
- Product integration based on the Clenshaw Curtis rule (P),
- Sparse Grid integration based on the Clenshaw Curtis rule (SG)

for the evaluation of the multivariate cumulative normal distributions (see Genz [39]).

We consider a Black-Scholes market with $n = 5$ assets. Thereby, we investigate the following three choices of bonus factors $a_{\mathbf{R}}$ in the payoff function (2.17):

In all cases, we use the model parameters $K = 100$, $S_i(0) = 100$, $i = 1, .., 5$, $T = 1$, $r = 5\%$ and as volatility matrix

$$\sigma = \begin{pmatrix} 0.1515 & 0.0581 & 0.0373 & 0.0389 & 0.0278 \\ 0.0581 & 0.2079 & 0.0376 & 0.0454 & 0.0393 \\ 0.0373 & 0.0376 & 0.1637 & 0.0597 & 0.0635 \\ 0.0389 & 0.0454 & 0.0597 & 0.1929 & 0.0540 \\ 0.0278 & 0.0393 & 0.0635 & 0.0540 & 0.2007 \end{pmatrix}. \tag{9.4}$$

and σ being a 5×5 volatility matrix whose entries are uniformly distributed in $[-1/d, 1/d]$. We assume that there are no dividend payments. This is motivated by a principal component analysis which we performed on the daily stock prices of all DAX companies in the year 2004. The analysis showed that already 5 principal components suffice to explain over 98% of the variance in the market.

In the performance-independent case of Example 2.4.4, an analytical solution is readily obtained by the Black-Scholes formula. In all other cases, we computed reference values for the option price on a very fine integration grid as a benchmark value to compare the efficiency of the different pricing approaches.

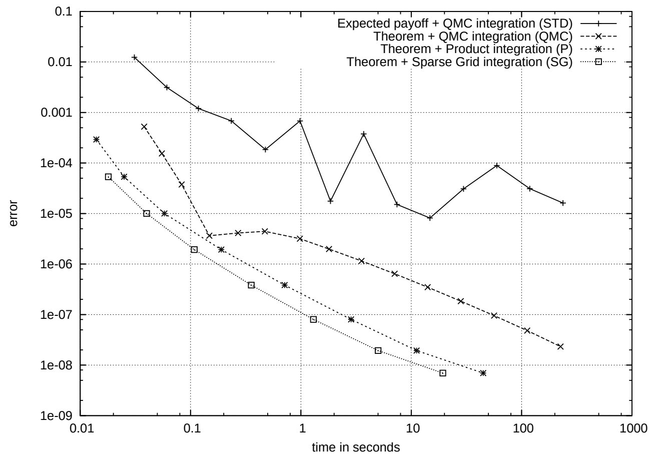
Example	$V(\mathbf{S}, 0)$	Discount	# Int
2.4.5	6.2354	34.02%	30
2.4.6	3.0183	68.06%	2
2.4.7	4.5612	51.73%	14

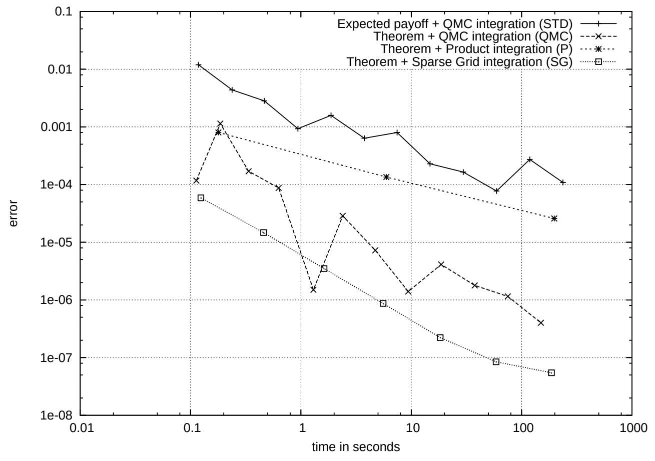
Table 9.1: Option prices, discounts compared to the corresponding plain vanilla option and number of computed normal distributions.

The computed option prices and discounts compared to the price of the corresponding plain vanilla option given by 9.4499 are displayed in the second and third column of Table 9.1. The number of normal distributions (# Int) which have to be computed is shown in the last column.

In principle, all bonus schemes described above can be hedged by the plain vanilla option in Example 2.4.4. The use of the appropriate performance-dependent option leads to much smaller costs for the company, however. The discounts of the performance-dependent option prices in comparison to the price of the corresponding plain vanilla option, which is also included in our framework by choosing $a_{\mathbf{R}} = 1$ if $R_1 = +$ and $a_{\mathbf{R}} = 0$ in all other cases, are displayed in the third column of Table 9.1.

The convergence behaviour of the four different approaches (STD, QMC, P, SG) to price the options from the Examples 2.4.5 – 2.4.7 are displayed in Figure 9.5. There, the time is displayed which is needed to obtain a given accuracy. One can see that the standard approach (STD) performs worst for all accuracies. The convergence rate is clearly lower than one in all Examples. The integration scheme suffers under the irregularity of the integrand which is highly discontinuous and not of bounded variation. The use of the pricing formula from Theorem 5.4.3 clearly outperforms the STD approach in terms of efficiency for all considered numerical integration methods. The QMC scheme exhibits a convergence rate of about one independent of the problem. The product integration approach (P) is competitive for low accuracies only. Its convergence rate suffers, however, under the curse of dimension and is smaller than 0.5. The combination of Sparse Grid integration with our pricing formula (SG) leads to the best overall accuracies and convergence rates in all cases. Even very high accuracy demands can be fulfilled in less than a few seconds.





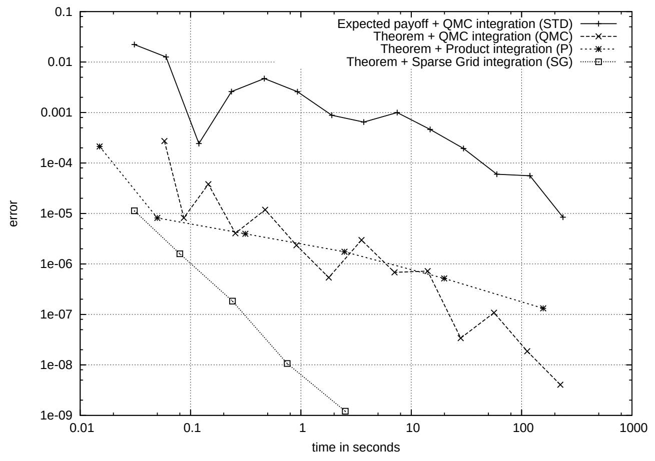


Figure 9.5: Errors and timings of the different numerical approaches to price the performance-dependent options of Examples 2.4.5 (top), 2.4.6 (middle) and 2.4.7 (bottom) in the full Black-Scholes model.

Example	$V(\mathbf{S}, 0)$	Discount	$\sharp$ Int	Dim	STD	QMC	P	SG
2.4.4	14.4995	-	1	1	1.1	-	-	-
2.4.5	12.9115	10.95%	41	2	0.58	0.88	1.45	1.55
2.4.6	1.8774	87.05%	31	5	0.6	1.1	0.27	1.87
2.4.7	8.6024	40.67%	38	3	0.52	1.3	0.89	1.54

Table 9.2: Option prices, discounts compared to the corresponding plain vanilla option, intrinsic dimensions and convergence rates of the different numerical approaches for the considered examples.

9.4.2 Reduced Model

Last but not least we present numerical examples to illustrate the performance of our approach to price performance-dependent options also in the reduced Black-Scholes model case using Theorem 5.4.7. In particular, we compare the efficiency of our algorithm to the direct pricing approach of (quasi-)Monte Carlo simulation of the expected payoff (4.3.4).

We consider a reduced Black-Scholes market with $n = 30$ assets and $d = 5$ processes. This setting corresponds, e.g., to the case of a performance-dependent option which includes the performance of all companies of the German stock index DAX in its payoff profile. We investigate the four different choices according to the Examples 2.4.4 – 2.4.7 from section 2.4.2 for the bonus factors $a_{\mathbf{R}}$ in the payoff function (2.17). Again, we use the model parameters $K = 100$, $S_1(0) = 100$, $T = 1$, $r = 5\%$. Now, σ is a 30×5 volatility matrix whose entries are uniformly distributed in $[-1/d, 1/d]$. We assume that there are no dividend payments.

In the performance-independent case of Example 2.4.4, an analytical solution is readily obtained by the Black-Scholes formula. In all other cases, we computed reference values for the option price on a very fine integration grid as a benchmark value to compare the efficiency of the different pricing approaches.

The prices of the performance-dependent options from the Examples 2.4.4–2.4.7 are displayed in the second column of Table 9.2. In principle, all bonus schemes described above could be hedged by the plain vanilla option in Example 2.4.4. The use of the appropriate performance-dependent option leads to much smaller costs for the company, however. The discounts of the performance-dependent option prices in comparison to the price of the corresponding plain vanilla option are displayed in the third column of Table 9.2. As explained in section 5.4.2, the complexity and dimensionality of our formula is often substantially reduced depending on the choice of the bonus factors. The number ($\sharp$ Int) and the maximum dimension (Dim) of normal distributions which have to be computed in the Examples 2.4.4–2.4.7 are displayed in the fourth and fifth column of Table 9.2. One can see that the number of required normal distributions is substantially lower than the theoretical bound (5.34) which is 174.437 for these examples. The maximum dimension varies from one to the nominal dimension five depending on the specific example.

In the last four columns of Table 9.2, the estimated asymptotic convergence rates are listed for four different schemes. In the standard approach denoted by STD, we used quasi-Monte Carlo integration to simulate the expected payoff (4.3.4). In the other three cases, the option prices were computed with the formula from Theorem 5.4.7. For the approximation of the normal distributions, we used the same integration schemes as in the previous section 8.3.2:

The convergence behaviour of the four different approaches STD, QMC, P and SG to price the performance-dependent options from the Examples 2.4.5 – 2.4.7 are displayed in Figure 9.6. There, the time is displayed which is needed to obtain a given accuracy. In the special case of Example 2.4.4, the application of Theorem 5.4.7 combined with the transformation (8.14) automatically reduces to the analytical solution given by the Black-Scholes formula with variance $\bar{\sigma}_1$. The exact solution up to machine precision is obtained in about 4.7 seconds by all integration schemes (QMC, P, SG). This is the time which is needed in the setup step of our algorithm to compute all vertices $\mathbf{v}$ and all weights $c_{\mathbf{v}}$. In the same time, the STD approach approximates the solution up to an error of $1e-03$. One can see that a simulation of the expected payoff (STD) performs similarly in all examples. Low accuracies are quickly achieved, the convergence rate is slow, though. The rate is about 0.6 in all examples and thus lower than one, as is maybe expected. The integration scheme suffers under the irregularity of the integrand which is highly discontinuous and not of bounded variation. The QMC scheme clearly outperforms the STD approach in all examples. It exhibits a convergence rate of about one and leads to much smaller errors after the setup time of 4.7 seconds. In contrast to the two previous approaches, the product integration approach (P) exhibits a high dependency on the specific example. While it performs very well in the Examples 2.4.4 and 2.4.5 it only converges with a rate of 0.27 in Example 2.4.6. Here, the curse of dimension, under which the product approach suffers, is clearly visible. While the intrinsic dimensions of Examples 2.4.4 and 2.4.5 are only one and two, respectively, the intrinsic dimension of Example 2.4.6 is five and, thus, equal to the nominal dimension. The combination of sparse grid integration with our pricing formula (SG) leads to the best convergence rates. The curse of dimension can be broken to some extent, while the favorable accuracy of the product approach is maintained. It is the most efficient scheme for the Examples 2.4.4, 2.4.5 and 2.4.7. However, for higher dimensional problems as Example 2.4.6, this advantage is only visible if very accurate solutions are required. In the pre-asymptotic regime, the QMC scheme leads to smaller errors.

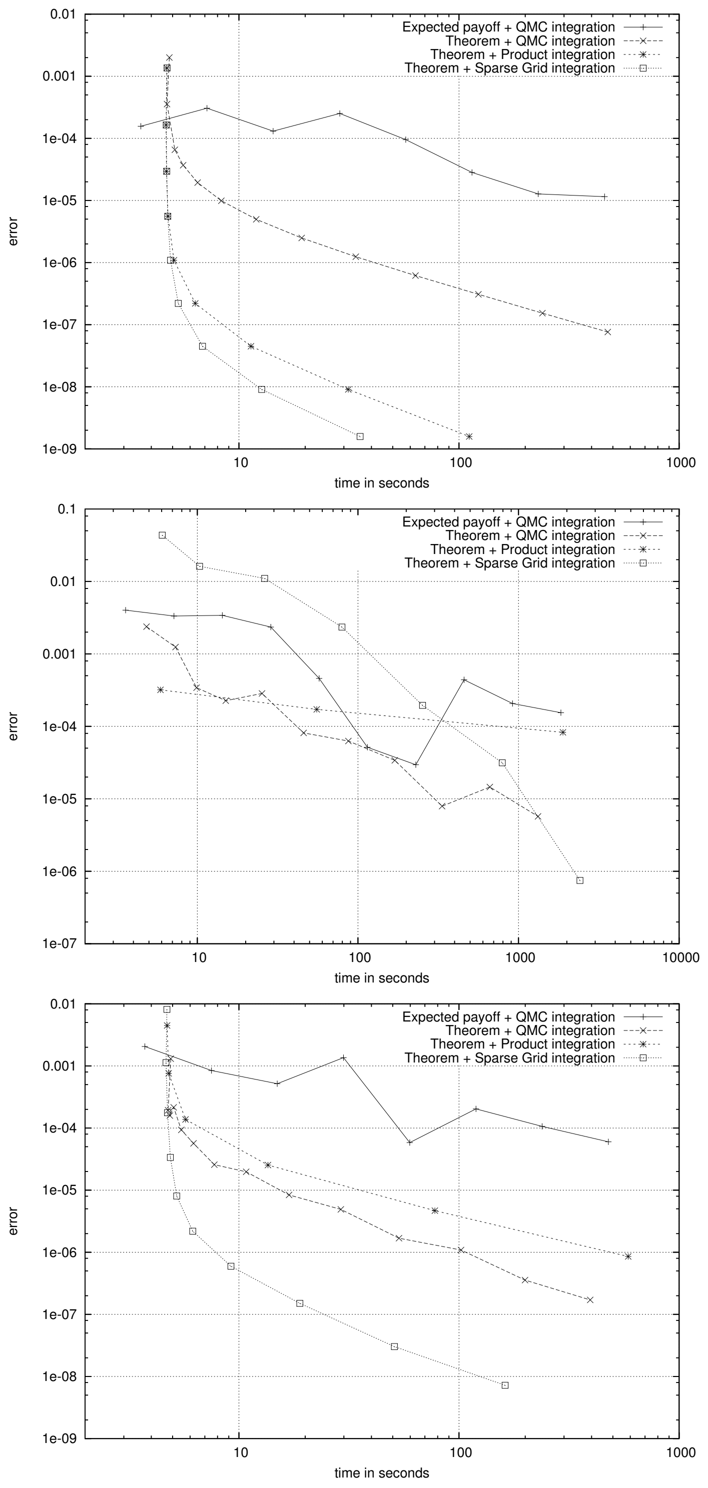


Figure 9.6: Errors and timings of the different numerical approaches to price the performance-dependent options of Examples 2.4.5 (top), 2.4.6 (middle) and 2.4.7 (bottom) in the reduced Black-Scholes model.

Chapter 10

Conclusions

Summary

In this thesis we applied sparse grid quadrature methods to computational finance problems. The sparse grid method is directly applicable to derivative security pricing problems which lead to smooth integrands which are often encountered during the pricing of interest rate derivatives or in some closed-form solutions. In the case of general option pricing problems, however, the corresponding integrands are usually not smooth and the convergence of the method deteriorates strongly. This way, the efficiency of sparse grid approach suffers significantly. As a second problem, the sparse grid method is, largely, but not completely independent of the dimension of the problem, which arises as the exponent of a logarithmic factor in the convergence rate.

We addressed these two problems of missing smoothness and dimension-dependence. To this end, we developed sparse grid methods which use zero-finding and transformation to treat discontinuities typically arising in the payoff of options. For the treatment of high-dimensional financial problems, we developed a dimension-adaptive numerical integration method. This method, which can be seen as a generalization of the sparse grid method, tries to find important dimensions automatically and places more integration points there. We have also discussed the implementation of the algorithm and proposed data structures which allow for the efficient bookkeeping of the sparse grid index sets.

We applied these techniques for the efficient valuation of path-dependent, interest rate and multi-asset derivatives. We have shown that this algorithm can substantially improve the convergence rate of the conventional sparse grid method through a reduction of the dependence on the dimension. This behaviour has been confirmed in numerical experiments and in application problems from computational finance. In these examples, the dimension-adaptive algorithm was clearly superior to the Monte Carlo and Quasi-Monte Carlo methods.

We also presented several approaches for the valuation of performance-dependent options in a Black-Scholes framework. The price of a such an option depends on the joint distribution of all stock prices in the benchmark. Thus, its valuation must be regarded as a high-dimensional integration problem.

As an alternative to a direct integration of the payoff we presented two analytical pricing formulas which involve the evaluation of several cumulative normal distribution functions. The pricing formula for the full model is useful in case of small benchmarks. It suffers, however, under a very high complexity and dimensionality if a larger number of benchmark companies are considered. Using novel tools from computational geometry we derived a more general formula for reduced models which incorporate less stochastic processes than companies and can be used for larger benchmarks as well.

In numerical examples we demonstrated for different typical bonus schemes that our pricing approach outperforms standard methods even for large benchmarks which may include as much as $n = 30$ companies and $d = 5$ stochastic processes. Thereby several deterministic integration methods were compared regarding their efficiency. Furthermore, for specific bonus schemes we showed that, independently of n and d, the pricing problem can be analytically reduced to a sum of two-dimensional normal distributions.

An additional advantage of our approach compared to standard Monte Carlo pricing is given by the fact that option price sensitivities can be obtained by analytical differentiation of the pricing formulas. The computation of the Greek letters can thus be integrated in the valuation algorithm without much additional effort.

In addition, we proposed two algorithms for the efficient management of hyperplane arrangements. The first algorithm performs the enumeration of all cells in a hyperplane arrangement. The second algorithms constructs an orthant decomposition of all cells. Both algorithms are based on a novel paradigm, a one-to-one correspondence of the cells in a hyperplane arrangement to a set of intersection points with an artificial bounding box. For fixed d, both algorithms run in optimal order complexity. They do not require complicated data structures and do not rely on other software such as a linear program solver. This way, they are easy to implement and to extend.

Possible Extensions

In the valuation formulas for performance-dependent options we restricted ourselves to payoff profiles which depend on relative performance comparisons to a specific benchmark. Payoff profiles which include absolute performance criteria, e.g., performance comparisons with different strike prices, can also be included. The corresponding valuation formulas then include weighted sums of gap option prices.

One important extension to the presented hyperplane arrangement algorithms is to use the bounding box as a selection tool and to consider only the part of the hyperplane arrangement inside the box. Since all vertices of the hyperplane arrangement which fall into this region (which can be much fewer than the original set of vertices) can be quickly determined, Algorithms 1 and 2 can be directly applied to this set only. This way, the cells inside the box can be enumerated and decomposed quickly. Both algorithms become output-sensitive and could then also be applied to very large hyperplane arrangements.

Let us finally remark that the whole dimension-adaptive sparse grid approach is not restricted to integration problems but can also be used for interpolation, the solution of partial differential and integral equations, or eigenvalue problems for the high-dimensional case. There, the possible application areas include computer simulations in statistical physics and chemistry, queueing theory, and data mining.

List of Figures

4.1 Overview and organization of the various methods for the valuation of financial derivatives. 38

5.1 Polyhedral cells and ranking vectors for two hyperplane arrangements with $d = 2, n = 2$ (left) and $d = 2, n = 3$ (right). 51

5.2 Illustration of the mapping between intersection points $\{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ and polyhedral cells $P_j := P_{\mathbf{v}_j}$ for the right arrangement from Figure 5.1 (left) and corresponding reflection signs $s_{\mathbf{v},\mathbf{w}}$ as well as the orthant $O_{\mathbf{v}_4}$ (right). . 52

6.1 Example hyperplane arrangement $\mathcal{A}_{3,2}$. Shown are the position vectors of the 7 cells and the 3 vertices. 58

6.2 Two mappings between intersection points $\{\mathbf{v}_1, \ldots, \mathbf{v}_7\}$ and cells $\{P_1, \ldots, P_7\}$ for the arrangement from Figure 5.1. 60

6.3 The orthant $O_{\mathbf{v}_4}$ and all reflection signs $s_{\mathbf{v},\mathbf{w}}$. 67

6.4 Decomposition of three cells using three orthants. 68

6.5 Time in seconds for the cell enumeration using Algorithm 6.3.1 (top) and Algorithm 6.3.7 (middle) as well as for the orthant decomposition using Algorithm 6.4.5 (bottom) of several hyperplane arrangements in dimensions $d = 4 - 7$. 71

7.1 The first two steps of a binomial tree. 74

7.2 Simulation tree with three branches and two time steps. 77

8.1 Left are the grid points of the trapezoidal sum for $l = 1, 2, 3$ in x- and y-direction as well as the corresponding product grids $\Delta_{k_1} \otimes \Delta_{k_2}$ for $1 \leq k_1, k_2 \leq 3$. Right is the corresponding sparse grid $Q_3^{(2)}$. 88

8.2 Examples for two-dimensional sparse grids based on the trapezoidal, Clenshaw-Curtis, Gauß-Patterson, and Gauß-Legendre rules ($l = 6$). 89

8.3 A few snapshots of the evolution of the dimension-adaptive algorithm. Shown are the sparse grid index sets (upper row) together with the corresponding sparse grids using the midpoint rule (lower row). Active indices are dark-shaded, old indices are light-shaded. The encircled active indices have the largest error indicators and are thus selected for insertion into the old index set. 95

8.4 The data types and memory requirements for the dimension-adaptive algorithm. 97

8.5 A schematic representation of the data structures. Shown are the arrays for the active and old indices `A` and `O`, the index elements `I`, the error estimates `G`, and the neighbours `N`. 99

8.6 Index **n** has just been generated as the forward neighbour in direction i of index **k**. The backward neighbour of **n** in direction $j \neq i$ can be found as the forward neighbour in direction i of the backward neighbour in direction j of **k**. 100

8.7 Integrands of the path- (left) and the performance-dependent option (right) for two-dimensional problems. 102

8.8 Sparse grids after zero finding and transformation for the path- (left) and the performance-dependent option (right). 103

9.1 Performance comparison of different quadrature methods for a smooth exponential function. 106

9.2 Computing time vs. integration error of the a geometric average Asian option (top) and a down-out barrier option (bottom) for various numerical integration methods. 107

9.3 A comparison of the conventional and dimension-adaptive sparse grids with random walk and Brownian bridge discretization of the CMO problem. We compare the level, respectively the error tolerance, the integral value, the number of function calls and the number of indices in the index sets. 109

9.4 Computational results for the CMO problem ($d = 256$): integration error vs. number of function evaluations (left) and maximum level over all dimensions (sorted) for the dimension-adaptive algorithm with and without Brownian bridge discretization (right). 110

9.5 Errors and timings of the different numerical approaches to price the performance-dependent options of Examples 2.4.5 (top), 2.4.6 (middle) and 2.4.7 (bottom) in the full Black-Scholes model. 113

9.6 Errors and timings of the different numerical approaches to price the performance-dependent options of Examples 2.4.5 (top), 2.4.6 (middle) and 2.4.7 (bottom) in the reduced Black-Scholes model. 116

List of Tables

6.1 Number of cells $c_{n,d}$ in a non-degenerate hyperplane arrangement for varying n and d. 59

6.2 Running times in seconds for several example hyperplane arrangements. Alg. 0, Alg. 1 and Alg. [116] enumerate the arrangement, Alg. 2 enumerates and decomposes it. 70

9.1 Option prices, discounts compared to the corresponding plain vanilla option and number of computed normal distributions. 112

9.2 Option prices, discounts compared to the corresponding plain vanilla option, intrinsic dimensions and convergence rates of the different numerical approaches for the considered examples. 114

Bibliography

[1] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8(3):295 – 313, 1992.

[2] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

[3] R. Bank. Hierarchical bases and the finite element method. *Acta Numerica*, 5:1–43, 1996.

[4] G. Baszenski and F.-J. Delvos. Multivariate Boolean midpoint rules. In *Numerical Integration IV*, pages 1–11. Birkhäuser, Basel, 1993.

[5] R. Bellman. *Dynamic Programming.* University Press, Princeton, 1957.

[6] J. Bernsten, T. Espelid, and A. Genz. Algorithm 698: DCUHRE – an adaptive multidimensional integration routine for a vector of integrals. *ACM Transactions on Mathematical Software*, 17:452–456, 1991.

[7] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[8] T. Bonk. A new algorithm for multi–dimensional adaptive numerical quadrature. In W. Hackbusch and G. Wittum, editors, *Adaptive Methods: Algorithms, Theory and Applications*, volume 46 of *Notes on Numerical Fluid Mechanics.* Vieweg, Braunschweig, 1993.

[9] H. Brass and K.-J. Förster. On the estimation of linear functionals. *Analysis*, 7:237–258, 1987.

[10] M. Broadie and P. Glasserman. Pricing american–style securities using simulation. *Journal of Economic Dynamics and Control*, 21:1323–1352, 1997.

[11] B. Büler, A. Enge, and K. Fukuda. Exact volume computation for convex polytopes: A practical study. In *Polytopes - Combinatorics and Computation*, pages 131–154. Birkhäuser, Basel, 2000.

[12] H.-J. Bungartz. *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson–Gleichung*. PhD thesis, Institut für Informatik, TU München, 1992.

[13] H.-J. Bungartz and M. Griebel. A note on the complexity of solving Poisson's equation for spaces of bounded mixed derivatives. *Jornal of Complexity*, 15:167–199, 1999.

[14] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:1–123, 2004.

[15] R. Caflisch, W. Morokoff, and A. Owen. Valuation of mortgage backed securities using Brownian bridges to reduce effective dimension. *Journal of Computational Finance*, 1, 1997.

[16] R. Carmona and V. Durrleman. Generalizing the Black-Scholes formula to multivariate contingent claims. *J. Computational Finance*, 9(2), 2006.

[17] C. Clenshaw and A. Curtis. A method for numerical integration on an automatic computer. *Numerische Mathematik*, 2:197–205, 1960.

[18] J. Cox, S. Ross, and M. Rubinstein. Option pricing: a simplified approach. *Journal of Financial Economics*, 7:229–263, 1979.

[19] P. Davis and P. Rabinowitz. *Methods of numerical integration*. Academic Press, 1975.

[20] F.-J. Delvos and W. Schempp. *Boolean methods in interpolation and approximation*, volume 230 of *Pitman Research Notes in Mathematics Series*. Longman, Essex, 1989.

[21] R. DeVore. Nonlinear approximation. *Acta Numerica*, 7:51–150, 1998.

[22] S. Dirnstorfer. Adaptive numerische Quadratur höherer Ordnung auf dünnen Gittern. Master's thesis, Institut für Informatik, TU München, 2000.

[23] W. Dörfler. A robust adaptive strategy for the nonlinear Poisson equation. *Computing*, 55:289–304, 1995.

[24] Z. Drezner and G. O. Wesolowsky. On the computation of the bivariate normal integral. *Journal of Statistical Computation and Simulation*, 35:101–107, 1990.

[25] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer, New York, 1987.

[26] H. Edelsbrunner. Algebraic decomposition of non-convex polyhedra. In *Proceedings of 36th Annual IEEE Symposium on Foundations of Computational Science*, pages 248–257. IEEE Computer Society, 1995.

[27] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal of Computation*, 15(2):341–363, 1986.

[28] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, 8(1):25–44, 1986.

[29] D. Egloff, W. Farkas, and M. Leippold. American options with stopping time constraints. *submitted to Mathematical Finance, available at SSRN: http://ssrn.com/abstract=798124*, 2005.

[30] K. Frank and S. Heinrich. Computing discrepancies of Smolyak quadrature rules. *Journal of Complexity*, 12, 1996.

[31] J. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19:1–141, 1991.

[32] J. Garcke. *Hochdimensionale Datenanalyse mit verallgemeinerten dünnen Gittern.* PhD thesis, Universität Bonn, 2004.

[33] J. Garcke. Berechnung der kleinsten Eigenwerte der stationären Schrödingergleichung mit der Kombinationstechnik. Master's thesis, Universität Bonn, 2006.

[34] J. Garcke and M. Griebel. On the parallelization of the sparse grid approach for data mining. In *Large-Scale Scientific Computations*, volume 2179 of *Lecture Notes in Computer Science*, pages 22–32, Heidelberg, 2001. Springer.

[35] J. Garcke and M. Griebel. Classification with anisotropic sparse grids using simplicial basis functions. *Intelligent Data Analysis*, 6(6):483–502, 2002.

[36] J. Garcke, M. Griebel, and M. Thess. Data mining using sparse grids. *Computing*, 67(3):225–253, 2000.

[37] J. Garcke, M. Hegland, and O. Nielsen. Parallelization of sparse grids for large scale data analysis. In *Proceedings of the International Conference on Computational Science*, volume 2659 of *Lecture Notes in Computer Science*, pages 683–692, Heidelberg, 2003. Springer.

[38] W. Gentleman. Implementing Clenshaw-Curtis quadrature. *Communications of the ACM*, 15:337–346, 1972.

[39] A. Genz. Numerical computation of multivariate normal probabilities. *J. Comput. Graph. Statist.*, 1:141–150, 1992.

[40] A. Genz. Comparison of methods for the computation of multivariate normal probabilities. *Comp. Science and Stat.*, 25:400–405, 1993.

[41] A. Genz. Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing*, 14:151–160, 2004.

[42] A. Genz and A. Malik. An adaptive algorithm for numerical integration over an n-dimensional rectangular region. *Journal of Computational and Applied Mathematics*, 6:295–302, 1980.

[43] V. Gerig. Parameterschätzung stochastischer Prozesse mit dünnen Gittern. Master's thesis, Universität Bonn, 2006.

[44] T. Gerstner. Adaptive hierarchical methods for landscape representation and analysis. In *Process Modelling and Landform Evolution*, Lecture Notes in Earth Sciences, pages 75–92. Springer, 1998.

[45] T. Gerstner, R. Goschnick, M. Griebel, M. Haep, and M. Holtz. Numerical simulation for the asset/liability management of life insurance contracts - part 1: Model. *Insurance: Mathematics & Economics*, 2007, in revision.

[46] T. Gerstner and M. Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18:209–232, 1998.

[47] T. Gerstner and M. Griebel. Dimension-adaptive tensor-product quadrature. *Computing*, 71(1):65–87, 2003.

[48] T. Gerstner and M. Holtz. Geometric tools for the valuation of performance-dependent options. In *Computational Finance and its Applications II*, pages 161–170. WIT Press, 2006.

[49] T. Gerstner and M. Holtz. Valuation of performance-dependent options. *Applied Mathematical Finance*, 2007, to appear.

[50] T. Gerstner, M. Holtz, and R. Korn. Valuation of performance-dependent options in a black-scholes framework. In *Proceedings Numerical Methods for Finance.* CRC Press, 2007, to appear.

[51] J. Geske and H. Johnson. The American put option valued analytically. *The Journal of Finance*, 39(5), 1984.

[52] P. Glasserman. *Monte Carlo Methods in Financial Engineering.* Springer, New York, 2003.

[53] G. Golub and J. Kautsky. Calculation of Gauss quadratures with multiple free and fixed knots. *Numerische Mathematik*, 41:147–163, 1983.

[54] J. Goodman and R. Pollack. Multidimensional sorting. *SIAM Journal of Computation*, 12:484–507, 1980.

[55] W. Gordon. Blending function methods of bivariate and multivariate interpolation and approximation. *SIAM Journal of Numerical Analysis*, 8:158–177, 1971.

[56] M. Griebel. A parallelizable and vectorizable multi-level algorithm on sparse grids. In *Parallel Algorithms for Partial Differential Equations*, volume 31 of *Notes on Numerical Fluid Mechanics*, Braunschweig, 1991. Vieweg.

[57] M. Griebel and S. Knapek. Optimized tensor-product approximation spaces. *Constructive Approximation*, 16(4):525–540, 2000.

[58] M. Griebel, P. Oswald, and T. Schiekofer. Sparse grids for boundary integral equations. *Numerische Mathematik*, 1999.

[59] M. Griebel and H. Wozniakowski. On the optimal convergence rate of universal and non-universal algorithms for multivariate integration and approximation. *Mathematics of Computation*, 75:1259–1286, 2006.

[60] M. Griebel and G. Zumbusch. Adaptive sparse grids for hyperbolic conservation laws. In *Proceedings of the 7th International Conference on Hyperbolic Problems*. Birkhäuser, Basel, 1998.

[61] K. Hallatschek. Fouriertransformation auf dünnen Gittern mit hierarchischen Basen. *Numerische Mathematik*, 63:83–97, 1997.

[62] D. Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry*, pages 389–412. CRC Press, 1997.

[63] J. Harrison and S. Pliska. Martingales and stochastic integrals in the theory of continuous trading. *Stochastic Processes and Applications*, 11:215–260, 1981.

[64] T. Hastie and R. Tibshirani. *Generalized additive models*. Chapman and Hall, London, 1990.

[65] T.-X. He. *Dimensionality reducing expansion of multivariate integration*. Birkhäuser, Basel, 2001.

[66] D. Heath and M. Schweizer. Martingales versus PDEs in finance: An equivalence result with examples. *Journal of Applied Probability*, 2000.

[67] M. Hegland. Adaptive sparse grids. In *Proceedings of CTAC, Brisbane, July 16–18, 2001*, 2001.

[68] M. Hegland and V. Pestov. Additive models in high dimensions. Technical Report 99–33, School of mathematical and computing sciences, Victoria University of Wellington, 1999.

[69] F. Hickernell. Quadrature error bounds with applications to lattice rules. *SIAM Journal of Numerical Analysis*, 33(5):1995–2016, 1996.

[70] J. Hull. *Options, Futures and other Derivative Securities*. Prentice Hall, Upper Saddle River, 2000.

[71] J. Hull and A. White. Accounting for employee stock options: A practical approach to handling the valuation issues. *Journal of Derivatives Accounting*, 1(1):3–9, 2004.

[72] J. Hull and A. White. How to value employee stock options. *Financial Analysts Journal*, 60(1):114–119, 2004.

[73] M. Kalos and P. Whitlock. *Monte Carlo Methods*. John Wiley & Sons, 1986.

[74] I. Karatzas. *Lectures on the Mathematics of Finance*, volume 8 of *CRM Monograph Series.* American Mathematical Society, Providence, R.I., 1997.

[75] S. Khavinson. *Best approximation by linear superposition (approximate nomography).* AMS Translations of Mathematical Monographs vol. 159. AMS, Providence, 1997.

[76] P. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations.* 1. Springer, Heidelberg, 1992.

[77] S. Knapek. *Approximation und Kompression mit Tensorprodukt-Multiskalenräumen.* PhD thesis, Universität Bonn, 2000.

[78] A. Kolmogoroff. On the representation of continuous functions of several variables by superpositions of continuous functions of fewer variables. *Dokl. Akad. Nauk SSSR*, 108:179–182, 1956. (in Russian, Engl. Transl.: Amer. Math. Soc. Transl. (2) 17:369–373, 1961).

[79] A. Kolmogoroff. On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR*, 114:953–956, 1957. (in Russian, Engl. Transl.: Amer. Math. Soc. Transl. (2) 28:55–59, 1963).

[80] R. Korn. Valuation of tailored options in a Black–Scholes framework. Working paper, Universität Kaiserslautern, 1996.

[81] R. Korn and E. Korn. *Option pricing and portfolio optimization.* volume 31 of *Graduate Studies in Mathematics.* American Mathematical Society: Providence, R.I., 2001.

[82] A. Kronrod. *Nodes and weights of quadrature formulas.* Consultants Bureau, New York, 1965.

[83] Y.-K. Kwok. *Mathematical Models of Financial Derivatives.* Springer, Heidelberg, 1998.

[84] L. Laloux, P. Cizeau, J. Bouchaud, and M. Potters. Noise dressing of financial correlation matrices. *Physical Review Letters*, 83(7):1467–1470, 1999.

[85] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–271, 1991.

[86] N. Meade and G. Salkin. Index funds-construction and performance measurement. *Journal of Operational Research Society*, 40(10):871–879, 1989.

[87] N. Meade and G. Salkin. Developing and maintaining an equity index fund. *Journal of Operational Research Society*, 41(7):99–607, 1990.

[88] T. Mertens. Optionspreisbewertung mit dünnen Gittern. Master's thesis, Universität Bonn, 2005.

[89] G. Monegato. Stieltjes polynomials and related quadrature rules. *SIAM Review*, 24(2):137–158, 1982.

[90] B. Moro. The full Monte. *RISK*, 8(2), 1995.

[91] M. Musiela and M. Rutkowski. *Martingale Methods in Financial Modelling.* Springer, Heidelberg, 1997.

[92] T. Nahm. Error estimation and index refinement for dimension-adaptive sparse grid quadrature with applications to the computation of path integrals. Master's thesis, Universität Bonn, 2005.

[93] H. Niederreiter. *Random number generation and quasi–Monte Carlo methods.* SIAM, Philadelphia, 1992.

[94] E. Novak and K. Ritter. Global optimization using hyperbolic cross points. In *State of the Art in Global Optimization*, pages 19–33, Dordrecht, 1996. Kluwer.

[95] E. Novak and K. Ritter. High dimensional integration of smooth functions over cubes. *Numerische Mathematik*, 75:79–98, 1996.

[96] D. Oeltz. *Ein Raum-Zeit Dünngitterverfahren zur Diskretisierung parabolischer Differentialgleichungen.* PhD thesis, Universität Bonn, 2004.

[97] P. Orlik and H. Terao. *Arrangements of Hyperplanes.* Springer, New York, 1992.

[98] J. O'Rourke. Computational geometry column. *International Journal of Computational Geometry and Applications*, 6(2):243–244, 1996.

[99] S. Paskov. Average case complexity of multivariate integration for smooth functions. *Journal of Complexity*, 9:291–312, 1995.

[100] S. Paskov and J. Traub. Faster valuation of financial derivatives. *Journal of Portfolio Management*, 22:113–120, 1995.

[101] T. Patterson. The optimum addition of points to quadrature formulae. *Mathematics of Computation*, 22:847–856, 1968.

[102] T. Patterson. Modified optimal quadrature extensions. *Numerische Mathematik*, 64:511–520, 1993.

[103] K. Petras. Asymptotically minimal Smolyak cubature. *Numerische Mathematik*, 2002.

[104] R. Piessens and M. Branders. A note on the optimum addition of abscissas to quadrature formulas of Gauss and Lobatto type. *Mathematics of Computation*, 28:135–140, 1974.

[105] L. Plaskota. The exponent of discrepancy of sparse grids is at least 2.1933. *Advances in Computational Mathematics*, 12:3–24, 2000.

[106] T. Rassias and J. Simsa. *Finite sums decompositions in mathematical analysis.* John Wiley & Sons, Chichester, 1995.

[107] M. Reiferscheid. Numerische Simulation von Sprung-Diffusions-Prozessen zur Optionspreisbewertung. Master's thesis, Universität Bonn, 2006.

[108] C. Reisinger. *Numerische Methoden für hochdimensionale parabolische Gleichungen am Beispiel von Optionspreisaufgaben.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2004.

[109] D. Röschke. Über eine Kombinationstechnik zur Lösung partieller Differentialgleichungen. Master's thesis, Institut für Informatik, TU München, 1991.

[110] M. Schervish. Multivariate normal probabilities with error bound. *Applied Statistics*, 33:81–87, 1984.

[111] C. Schwab and R. Todor. Sparse finite elements for stochastic elliptic problems – higher order moments. *Computing*, 71(1):43–63, 2003.

[112] R. Sedgewick. *Algorithms in C.* Addison Wesley, 1990.

[113] R. Seydel. *Tools for Computational Finance.* Springer, Heidelberg, 2002.

[114] J. Simsa. The best L^2-approximation by finite sums of functions with separable variables. *Aequationes Mathematicae*, 43:284–263, 1992.

[115] N. Sleumer. Output-sensitive cell enumeration in hyperplane arrangements. *Nordic Journal of Computing*, 6(2):137–147, 1999.

[116] N. Sleumer. *Hyperplane Arrangements: Construction, Visualization and Applications.* PhD thesis, ETH Zürich, 2000.

[117] I. Sloan and S. Joe. *Lattice methods for multiple integration.* Oxford University Press, Oxford, 1994.

[118] S. Smolyak. Interpolation and quadrature formulas for the classes W_s^a and E_s^a. *Dokl. Akad. Nauk SSSR*, 131:1028–1031, 1960. (in Russian, Engl. Transl.: Soviet Math. Dokl. 4:240–243, 1963).

[119] F. Sprengel. *Interpolation und Waveletzerlegung multivariater periodischer Funktionen.* PhD thesis, Universität Rostock, 1997.

[120] A. Steinbauer. *Quadraturformeln für das Wienermaß.* PhD thesis, Mathematisches Institut, FAU Erlangen–Nürnberg, 2000.

[121] A. Stroud. *Approximate Calculation of Multiple Integrals.* Prentice Hall, 1971.

[122] V. Temlyakov. *Approximation of periodic functions.* Nova Science Publishers, New York, 1994.

[123] J. Traub, G. Wasilkowski, and H. Woźniakowski. *Information–based complexity.* Academic Press, New York, 1988.

[124] P. Van Dooren and L. De Ridder. An adaptive algorithm for numerical integration over an n–dimensional cube. *Journal of Computational and Applied Mathematics*, 2:207–217, 1976.

[125] R. VerfühRt. *A review of a posteriori error estimation and adaptive mesh–refinement techniques.* Teubner, 1996.

[126] G. Wahba. *Spline models for observational data.* SIAM, Philadelphia, 1990.

[127] S. Wahl. Numerical valuation of financial derivatives by sparse grid integration methods. Diploma theses, Universität Bonn, 2001.

[128] C. Warawko. Numerische Verfahren zur Bewertung Bermudscher Optionen. Master's thesis, Universität Bonn, 2006.

[129] G. Wasilkowski and H. Woźniakowski. Explicit cost bounds of algorithms for multivariate tensor product problems. *Journal of Complexity*, 11:1–56, 1995.

[130] G. Wasilkowski and H. Woźniakowski. Weighted tensor product algorithms for linear multivariate problems. *Journal of Complexity*, 15:402–447, 1999.

[131] P. Wilmott, S. Howison, and J. Dewynne. *The Mathematics of Financial Derivatives.* Cambridge University Press, 1995.

[132] R. Yue and F. Hickernell. Robust designs for smoothing spline ANOVA models. *Metrika*, 55:161–176, 2002.

[133] C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations*, volume 31 of *Notes on Numerical Fluid Mechanics.* Vieweg, Braunschweig, 1991.

[134] P. Zhang. *Exotic Options.* World Scientific, Singapore, 2nd edition, 1998.