Algorithms for Nonlinear Models in Computational Finance and their Object-oriented Implementation

by

Robert Buff

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Computer Science
Graduate School of Arts and Sciences
New York University
September 1999

Approved: ______________________
Marco Avellaneda

Acknowledgment

I have traded options only once in my life, almost a decade ago. Having learned about options from newspaper diagrams that proliferated after the Deutsche Terminbörse was set up, my initial investment of 5000 DEM shriveled to 1500 DEM in a matter of days.

A few years ago, I returned to the subject as a computer science student at NYU. To create risk management tools appealed as a safer road to the holy grail of high finance, and one for which I was well equipped, given my experience and education. Having finished this thesis, I am encouraged to promise that eventually, I'm going to use the insight I gained over the last few years to make those 3500 DEM (and some more) back!

I would like to thank my advisor, Prof. Marco Avellaneda, for accepting me as the financial greenhorn I was and guiding me to a stage where I can navigate Wall Street with confidence. Besides teaching me the intricacies of UVM and dynamic programming, Marco has always allowed me to contribute from the perspective of a computer scientist.

I thank Prof. Robert Kohn and Prof. Bud Mishra, Prof. Jonathan Goodman and Prof. Michael Overton for serving as members of my committee.

I thank Prof. Arthur Goldberg for funding me over the last two years. Readers of the last chapter of this thesis will notice that our work on Internet performance issues has inspired the direction of my research quite a bit.

Contents

List of Figures

1 Introduction

This thesis tries to pioneer some unexplored aspects in computational finance. Computational finance is rapidly gaining reputation as a field worthy of dedicated research. Although far from maturity, it may one day complement the established league of financial economics, corporate, mathematical and statistical finance as equal partner.

There are some institutional activities that try to advance computational finance as a field in its own right. Publications such as the "Journal of Computational Finance" or the "Journal of Computational Intelligence in Finance" encourage research that uses computational techniques. Topics range from numerical methods to neural networks to genetic algorithms. Some academic places offer graduate programs in computational finance, based on a mixture of finance, mathematics and computer science courses.

Then, of course, there is the vast collection of books on standard derivative pricing models that, to a more or lesser degree, contain recipes on how these models are implemented on a computer. In virtually all cases, instructions are kept on a very high level, or actual programs are rudimentary and isolated solutions.

In this thesis, we attempt to combine results in mathematical finance with object-oriented software-development techniques. The goal is to create a program that solves the particular financial problem that we have posed ourselves, is extensible, durable, and capable of forming the base of an industrial-strength product.

These features make it necessary to create a shell of supporting software first. The mathematically sophisticated code that tackles the particular pricing problem must be inserted into this shell. The shell, in fact, turns out to be rather large—81500 lines of C++ code and 11500 of Java code have been written altogether for this thesis, of which, for instance, only 2800 deal with the combinatorial structure of the pricing problem (these 2800 lines do not contain numerical code).

Large programs are best developed through modularization. The unit of modularization under the object-oriented approach is the class—an abstract data type that may inherit properties from super or parent classes and defer the instantiation of properties to child or sub classes. The final application ideally consists of a forest of shallow class hierarchies. Our code contains classes and class hierarchies for entities that have direct financial significance (instruments, portfolios, models, scenarios), classes that support certain mathematical methods (lattices, finite difference solvers, path spaces, optimizers), classes that control the evaluation loop (compute engines and evaluaters), scripting classes (parser, scanner, script sources, expressions), system classes (sockets, pipes, services), and many others. Altogether, there are about 135 classes in our code.

In the following pages we report on our concrete achievements, and hope to induce the reader to participate in our vision. Our achievements are two-fold:

1. We have solved the worst-case pricing problem for path-dependent options such as barrier or American options under uncertain volatility assumptions. We have also added a new type of uncertain volatility scenario: volatility shock scenarios allow a fixed number of limited-duration volatility oscillations of high amplitude.

2. We have done so by creating a software environment that is thoroughly modular, object-oriented, and extensible. Combinatorial and numerical algorithms are separated and orthogonal. The system is downwards-compatible without performance loss (i.e., it solves the plain Black-Scholes PDE without performance penalty). It is upwards-compatible in the sense that extensions to the system are indeed *extensions*—they don't require an overhaul of the existing code (this has been proven experimentally when we added Monte Carlo optimization methods).

Our vision has two aspects as well:

1. The worst-case pricing problem for path-dependent options is only one case among many that require combinatorial and numerical methods for their solution. In this singular example, the combinatorial aspect dominates the running time and therefore justifies the search for efficient algorithmic techniques. In general, we think the convergence of numerical methods and discrete algorithms promises interesting research directions and practical applications that benefit the financial community.

2. The evaluation of portfolios does not occur in a vacuum. Data needs to flow in from somewhere, and prices, curves, or calibrated surfaces need to be propagated. The problems we are investigating require considerable computing resources. For that reason, a client-server approach is very attractive: the client specifies the concrete pricing problem and supplies some of the data, and the high-powered server computes the answer, possibly augmenting the data with pre-fabricated data that resides on the server (such as a calibrated volatility surface, for instance).

We have started to explore this architecture through Java- and HTML-based client frontends and server backends that receive requests via TCP or CGI. Ultimately, we envision a centralized site that offers a variety of pricing, hedging and calibration services that are based on novel techniques such as those presented in this thesis.

For the reader in a hurry, Fig. 1.1 summarizes the microscopic and macroscopic aspects of our achievements and vision in a nutshell.

The overview on the next few pages describes and motivates our interest in the algorithmic and architectural topics in this thesis.

1.1 Uncertain Volatility Scenarios and Exotic Options

It is widely accepted that the assumption of constant volatility in financial models (such as the original Black-Scholes model) and derivatives prices observed in the market are incompatible. There are several ways to fix this deficiency: prescribed heterogeneous yet deterministic volatility models, stochastic volatility models, or the calibration of a volatility surface to market prices are common approaches.

Strongly related to finding the right method of modeling volatility is the problem to measure the exposure of the options portfolio under investigation to volatility risk; how does the model value of the portfolio change if the volatility is perturbed a little?

Uncertain volatility models attack both problems: they select a concrete volatility surface among a candidate set of volatility surfaces, and they answer the sensitivity question by computing an upper bound that the value of the portfolio can take under any

	achievement	vision
microscopic	uncertain volatility models for barrier and American options	discrete algorithms dominate numerical methods
macroscopic	scalable, object-oriented software solution	Web-computing for finance takes off

Figure 1.1: Our achievements and vision in a nutshell

candidate volatility. (By inverting the position, a lower bound can be computed as well.) This is achieved by choosing the local volatility $\sigma(S_t, t)$ among two extremal values $\sigma_{\min}$ and $\sigma_{\max}$ such that the value of the portfolio is maximized locally.

Uncertain volatility *scenarios* generalize this approach: given a model that exhibits uncertainty in some of its coefficients (the volatility, in particular), instantiate those uncertain coefficients such that some objective is fulfilled. This objective is called a *scenario.*

The original uncertain volatility model by Avellaneda and Parás (1995) is a worst-case scenario for the sell-side. By maximizing the portfolio value and charging accordingly, sellers are guaranteed coverage against adverse market behavior if the realized volatility belongs to the candidate set. Worst-case prices are nonlinear, due to diversification of volatility risk and "gamma-risk." Worst-case evaluation is based on a nonlinear Hamilton-Jacobi-Bellman equation that generalizes Black-Scholes by adjusting the local volatility, or conditional variance, to the local gamma.

The worst-case volatility scenario (our notion) has been implemented for portfolios of vanilla options, for which the Hamilton-Jacobi-Bellman equation is straightforward to implement on a computer. An extension that hedges a portfolio of vanilla options with liquidly traded market benchmarks is presented in Avellaneda and Parás (1996).

The computational overhead, however, grows quite dramatically once path-dependent options, such as barrier or American options, are added to the portfolio. The worst-case volatility scenario from today's perspective of a portfolio containing an American option, for instance, depends on whether the option is exercised today or not (for simplicity, assume the option can be exercised only at finitely many times). A worst-case pricer must compare

- the worst-case price of the portfolio under the assumption that the American option is exercised tomorrow at the earliest;
- the worst-case price of the portfolio minus the American option, plus the cashflow received or paid immediately from early exercise.

The pricer then must select the early exercise strategy that fits the worst-case assumption. As the number of American options in the portfolio increases, the number of different early exercise strategies that must be invesigated increases potentially exponentially, as nonlinearity forces the pricer to consider all relevant combinations. This leads to a hierarchy of interdependent PDE's, each solving a Hamilton-Jacobi-Bellman problem.

In this thesis, we solve the pricing problem for portfolios containing barrier and American options, under worst-case volatility scenarios. For barrier options, the computational complexity can be determined beforehand and is always $O(n^2)$, n being the number of barrier options in the portfolio. For American options, the situation becomes more difficult since

- the early exercise boundaries are not known a priori: each PDE describes a free boundary problem, the boundary value being selected locally from a hierarchy of subordinate PDE's (numerical aspect);
- the pricer must distinguish between long and short positions, as agents can use their long positions to counter somewhat the worst-case early exercise strategies ascribed to the investors with whom they have established their short positions. This gives rise to the notion of best worst-case scenario (combinatorial aspect).

Potentially, up to $O(2^n)$ early exercise combinations need to be considered (n being the number of American options in the portfolio). This, of course, is unacceptably expensive. We have developed algorithms that reduce the number of combinations tested locally, but remain correct in the sense that, locally, the best worst-case scenario is always found. We also present a heuristic which reduces the compute time further, but is no longer guaranteed to be correct.

1.2 Volatility Shock Scenarios

Worst-case volatility scenarios limit the candidate set to volatilities that oscillate between two extremal bounds (which may be heterogeneous). The resulting spread between the worst-case values for the original and inverted position is often unacceptably large. To narrow the extremal bounds $\sigma_{\min}$ and $\sigma_{\max}$ is a possible solution, but also makes it less likely that the volatility realized later indeed observes those bounds. To narrow the extremal bounds selectively in some places, and leave them unmodified (or even widened) in others seems a plausible alternative, allowing for periods of relative calm and periods of *volatility shocks* with high amplitude.

Where on the time axis should those periods of high volatility fluctuation be located? If market events that influence volatility cannot be foreseen, the exact location of volatility shocks is difficult to determine. The worst-case paradigm comes to rescue: it is the pricer's task to locate volatility shock periods where they cause the most damage, in a path-dependent way. Thus, the portfolio is not only maximized over the local volatility, but also over the location of volatility shock periods.

An example helps to clarify. Suppose the volatility is estimated at 15%. There exists very likely, we assume, one short period of 3 days during which the volatility may vary between 15 and 100%. Given some portfolio, what is its worst-case value under the assumption that the 3-day volatility shock period can start anytime? Its start date may even be path-dependent: it may start earlier if the stock price moves up, and later if it moves down.

Volatility shock scenarios can be solved with dynamic programming. We have developed algorithms that solve volatility shock scenarios for portfolios of vanilla, barrier and American options. The number f of volatility shock periods is not limited; the overhead is linear in f (for instance, if there is exactly one volatility shock period of duration one day, then the slowdown factor compared to the regular worst-case scenario is 3).

Volatility shock scenarios are a useful new member in the arsenal of tools that assess volatility risk. They furthermore fit neatly into the scenario paradigm introduced above.

1.3 Object-Oriented Implementation

The thesis title promises insight into the actual implementation. We comply by, first of all, giving a name to our creation: Mtg. Mtg consists of modules MtgLib, MtgClt, MtgSvr, MtgCal and MtgMath, where the latter three are essentially only wrappers around the C++ class library MtgLib, offering different ways to access its features. MtgClt is a Java frontend to MtgSvr.

MtgLib contains object-oriented code that solves the worst-case volatility and volatility shock scenarios for vanilla, barrier and American options. Its higher-level combinatorial classes are geared towards multi-factor models on lattices. Its numerical classes for finite difference solutions (explicit and mixed implicit/explicit) accept any one-factor model. MtgLib strictly adheres to the scenario concept.

Figure 1.2 shows how successive refinement leads from a general view on evaluation

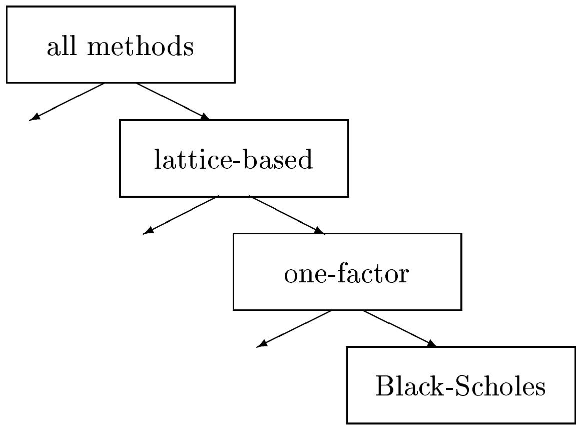


Figure 1.2: Progressing from a general view on evaluation to the concrete method for the concrete model. The boxes correspond to classes `tEngine`, `tFDEngine`, `tOFEngine` and `tGeoEngine` in MtgLib

to a concrete lattice-based method supporting a one-factor Black-Scholes model. At each level in the hierarchy, alternative approaches can be spawned off.

A similar hierarchy can be drawn for scenarios: scenarios in general are refined to worst-case volatility scenarios, which in turn are extended to volatility shock-scenarios. The choice of the scenario is orthogonal to the choice of the method of evaluation. At the deepest level, Black-Scholes may be evaluated under either scenario.

In this thesis, we give a broad overview over the categories of classes in MtgLib. Interfaces are emphasized over implementation details. We hope our exposition proves that MtgLib is an example of good object-oriented design.

1.4 Client-Server Computing on the Web

Two of our programs are accessible on the World Wide Web:

- MtgSvr is a general-purpose server that accepts requests in a customized scripting language and returns the result in ASCII format. MtgClt is a Java frontend to MtgSvr that can be downloaded from our website. It connects to MtgSvr through the TCP protocol. Through MtgSvr, MtgClt handles all cases of exotic options (barrier, American) and volatility scenarios (worst-case, volatility shock) discussed in this thesis.

- MtgCal is a calibrator for fixed-income markets. The user specifies model coefficients and benchmark instruments in an HTML form. After the data has been submitted through the CGI protocol to MtgCal, calibration is started on the server and eventually produces a result HTML page, which can then be inspected by the user.

MtgSvr uses lattice-based numerical methods. MtgSvr demonstrates that the algorithms proposed in this thesis can be implemented. MtgCal uses Monte Carlo simulation and minimum-entropy optimization to calibrate. We mention MtgCal and discuss some of its features to give an idea of the direction in which our work is heading.

Client-server computing based on standard web technology creates a variety of problems:

- The sandbox and firewall problem: MtgClt may be unable to connect to the server via low-level TCP if the security settings in the web-browser are high, or there is a firewall between the server and the client (this is the case for most corporate clients that access our server at NYU). Possible solutions to this problem exist (HTTP tunneling, the SOCKS protocol), but haven't been explored by us.

- The HTTP protocol is designed for simple request-response transactions. Long-lived transactions that lead to considerable CPU overhead at the server (like calibration which might take several minutes to complete) do not fit this paradigm very well. MtgCal uses a polling mechanism that lets the client detect the final result (almost) as soon as it becomes available, and yet does not go beyond basic HTTP/Javascript technology.

Other issues are related to security (HTTPS versus HTTP), dissemination of results (the calibrated surface for subsequent pricing), and the fee structure for such online services. In summary, we consider *web computing for finance* a challenging field which will definitely encourage future research here at NYU.

1.5 Related Work

Starting point of this thesis is the uncertain volatility model by Avellaneda and Parás (1995). Its extension to barrier and American options is, to the best of our knowledge, original. The volatility shock scenario is a refinement of the band-approach and also, to the best of our knowledge, original.

Part I gives an overview over the literature in mathematical and computational finance, as far as it is relevant to our work. Chapter 4 in particular reviews uncertain volatility models and the notion of scenario-based pricing to which they give rise.

1.6 How to Best Read this Thesis

We summarize the following chapters and try to assess their respective value for readers of different backgrounds.

Chapter 2 summarizes notation and conventions, most of which are standard or intuitive. This chapter can be consulted as need arises.

Chapter 3 gives a short overview over mathematical finance. The Black-Scholes model receives the most attention, although interest-rate models such as HJM are also mentioned. The distinction between deterministic and stochastic volatility is emphasized. This chapter can be skipped safely by anyone familiar with the terms.

Chapter 4 reviews the concept of uncertain model coefficients and introduces the notion of "scenarios." Pricing, hedging and calibration are briefly discussed as three applications. An understanding of these issues is essential.

Chapter 5 introduces the multi-lattice framework within which our algorithms are developed. This chapter defines notation and key data structures that should not be missed. It also presents insight into some numerical issues regarding stability in Sect. 5.2. This section is rather technical and can be skipped at first reading (not by anyone actually implementing our algorithms, though).

Chapter 6 discusses algorithms for scenario-base evaluation of barrier option portfolios. In particular, it shows how to set up multi-lattice dynamic programming so that the potentially large number of PDE's can be handled. This chapter and the next describe the key algorithmic achievements of this thesis. Please read them.

Chapter 7 discusses algorithms for scenario-based evaluation of American option portfolios. Evaluating American options is more complex than evaluating barrier options, but the same idea of ordering solutions of PDE's hierarchically applies. The economic implications are discussed in Sect. 7.1.

Chapter 8 describes an extension to worst-case volatility scenarios. The volatility is now allowed to exhibit short shocks at unpredictable times. This chapter is independent of Chapters 6 and 7 and can be read immediately after Chapter 5. The style is less formal.

Chapter 9 gives an overview over the class library MtgLib and has the class declarations for the core classes. This chapter gives an idea of the architecture of a system that has the capabilities described in Chapters 6, 7 and 8. Readers can benefit from the exposition even if they have not read those chapters. Chapter 9 focuses on the architecture of MtgLib and is not a tutorial on its use.

Chapter 10 describes some aspects of MtgSvr and MtgCal, our two online applications. MtgSvr is discussed to demonstrate the feasibility of the algorithms in Chapters 6, 7 and 8. MtgCal is included to show ongoing work and motivate possible future directions. This chapter is fairly self-contained.

More information on the work presented in this thesis can be obtained by following these links:

`buff@cs.nyu.edu`

`http://www.courantfinance.cims.nyu.edu`

Part I

Computational Finance: Theory

2 Notation and Basic Definitions

2.1 Linear Algebra

$\mathbb{N}$ denotes the nonnegative integers. $\mathbb{R}$ denotes the real numbers. $\mathbb{R}_+$ denotes the nonnegative real numbers. $\mathbb{R}_{++}$ denotes the strictly positive real numbers.

Vectors and matrices are typeset in boldface (except when greek symbols are used): $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n\times m}$. Vectors are interpreted as column vectors: $\mathbb{R}^n = \mathbb{R}^{n\times 1}$. In text, they are quoted in transposed form.

The normal font is used for vector or matrix components: $\mathbf{a} = (a_1, \dots, a_n)^{\mathrm{T}} = (a_i \mid 0 \le i \le n)^{\mathrm{T}}$.

The zero vector is denoted by $\mathbf{0}$. The dot product is written $\mathbf{a}\cdot\mathbf{b} = \mathbf{a}^{\mathrm{T}}\mathbf{b}$. If $\mathbf{a} \in \mathbb{R}^n$ is a vector, $\mathbf{B} = \mathbf{I_a}$ denotes the diagonal matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ with $b_{ii} = a_i$ and all offdiagonal elements vanishing. For $\mathbf{B} \in \mathbb{R}^{n\times n}$, the trace of $\mathbf{B}$ is the sum of its diagonal elements: $\mathrm{tr}\,(\mathbf{B}) = \sum_{i=1}^{n} b_{ii}$.

For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \ge \mathbf{y}$ means $x_i \ge y_i$ for $1 \le i \le n$. $\mathbf{x} > \mathbf{y}$ means $\mathbf{x} \ge \mathbf{y}$ and there is at least one $j \in \{1, \dots, n\}$ such that $x_j > y_j$. $\mathbf{x} \gg \mathbf{y}$ means $x_i > y_i$ for $1 \le i \le n$ throughout.

2.2 Probability and Stochastic Processes

Let $(\Omega, \mathcal{F}, P)$ be a probability space. A family of σ-algebras $\{\mathcal{F}_t \mid t \ge t_0\}$ is called a *filtration* on $(\Omega, \mathcal{F})$ if $\mathcal{F}_t \subseteq \mathcal{F}_{t'} \subseteq \mathcal{F}$ for $t_0 \le t \le t'$. Here, both $t \in \mathbb{N}$ or $t \in \mathbb{R}_+$ are admissible. $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$ is called a *filtered probability space*. It satisfies the *usual conditions* if $\mathcal{F}$ is P-complete, $\mathcal{F}_0$ contains all P-nullsets of $\mathcal{F}$ and $\{F_t\}$ is right-continuous.

Let $X = \{X_t \mid t_0 \le t \le t_1\}$ be a stochastic process defined on Ω. If the range of the index t is clear we write $\{X_t\}$. If the sample points X_t are random variables in $\mathbb{R}$ we write $X_t \in \mathbb{R}$. If the sample points $\mathbf{X}_t$ are n-vectors of random variables we write $\mathbf{X}_t \in \mathbb{R}^n$. Given $\omega \in \Omega$, we call $\{X_t(\omega) \mid t_0 \le t \le t_1\}$ the *sample path* of the process X on ω.

X is called *adapted* to the filtration $\{\mathcal{F}_t\}$ if the random variable X_t is $\mathcal{F}_t$-measurable for every t. The filtration $\{\mathcal{F}_t^X\} = \{\sigma\{X_s \mid s \le t\} \mid t \ge t_0\}$ is called the *natural filtration* of X. If $\{\mathcal{F}_t\} = \{\mathcal{F}_t^X\}$ we say that $\{\mathcal{F}_t\}$ is generated by X. If a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$ appears without further comments, we assume that $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$

satisfies the usual conditions, and the stochastic process X under consideration generates $\{\mathcal{F}_t\}$. In particular, we assume $\mathcal{F}_0 = \{\Omega, \emptyset\}$. These definitions can be looked up in Borodin and Salminen (1996) or any textbook on stochastic processes. They apply to the discrete case ($t, t_0, t_1 \in \mathbb{N}$) as well as to the continuous case ($t, t_0, t_1 \in \mathbb{R}_+$). In the discrete case, i, j, k are the preferred index symbols (instead of t, u etc.).

For any event $A \in \mathcal{F}$, we write $P(A)$ for the probability of A under the measure P. For any random variable X, we write $\mathrm{E}_P(X)$ for the expectation of X under the measure P. If it is clear which measure is meant, we simply write $\mathrm{E}(X)$. Two measures P and Q are *equivalent* if they have the same nullsets. The indicator random variable for $A \in \mathcal{F}$ is denoted by 1_A.

2.3 Portfolios and Partial Portfolios

Let $\mathbf{X}$ be a vector of k random variables (i.e., a portfolio of k contingent claims), and let $\lambda \in \mathbb{R}^k$ be a position in $\mathbf{X}$. Let $M \subseteq \{1, \dots, k\}$, and let $i_1 < i_2 < \cdots < i_n$ be an enumeration of the n elements in M. The selection operator on $\mathbf{X}$ and λ is defined as follows:

$$\begin{aligned} \mathrm{select}(\mathbf{X}, M) &= (X_{i_1}, \dots, X_{i_n})^{\mathrm{T}} \\ \mathrm{select}(\lambda, M) &= (\lambda_{i_1}, \dots, \lambda_{i_n})^{\mathrm{T}} \end{aligned} \tag{2.2.1}$$

A vector $\mathbf{Y}$ of $k' \leq k$ random variables is called a *restriction* or *partial portfolio* of $\mathbf{X}$, in symbolic notation $\mathbf{Y} \subseteq \mathbf{X}$, if there is $M \subseteq \{1, \dots, k\}$, $|M| = k'$, such that $\mathbf{Y} = \mathrm{select}(\mathbf{X}, M)$. $\mathbf{Y}$ results by removing some claims from $\mathbf{X}$ (possibly none!). The interpretation of "=" and "<" is obvious.

The definition of a *restriction* or *partial position* of $\lambda' \in \mathbb{R}^{k'}$ is analoguous (in symbolic notation $\lambda' \subseteq \lambda$).

We write $(\mathbf{Y}, \lambda') \subseteq (\mathbf{X}, \lambda)$ if $\mathbf{Y} \subseteq \mathbf{X}$ and $\lambda' \subseteq \lambda$. In this case we re-use the term *partial portfolio.*

3 Continuous Time Finance

In this chapter we give a brief survey of continuous time finance. Since the dominant state variable in all models is the diffusion coefficient—the volatility—of the asset price process, we categorize models according to the nature of this coefficient. Models whose volatility coefficient does not exhibit randomness are treated in Sect. 3.1. Models whose volatility coefficient follows a stochastic process are discussed in Sect. 3.2.

The material presented in this chapter is standard. Uncertainty volatility models, on which the original work of this dissertation is grounded, are discussed in Chapter 4.

3.1 Deterministic Volatility

Most of our work is based on equity/FX Black-Scholes models. For this reason, Black-Scholes analysis is reviewed in rather more detail in the first half of this section.

In Chapter 10.2 we outline a client/server architecture for model calibration. Since our prototype calibrates to fixed income data, the second half of this section is dedicated to the HJM framework and the Vasicek short rate model.

3.1.1 One-Factor Black-Scholes Analysis

There are many ways to derive the Black-Scholes partial differential equation. Baxter and Rennie (1996), Duffie (1996), Hull (1993) and Wilmott *et al.* (1993) use the common approach based on stochastic calculus. Cox and Rubinstein (1985) show that the Black-Scholes formula can be interpreted as the continuous-time limit of the binomial random walk model.

Given is a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$ and a finite time horizon T. In this probability space, let $B = \{B_t\}$, $B_0 = 1$ be the price process of a riskless asset (B for b̲ond), and let $S = \{S_t\}$ be the security price process:

$$\begin{aligned} dB_t &= r_t B_t \\ dS_t &= S_t(\mu_t\,dt + \sigma_t\,dW) \end{aligned} \tag{3.3.1}$$

W is a Brownian motion and r_t, μ_t and σ_t are sufficiently well-behaved functions. Let X be a nonnegative $\mathcal{F}_T$-measurable random variable that represents the payoff structure of a contingent claim on S.

$$\phi_t = \frac{\mu_t - r_t}{\sigma_t} \tag{3.3.2}$$

and

$$\zeta_t = \exp\left\{ - \int_0^t \phi_u \,\mathrm{d}W_u - \frac{1}{2}\int_0^t \phi_u^2 \,\mathrm{d}u \right\} \tag{3.3.3}$$

define a martingale measure Q equivalent to P via $Q(A) = \mathrm{E}_P\left(\zeta_t 1_A\right)$ for all $A \in \mathcal{F}$. The arbitrage-free price π of the contingent claim X is given by

$$\pi = \mathrm{E}_Q\left(\beta_T X\right) \tag{3.3.4}$$

where $\beta = \{\beta_t\}$, $\beta_t = 1/B_T$, is the discount process belonging to B.

In order to *compute* π, a replicating strategy for X is constructed explicitely. Let $f(S_t, t)$ denote the (yet unknown) price of X at time t for security price S_t^1, with final value $f(S_T, T) = X$. Let $F = \{F_t\}$ be the associated price process: $F_t = f(S_t, t)$. Assume for the moment that f is twice differentiable. A partial differential equation for f can be determined as follows.

Define the $\mathbb{R}^2$-valued process $\theta = \{\theta_t\}$

$$\theta_t^0 = \beta_t(F_t - \frac{\partial}{\partial S} f(S_t, t) S_t) \quad \text{and} \quad \theta_t^1 = \frac{\partial}{\partial S} f(S_t, t) \tag{3.3.5}$$

θ replicates F and thus X: $\theta_t^0 B_t + \theta_t^1 S_t = F_t$. Now notice that, with Ito's formula,

$$dF_t = \left(\frac{\partial f}{\partial t} + \mu_t S_t \frac{\partial f}{\partial S} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 f}{\partial S^2} \right) dt + \sigma_t S_t \frac{\partial f}{\partial S} dW_t \tag{3.3.6}$$

This implies together with the definition of θ that the instantaneous change of the value of the portfolio $(\theta_t^0 B_t, \theta_t^1 S_t)$ is

$$\begin{aligned} &\theta_t^0 dB_t + \theta_t^1 dS_t = \\ &(\theta_t^0 \mu_t S_t + \theta_t^1 r_t B_t) dt + \theta_t^1 \sigma_t S_t dW_t = \\ &dF_t - \left(\frac{\partial f}{\partial t} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 f}{\partial S^2} + r_t S_t \frac{\partial f}{\partial S} - r_t F_t \right) dt = \\ &dF_t - \rho \, dt \end{aligned} \tag{3.3.7}$$

where ρ stands for the term in the brackets.

θ is self-financing only if $\rho = 0$, for in this case F becomes the value process corresponding to θ, and

$$F_t - F_0 = \int_0^t \theta_u^0 dB_u + \theta_u^1 dS_u \,\mathrm{d}u \tag{3.3.8}$$

holds.

It is the condition $\rho = 0$ which gives rise to the Black-Scholes partial differential equation

$$\frac{\partial f}{\partial t} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 f}{\partial S^2} + r_t S_t \frac{\partial f}{\partial S} - r_t f_t = 0 \tag{3.3.9}$$

with boundary condition

$$f(S_T, T) = X \tag{3.3.10}$$

Fact 3.1. *If there is no arbitrage, then the price function* $f\colon (0,\infty) \times [0,T] \to \mathbb{R}_+$ *for* X *satisfies* (3.3.9)*. In this case,* (3.3.5) *defines the replicating trading strategy.*

There is an intuitiv economic interpretation of (3.3.9): the difference of the return of a hedged option portfolio (the first two terms) and a bank account (the last two terms) must be zero. The prominent role of the volatility σ_t in the determination of the arbitrage-free price for S becomes clear after the derivation of (3.3.9) (μ_t, on the other hand, can be "hedged away").

We have $f(S_0, 0) = \pi$ and therefore $f(S_0, 0) = \mathrm{E}_Q(\beta_T X)$. Moreover,

$$F_t = \frac{1}{\beta_t}\mathrm{E}_Q(\beta_T X \mid \mathcal{F}_t) \tag{3.3.11}$$

This is sometimes called the *probabilistic solution* of (3.3.9).

3.1.2 Interest Rate Models

Interest-rate derivatives can in some sense be regarded as a bet on the future cost of money. The role of the security price process S is played by processes of bond prices, yields, spot or forward rates, depending on the focus of the model.

Terminology

Let $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$ be the underlying filtered probability space. $\mathbf{W} = \{\mathbf{W}_t\}$ is an N-dimensional Brownian motion on it; τ a finite time horizon. In this context, the symbol T usually denotes the maturity of a bond in the literature. We follow this convention here, but use T for other purposes in later sections.)

Assume a continuum of *discount bonds*, one for each maturity $T \leq \tau$. The time t-price of the bond with maturity T is denoted by $P(t,T)$, with terminal price $P(T,T) = 1$ (all bonds are normalized). The *instantaneous forward rate* at time t for borrowing at time T, $f(t,T)$, and the *yield*—the average implied interest rate—at time t of the bond maturing at time T, $R(t,T)$, fulfill

$$\begin{aligned} f(t,T) &= -\frac{\partial}{\partial T} \log P(t,T) \\ R(t,T) &= -\frac{\log P(t,T)}{T-t} \end{aligned} \tag{3.3.12}$$

for all $0 \leq t < T \leq \tau$, respectively. Solving for P, one gets

$$P(t,T) = \exp\left(-\int_t^T f(t,u)\, du\right) \tag{3.3.13}$$

The time t instantaneous forward rate, defined as

$$r_t = f(t,t) \tag{3.3.14}$$

is called the *spot rate*. Note that the spot rate is not sufficient to recover $P(t,T)$; the entire forward rate curve is needed.

It is assumed that there exists a *cash bond* $B = \{B_t\}$, whose stochastic differential equation is

$$dB_t = r_t B_t dt \tag{3.3.15}$$

B is the numeraire. With $B_0 = 1$, the solution of (3.3.15) is

$$B_t = \exp\left(\int_0^t r_u\, du\right) \tag{3.3.16}$$

for $0 \leq t \leq \tau$. Again, we define a discount factor $\beta_t = 1/B_t$.

The HJM Model

This no-arbitrage model by Heath *et al.* (1992) models the evolution of the entire forward rate curve, starting with a term-structure of interest rates observed in today's market. Jarrow (1996) is another comprehensive source.

For $0 \leq T \leq \tau$, let the $\mathbb{R}$-valued process $f^T = \{f_t^T\}$ denote the evolution of the time t forward rate for borrowing at time T: $f_t^T = f(t,T)$. The dynamics of the f^T are

$$f_t^T = f_0^T + \int_0^t \alpha_u^T(\omega)\,\mathrm{d}u + \sum_{i=1}^N \int_0^t \sigma_u^{Ti}(\omega)\,dW_u^i \tag{3.3.17}$$

for $0 \leq t \leq T$. Here $\{f_0^T = f(0,T) \mid 0 \leq T \leq \tau\}$ is a nonrandom initial forward rate curve, and the $\mathbb{R}$-valued processes α^T and σ^{Ti} $(0 \leq T \leq \tau,\ 1 \leq i \leq N)$ may depend on ω, are adapted to $\{\mathcal{F}_t\}$ and satisfy certain continuity, integrability and boundedness conditions. We will omit the argument ω to enhance readability. (In the literature, f_t^T is usually written $f(t,T)$, α_t^T is written $\alpha(t,T)$ and σ_t^{Ti} is written $\sigma_i(t,T)$. In order to be consistent with our earlier notation, we keep the current time t as a subscript, the index of the asset as first superscript and the index of the source of uncertainty as the second superscript.)

With (3.3.14) and (3.3.17), the spot rate process can be written as

$$r_t = f_0^t + \int_0^t \alpha_u^t\,\mathrm{d}u + \sum_{i=1}^N \int_0^t \sigma_u^{ti}\,dW_u^i \tag{3.3.18}$$

Ito's lemma together with some regularity conditions on B show that $P(t,T)$ is the solution of

$$dP(t,T) = P(t,T)\left[(r_t + b_t^T)dt + \sum_{i=1}^N a_t^{Ti} dW_t^i\right] \tag{3.3.19}$$

with

$$\begin{aligned} a_t^{Ti} &= -\int_t^T \sigma_u^{Ti}\,\mathrm{d}u \quad (1 \leq i \leq N) \\ b_t^T &= -\int_t^T \alpha_u^T\,\mathrm{d}u + \frac{1}{2}\sum_{i=1}^N \left(a_t^{Ti}\right)^2 \end{aligned} \tag{3.3.20}$$

b_t^T is the excess rate of return of the T-maturity bond at time t. The bond price processes $P(t,T)$ are not necessarily Markovian!

Under no-arbitrage assumptions, it is necessary to find an equivalent measure Q which makes the discounted bond price processes $\beta_t P(t,T)$ martingales, simultaneously for all $0 \leq T \leq \tau$. Heath *et al.* (1992) argue that it is sufficient to find such Q for a "basis" of N different bonds. It can furthermore be shown that Q, if it exists, is unique and does not depend on the choice of the basis.

After doing this, the spot rate r_t follows the process

$$r_t = f_0^t + \sum_{i=1}^{N} \int_0^t \sigma_u^{ti} \int_u^t \sigma_u^{vi} \, dv \, du + \sum_{i=1}^{N} \int_0^t \sigma_u^{ti} \, d\tilde{W}_u^i \tag{3.3.21}$$

where $\tilde{\mathbf{W}}$ is a Q-Brownian motion. In general, r_t is path dependent. Note that the drift α does not appear in (3.3.21).

Given the martingale measure Q, contingent claims X that mature at some time T are evaluated in standard fashion, with fair price $\pi = \mathrm{E}_Q\left(\beta_T X\right)$ and value process

$$X_t = \frac{1}{\beta_t} \mathrm{E}_Q\left(\beta_T X \mid \mathcal{F}_t\right) = \mathrm{E}_Q\left(\exp\left(-\int_t^T r_u \, du\right) X \,\middle|\, \mathcal{F}_t\right) \tag{3.3.22}$$

In particular,

$$P(t,T) = \mathrm{E}_Q\left(\exp\left(-\int_t^T r_u \, du\right) \,\middle|\, \mathcal{F}_t\right) \tag{3.3.23}$$

The Vasicek Short-rate Model

HJM offers a general framework that can be instantiated with specific drift and volatility coefficients. Vasicek (1977) proposes the one-factor model where the spot rate follows an Ornstein-Uhlenbeck mean-reverting process under the equivalent martingale measure Q:

$$dr_t = (\theta - \alpha \, r_t)dt + \sigma \, d\tilde{W}_t \tag{3.3.24}$$

with constants θ, α and σ. In terms of HJM, this means

$$\begin{aligned} \sigma_t^T &= \sigma \mathrm{e}^{-\alpha(T-t)} \\ f_0^T &= \theta/\alpha + \mathrm{e}^{-\alpha T}(r_0 - \theta/\alpha) - \frac{\sigma^2}{2\alpha^2}(1 - \mathrm{e}^{-\alpha T})^2 \end{aligned} \tag{3.3.25}$$

3.2 Stochastic Volatility

Some authors include under the concept of "stochastic volatility" the case where the coefficient $\sigma(S_t, t)$ of the asset price process

$$dS_t = \mu(S_t, t)\, dt + \sigma(S_t, t)\, d\mathbf{W}_t \tag{3.3.26}$$

depends on S_t. A time and/or space-heterogeneous yet deterministic volatility coefficient, however, merely makes the arithmetic more challenging and often precludes the existence of a closed-form solution; the argument from replication still goes through. The situation is different if σ undergoes random shocks which are "nontradable" in the economy. It is this case which is discussed in this section.

3.2.1 Tradable and Nontradable Factors

The following exposition is taken from Hofmann *et al.* (1992). Their work, in turn, draws from results presented in Föllmer and Schweizer (1991). Their model is general enough to include the concrete models of Sect. 3.2.2 as special cases.

Let $\mathbf{W} = \{\mathbf{W}_t\}$ be an N-dimensional Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$. Fix some time horizon T. Define the $\mathbb{R}^M$-valued process $\mathbf{X} = \{\mathbf{X}_t\}$ with component processes $X^1, \ldots, X^M$ by

$$d\mathbf{X}_t = \mu(\mathbf{X}_t, t)\, dt + \sigma(\mathbf{X}_t, t)\, d\mathbf{W}_t \tag{3.3.27}$$

where $\mu = (\mu^i\colon \mathbb{R}^M \to \times[0,T]\mathbb{R} \mid 1 \le i \le M)$ and $\sigma = (\sigma^{ij}\colon \mathbb{R}^M \times [0,T] \to \mathbb{R} \mid 1 \le i \le M, 1 \le j \le N)$ are functions satisfying appropriate regularity conditions. The component processes $X^1, \ldots, X^M$ may represent tradable assets or economic factors; trivially, there must be at least one tradable asset X^i and we assume $i = 1$ without loss of generality.

We also postulate the existence of an $\mathbb{R}_{++}$-valued process X^0 which plays the role of the riskless asset:

$$dX_t^0 = r(\mathbf{X}_t, t) X_t^0\, dt \quad \text{and} \quad X_0^0 = 1 \tag{3.3.28}$$

The discount factor $\beta = \{\beta_t\}$ is defined via $\beta_t = 1/X_t^0$, as usual.

Let the random variable Y on $(\Omega, \mathcal{F}_T)$ be a contingent claim. Standard procedure would imply a trading strategy $\theta = \{\theta_t\}$, $\theta_t \in \mathbb{R}^{M+1}$, for Y and a value process V^θ, which would then satisfy

$$V_t^\theta = \frac{1}{\beta_t} \mathrm{E}_Q \left(\beta_T Y \mid \mathcal{F}_t \right) \tag{3.3.29}$$

under some P-equivalent measure $Q \in \mathbb{P}$ which makes $\beta\mathbf{X}$ a martingale.

This is indeed the case if the economy is complete, i.e. $N = M$ and all components are tradable. In this case, θ exists and is self-financing, and Q and θ are uniquely determined by μ, σ and Y_T.

In the general, incomplete situation, this need not be so. We would certainly wish $\theta^i \equiv 0$ to hold for all nontradable components i. However, this restriction might make a self-financing replicating strategy impossible. There are several ways out of this dilemma. Schweizer (1991) discusses "mean-self financing" strategies; here, we briefly summarize some fixes which are more concrete.

3.2.2 Some Concrete One-dimensional Models

We present some concrete models based on a ome-dimensional asset price process and stochastic volatility. The models differ in how they supplement no-arbitrage theory. Hull-White and Wiggins, for instance, advance equilibrium arguments, while others try to exploit ad-hoc hedging opportunities in the Black-Scholes spirit.

Let $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$ be a filtered probability space and T a fixed, finite time horizon. Let $\mathbf{W} = \{\mathbf{W}_t\}$, $\mathbf{W}_t \in \mathbb{R}^2$, be a two-dimensional Brownian motion with correlation coefficient ρ, or $\mathrm{E}_P \left(dW^1 dW^2 \right) = \rho\, dt$. (At this point, we deviate from our standard assumption that the component processes of $\mathbf{W}$ are independent, i.e. $\rho = 0$.)

There is a riskless asset $X^0 = \{X_t^0\}$ with $X_0^0 = 1$ and $X_t^0 = \mathrm{e}^{rt}$. r is the riskless rate and $\beta = 1/X^0$ the discount process, as usual.

Hull and White's Model

Hull and White (1987) propose the following model:

$$\begin{aligned} dS_t &= S_t(r\, dt + \sigma_t\, dW^1) \\ d\sigma_t^2 &= \sigma_t^2 \left(\xi(\sigma_t^2, t)\, dt + \phi(\sigma_t^2, t)\, dW^2 \right) \end{aligned} \tag{3.3.30}$$

where ξ and ϕ may depend on σ^2 and t, but not on S. Under the additional assumption that (a) $\rho = 0$ and (b) σ^2 does not have systematic risk (a statement we shall not explain further at this point), a partial differential equation slightly more complex than (3.3.9) can be derived by using CAPM equilibrium arguments, eliminating randomness and therefore precluding any risk preferences.

Now define the mean volatility V over a particular path $\{\sigma_t^2\}$ as

$$V = \frac{1}{T} \int_0^T \sigma_t^2 \, \mathrm{d}t \tag{3.3.31}$$

For any attainable contingent claim X, let

$$\pi(V) = \mathrm{E}_P \left(\beta_T X \mid \sigma^2 \equiv V \right) \tag{3.3.32}$$

denote the fair price of X under the restricted scenario where $\sigma_t^2 = V$ for $0 \leq t \leq T$. Then it can be shown that the no-arbitrage price of π is

$$\pi = \int_{-\infty}^{\infty} \pi(V) \, h(V \mid \sigma_0^2) \, dV \tag{3.3.33}$$

where $h(V \mid \sigma_0^2)$ is the density of V conditional on σ_0^2 under P. In other words, the price of a contingent claim X turns out to be the weighted average Black-Scholes price for any realizable mean volatility. This result does not hold if $\rho \neq 0$ or ξ or ϕ depend on S.

Wiggins' Model

The model advocated in Wiggins (1987) has the dynamics

$$\begin{aligned} dS_t &= S_t(\mu \, dt + \sigma_t \, dW^1) \\ d\sigma_t &= h(\sigma_t) \, dt + \phi \, \sigma_t \, dW^2 \end{aligned} \tag{3.3.34}$$

It is not required that $\rho = 0$. Let $F = \{F_t\}$, $F_t = f(S_t, t)$, be the value process of a contingent claim X. Wiggins defines a hedge portfolio $\theta = \{\theta_t\}$, $\theta_t \in \mathbb{R}^2$, in the riskless asset and the security by

$$\begin{aligned} \theta_t^0 &= \beta_t \left(F_t - \frac{\partial}{\partial S} f(S_t, t) S_t - \rho\phi \frac{\partial}{\partial \sigma} f(S_t, t) \right) \\ \theta_t^1 &= \frac{\partial}{\partial S} f(S_t, t) + \frac{\rho\phi}{S_t} \frac{\partial}{\partial \sigma} f(S_t, t) \end{aligned} \tag{3.3.35}$$

θ is a modification of (3.3.5) with the property that its value process V^θ satisfies

$$\frac{dV_t^\theta}{V_t^\theta} \frac{dS_t}{S_t} = 0 \tag{3.3.36}$$

for $0 \leq t \leq T$. I.e., the return of the hedge portfolio is uncorrelated with the return of the security. If S is an index on the market, the hedge portfolio has therefore a zero beta coefficient. Under some additional economic assumptions and for the special case that S is indeed a contingent claim on the market, f is a solution to the partial differential equation

$$\mathrm{BS} + \frac{1}{2}\sigma_t^2\phi^2\frac{\partial^2 f}{\partial\sigma^2} + \rho\,\phi\,\sigma_t^2 S_t\frac{\partial f^2}{\partial S\partial\sigma} + \frac{\partial f}{\partial\sigma}\left(h(\sigma_t) - \rho\,\theta\sigma_t^2\right) = 0 \tag{3.3.37}$$

where BS stands for the left side terms of the Black-Scholes partial differential equation (3.3.9).

Johnson and Shanno's Model

Johnson and Shanno (1987) choose the model

$$\begin{aligned} dS_t &= S_t(\mu\,dt + \sigma_t S_t^{\alpha_1-1}\,dW^1) \\ d\sigma_t &= \sigma_t(\xi\,dt + \phi\sigma_t^{\alpha_2-1}\,dW^2) \end{aligned} \tag{3.3.38}$$

with $\alpha_1, \alpha_2 \geq 0$. The correlation coefficient between W^1 and W^2 is ρ. Setting up the Black-Scholes hedge portfolio θ as in (3.3.5), one finds that the value process V^θ of θ satisfies

$$dV_t^\theta = dF_t - (\mathrm{BS} + \mathrm{JS})dt - \phi\sigma_t^{\alpha_2}\frac{\partial f}{\partial\sigma}\frac{\partial f}{\partial S}dW^2 \tag{3.3.39}$$

where BS represents the standard Black-Scholes terms—see (3.3.9)—and JS stands for additional nonrandom terms which are easy to derive with Ito calculus. At this point, Johnson and Shanno assume that the dW^2 term can be diversified away (this assumption replaces the equilibrium principles in the previous two models), and get a partial differential equation $\mathrm{BS} + \mathrm{JS} = 0$ with appropriate boundary conditions for X.

Scott's Model

Scott (1987) uses a model in which the volatility follows a mean-reverting process with mean $\bar{\sigma}$:

$$\begin{aligned} dS_t &= S_t(\mu\,dt + \sigma_t\,dW^1) \\ d\sigma_t &= \xi(\bar{\sigma} - \sigma)\,dt + \phi\,dW^2 \end{aligned} \tag{3.3.40}$$

Again, ρ is the correlation coefficient between dW^1 and dW^2. Assume there are two contingent claims, X and Y, with price functions f and g, respectively, and price processes $F = \{F_t\}$ and $G = \{G_t\}$. Assume furthermore that X expires at time $T_X \leq T$, and Y expires at time $T_X < T_Y \leq T$. A trading strategy θ that hedges a portfolio of X and Y (with dynamic weights) during times $0 \leq t \leq T_X$ gives rise to a partial differential equation

$$\begin{aligned}\frac{\partial g}{\partial \sigma}&\left(\mathrm{BS}_f + \rho\,\phi\,\sigma_t S_t \frac{\partial f^2}{\partial S \partial \sigma} + \frac{1}{2}\phi^2 \frac{\partial f^2}{\partial \sigma^2}\right)\\ &- \frac{\partial f}{\partial \sigma}\left(\mathrm{BS}_g + \rho\,\phi\,\sigma_t S_t \frac{\partial g^2}{\partial S \partial \sigma} + \frac{1}{2}\phi^2 \frac{\partial g^2}{\partial \sigma^2}\right) = 0\end{aligned} \tag{3.3.41}$$

which does no longer have terms in dW^1 or dW^2. BS_f and BS_g represent the standard Black-Scholes terms corresponding to f and g terms as they appear in (3.3.9).

However, the PDE in (3.3.41) does not have a unique solution for given boundary conditions at $t = T_X$. There are two ways in which this situation can be resolved:

- Equilibrium arguments can be applied. This approach is chosen in Scott (1987) and leads to a partial differential equation for X which depends on λ^*, the risk premium associated with $d\sigma$.

- If the price for the claim Y is known (for instance, if Y is a liquid option), one can model Y's price process G as a geometric Brownian motion diffusion with the constant volatility implied by Y's price. This path is explored—based on a slightly different model for $d\sigma$—in Zhu and Avellaneda (1997).

A theoretical third possibility is to postulate the existence of an asset whose price is perfectly correlated with $d\sigma$.

4 Scenario-based Evaluation and Uncertainty

The following problems arise as soon as arbitrage pricing theory is applied in practice:

- Plausible values for volatility and other coefficients must be found to instantiate the chosen model.
- Once instantiated, models often prove too weak to represent the market dynamics adequately; in the case of Black-Scholes, this deficiency shows itself in the often cited implied volatility smile.

It is natural to try to cure the second problem by introducing time- and space-dependency in the volatility and other coefficients. If this leads to randomness in the evolution of the volatility, one has created a stochastic volatility model. The first problem still looms large, however, and some sort of parameter calibration becomes necessary before the stochastic volatility model can be applied.

Uncertain volatility takes a different approach. Instead of chosing a fixed set of a priori model coefficients, agents specify priorities which they would like to see applied when a given portfolio is evaluated under the model. These priorities are initially stated "in prose" and have some economic function. They usually correspond to stochastic control problems and require dynamic programming methods for their solution.

4.1 Preliminaries

Definition 4.1 (Scenario). *We call a set of (declarative) agent priorities and the (imperative) evaluation rules they imply a* scenario*.*

Definition 4.2 (Uncertain coefficients). *Model coefficients which are variable under a given scenario are called* uncertain*. The evaluation rules of the scenario control the instantiation of uncertain coefficients, locally or globally.*

These definitions are not strictly formal. The soundness of the concept needs to be established for each concrete scenario. In this thesis, we restrict ourselves to two scenarios:

- the worst-case volatility scenario;
- the volatility-shock scenario.

$$\begin{tikzcd}
 & \text{(Patterns for model coefficients)} \arrow[d] & \\
\text{Scenario} \arrow[r] & \bigcirc \arrow[d] & \text{Portfolio} \arrow[l] \\
 & \textit{Instantiated model coefficients} &
\end{tikzcd}$$

Figure 4.1: Both scenario and portfolio are required components when model coefficients are instantiated. Model coefficients can, but must not, be restricted by patterns

We review the foundations of the former and its companion, the Uncertain Volatility Model (UVM) by Avellaneda and Paràs, in this chapter. Algorithmic issues of worst-case scenarios are moved as original work to Part II. The volatility shock scenario is an extension of the worst-case scenario and is discussed, also as original work, in Chapter 8 of Part II.

The benefit of the scenario approach is clear: no a-priori choice of model coefficients has to be made. Furthermore, once evaluation rules have been applied to instantiate uncertain coefficients, we're back in the realm of arbitrage pricing theory. On the other hand, as seen in Sect. 3.2, no-arbitrage arguments alone are not sufficient when coefficients are stochastic; disputable assumptions, equilibrium arguments and other methods which are not easily generalizable are required to complete the job.

The scenario approach may yield different instantiations of model coefficients for different portfolios. Figure 4.1 shows how scenario and portfolio are both taken into account when the evaluation rules of the scenario are executed.

The separation into model and scenario is in fact strong enough to reappear in the object-oriented implementation in Part III. Models, scenarios and portfolios all have associated class hierarchies.

In this thesis, we exclusively focus on the volatility as the only uncertain coefficient. Formally, we assume a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, P)$, a one-dimensional Brownian motion W, and some finite time horizon T. In this probability space, let $S = \{S_t\}$ be a security price process with the stochastic differential equation

$$\frac{dS_t}{S_t} = \mu(S_t, t)\,dt + \sigma(S_t, t)\,dW \tag{4.4.1}$$

Let $r\colon [0,T] \to \mathbb{R}_+$ be the time-dependent interest rate, and $\beta = \{\beta_t\}$ the corresponding discount process:

$$\beta_t = \int_0^t \mathrm{e}^{-r_s}\, ds \tag{4.4.2}$$

We assume r and μ are continuous functions that are sufficiently well behaved for our purpose. $\sigma\colon (0,\infty) \times [0,T] \to \mathbb{R}_{++}$ is our uncertain model coefficient.

Definition 4.3 (Candidate set and scenario measure). *A set*

$$\mathcal{C} \subseteq \{\sigma \mid (4.4.1) \textit{ has a solution}\} \tag{4.4.3}$$

is called a candidate set *for σ. For each $\sigma \in \mathcal{C}$ there exists a unique measure $Q(\sigma)$ which makes βS a martingale: we say $Q(\sigma)$ is the* scenario measure *for σ.*

Sometimes we also refer to the "scenario σ" or "scenario volatility." The candidate set implements the optional pattern for the uncertain coefficient referred to in Fig. 4.1.

Let the nonnegative, continuous random variable X denote the payoff of a contingent claim at time T. The no-arbitrage price of the contingent claim for fixed σ follows the process

$$F_t(X,\sigma) = \frac{1}{\beta_t}\, \mathrm{E}_{Q(\sigma)}\left(\beta_T X \mid \mathcal{F}_t\right) \tag{4.4.4}$$

Extension to portfolios of contingent claims is straightforward. Let $\mathbf{X} = (X_1, \ldots, X_k)^{\mathrm{T}}$ be a set of $k > 0$ nonnegative contingent claims—a portfolio!—on $(\Omega, \mathcal{F})$, all maturing at time T. (The theory can be easily generalized to contingent claims with different expiration dates.) For any combined position $\lambda = (\lambda_1, \ldots, \lambda_k)^{\mathrm{T}} \in \mathbb{R}^k$, $\lambda \cdot \mathbf{X}$ is also a—not necessarily nonnegative—random variable on $(\Omega, \mathcal{F})$ and represents the final cashflow at time T for the holder of the portfolio. (At this time we assume that contingent claims are not path-dependent; i.e., their payoff can be written as $g(S_T)$ for some function g. Later, of course, we will include barrier and American options.) The value process $F = \{F_t\}$ is extended to cover combined positions through

$$F_t(\lambda \cdot \mathbf{X}, \sigma) = \sum_{i=1}^{k} \lambda_i F_t(X_i, \sigma) \tag{4.4.5}$$

$$\begin{tikzcd}
& \mathcal{C} \arrow[d] & \\
\textit{Worst-case scenario} \arrow[r] & \bigcirc \arrow[d] & (\lambda, \mathbf{X}) \arrow[l] \\
& \sigma\colon (0,\infty) \times [0,T] \to \mathbb{R}_{++} &
\end{tikzcd}$$

Figure 4.2: The generic terms of Fig. 4.1 filled in. The worst-case scenario can be tailored to pricing, hedging or calibration situations as described in the text

4.2 The Worst-case Volatility Scenario

We distinguish three concrete worst-case volatility scenarios, or worst-case scenarios for short, each illuminating the exposure to volatility risk from a slightly different perspective. All scenarios have in common that

$$\mathcal{C} = \{\sigma \mid \sigma_{\min} \le \sigma(S_t, t) \le \sigma_{\max} \text{ and (4.4.1) has a solution}\} \qquad (4.4.6)$$

where $0 < \sigma_{\min} \le \sigma_{\max}$ represents a prescribed bound. For simplicity, we assume constant bounds, but the theory holds for time-heterogeneous bounds as well. Figure 4.2 illustrates the flow of information that leads from $\mathcal{C}$, $(\lambda, \mathbf{X})$ and the concrete scenario to the selection of $\sigma \in \mathcal{C}$.

The agent priorities in each of the worst-case scenario variations can be informally stated as follows:

Worst-case pricing. Given the portfolio $\mathbf{X}$ and a position $\lambda \in \mathbb{R}^k$ in $\mathbf{X}$. Which $\hat{\sigma} \in \mathcal{C}$ maximizes today's value $F_0(\lambda \cdot \mathbf{X}, \sigma)$?

The optimal hedge-portfolio. Given two portfolios $\mathbf{X}$ and $\bar{\mathbf{X}}$ of k resp. $\bar{k}$ contingent claims, and a position $\lambda \in \mathbb{R}^k$ in $\mathbf{X}$. For each $\bar{X}_i$, $1 \le i \le \bar{k}$, a "market price" $\bar{\pi}_i$ is known. (Assume, for instance, that the $\bar{X}_i$ are traded frequently, and the X_i are exotic over-the-counter instruments.) Which $\hat{\sigma} \in \mathcal{C}$ maximizes $F_0(\lambda \cdot \mathbf{X}, \sigma)$ under the additional constraint that $F_0(\bar{X}_i, \hat{\sigma}) = \bar{\pi}_i$ for $1 \le i \le \bar{k}$?

Calibration. Given a portfolio $\bar{\mathbf{X}}$ of $\bar{k}$ contingent claims, and market prices $\bar{\pi}_i$ for all $\bar{X}_i$, $1 \le i \le \bar{k}$. Fix a subjective "prior" $\bar{\sigma} \in \mathcal{C}$. Which $\hat{\sigma} \in \mathcal{C}$ minimizes $\|\sigma - \bar{\sigma}\|$

under the additional constraint that $F_0(\bar{X}_i, \hat{\sigma}) = \bar{\pi}_i$ for each $1 \leq i \leq \bar{k}$? We leave the semantics of the distance $\|\cdot\|$ unspecified.

Section 4.2.1 is dedicated to the the worst-case pricing problem. Section 4.2.2 is a short treatise on the problem of finding the optimal hedge portfolio. Section 4.2.3 investigates calibration issues.

Here and throughout the rest of the work, optimality is denoted by a "^" accent.

4.2.1 Worst-case Pricing

The objective is to find the volatility coefficient $\hat{\sigma} \in \mathcal{C}$ which maximizes $F_0(\lambda \cdot \mathbf{X}, \sigma)$ for a given vector $\mathbf{X}$ of k contingent claims, and given position $\lambda \in \mathbb{R}^k$. *Sellers* of $\lambda \cdot \mathbf{X}$ are completely hedged against volatility risk within the bounds (4.4.6) if they charge at least $F_0(\lambda \cdot \mathbf{X}, \hat{\sigma})$. (From this point of view, $\lambda_i > 0$ means X_i is sold, and $\lambda_i < 0$ means X_i is bought. Positive quantities signify liabilities of the seller, while negative quantities signify cash inflow.)

The objective must be formalized with care, since $\hat{\sigma}$ may not exist. For instance, assume the final payoff $\lambda \cdot \mathbf{X}$ is convex and continuous, and $\mathcal{C} = \{0.2 - \frac{1}{n} \mid n \geq 6\}$. It is clear that $F_0(\lambda \cdot \mathbf{X}, 0.2 - \frac{1}{n}) \to F_0(\lambda \cdot \mathbf{X}, 0.2)$ from below as $n \to \infty$, yet $0.2 \notin \mathcal{C}$. Nevertheless, $F_0(\lambda \cdot \mathbf{X}, 0.2)$ should be regarded as the worst-case price, and $\sigma = 0.2$ as its its scenario coefficient.

Convex Contingent Claims

It is instructive to consider the simple case of convex portfolios first. Let $Y = \lambda \cdot \mathbf{X}$, and assume Y can be written $g(S_T(\omega)) = Y(\omega)$ for $\omega \in \Omega$ and some nonnegative convex function $g\colon (0, \infty) \to \mathbb{R}_+$. (For instance, $\mathbf{X}$ might be a vector of European call or put options, with positions $\lambda_i > 0$ throughout). In this case, the Black-Scholes solution is also convex in S. Jeanblanc-Picque *et al.* (1991) conclude

Fact 4.4. *For convex Y, the value processes $F(Y, \sigma_{\max})$ and $F(Y, \sigma_{\min})$ form a super- resp. submartingale under any measure $Q(\sigma)$ with $\sigma \in \mathcal{C}$. This implies*

$$F_t(Y, \sigma_{\min}) \;\leq\; F_t(Y, \sigma) \;\leq\; F_t(Y, \sigma_{\max}) \tag{4.4.7}$$

for $0 \leq t \leq T$ and for all $\sigma \in \mathcal{C}$.

For a nonnegative convex overall position Y, the solution of the maximization problem is thus $\hat{\sigma} = \sigma_{\max}$. Similarly, if Y is negative and concave, $|Y|$ is positive and convex, and $F_t(Y, \sigma) \leq F_t(Y, \sigma_{\min})$ for all $\sigma \in \mathcal{C}$.

General Portfolios

Let $Y = \lambda \cdot \mathbf{X}$ be the liability structure at time T for a portfolio $\mathbf{X}$ of k contingent claims and position $\lambda \in \mathbb{R}^k$. This time we make no assumptions about $Y : \Omega \to \mathbb{R}$. Avellaneda and Parás (1995) generalize Fact 4.4 as follows:

Fact 4.5. *Given Y, define a value process $\hat{F}(Y) = \{\hat{F}_t(Y)\}$ by $\hat{F}_t(Y) = \hat{f}(S_t, t; Y)$, where $\hat{f}$ is the solution of the partial differential equation*

$$\frac{\partial f}{\partial t} + \frac{1}{2}\Sigma\left(\frac{\partial^2 f}{\partial S^2}\right) S_t^2 \frac{\partial^2 f}{\partial S^2} + r_t S_t \frac{\partial f}{\partial S} - r_t f_t = 0 \tag{4.4.8}$$

with boundary condition $\hat{f}(S_T, T) = Y(S_T)$ and

$$\Sigma(x) = \begin{cases} \sigma_{\max} & \text{if } x \geq 0 \\ \sigma_{\min} & \text{if } x < 0 \end{cases} \tag{4.4.9}$$

Then $\hat{F}(Y)$ is a supermartingale under any measure $Q(\sigma)$ where $\sigma \in \mathcal{C}$.

The informal rationale is the following: take the original Black-Scholes equation (3.3.9) and bring $r_t f_t$ to the right side, while observing that the remaining terms on the left side do not contain f. To make f as large as possible, we maximize the only term on the left side which has some degree of freedom: $\frac{1}{2}\sigma S_t^2 \frac{\partial^2 f}{\partial S^2}$. This is accomplished in (4.4.9).

Fact 4.6. *Let $\hat{F}(Y)$ be the value process for Y defined in Fact 4.5. Then*

$$\hat{F}_0(Y) = \sup_{\sigma \in \mathcal{C}} F_0(Y, \sigma) \tag{4.4.10}$$

Moreover, the σ which yields the supremum is given by (4.4.9).

Thus, there actually exists a "scenario $\hat{\sigma}$", and it can be constructed locally.

Fact 4.7. *For $c \in \mathbb{R}_{++}$ and two liability structures $Y = \lambda \cdot \mathbf{X}$ and $Z = \lambda' \cdot \mathbf{X}'$,*

$$\begin{aligned} \hat{F}_t(cY) &= c\hat{F}_t(Y) \\ \hat{F}_t(Y + Z) &\leq \hat{F}_t(Y) + \hat{F}_t(Z) \\ \hat{F}_t(Y + Z) &\geq \hat{F}_t(Y) - \hat{F}_t(-Z) \end{aligned} \tag{4.4.11}$$

Thus, positions may be scaled, but $\hat{F}$ is nonlinear and sub-additive. (The third statement follows from the second with $F_t(Y) = F_t(Y + Z - Z) \leq F_t(Y + Z) + F_t(-Z)$). Notice also that Fact 4.7 is vaiud for $0 \leq t \leq T$, not just for $t = 0$.

4.2.2 The Optimal Hedge Portfolio

Let $\mathbf{X}$ and $\bar{\mathbf{X}}$ be two portfolios of size k and $\bar{k}$, respectively. Assume furthermore that $\lambda \in \mathbb{R}^k$ is a position for X, and $\bar{\pi} \in \mathbb{R}^{\bar{k}}_{++}$ is a market price vector for $\bar{\mathbf{X}}$. ($\mathbf{X}$ might be a book position, and $\bar{\mathbf{X}}$ might be a set of liquid options.) It is a natural restriction to consider only those $\sigma \in \mathcal{C}$ under whose scenario measure $Q(\sigma)$ the prices $\bar{\pi}$ for $\bar{\mathbf{X}}$ are matched. This restriction on $\mathcal{C}$ is defined as follows:

$$\mathcal{C}' = \{\sigma \in \mathcal{C} \mid F_0(\bar{X}_i, \sigma) = \bar{\pi}_i \text{ for } 1 \leq i \leq \bar{k}\} \tag{4.4.12}$$

Now let $Y = \lambda \cdot \mathbf{X}$ be the combined payoff of portfolio $\mathbf{X}$. Avellaneda and Parás (1996) show

Fact 4.8. *Given $\mathbf{X}$, $\bar{\mathbf{X}}$, λ and $\bar{\pi}$. Assume $\hat{\lambda} \in \mathbb{R}^{\bar{k}}$ is a finite solution of the optimization problem*

$$\inf_{\bar{\lambda} \in \mathbb{R}^{\bar{k}}} \left\{ \sup_{\sigma \in \mathcal{C}} F_0(Y + \bar{\lambda} \cdot \bar{\mathbf{X}}, \sigma) - \bar{\lambda} \cdot \bar{\pi} \right\} \tag{4.4.13}$$

Let $\hat{\sigma}$ be the scenario volatility for $\hat{\lambda}$ as in Fact 4.6:

$$F_0(Y + \hat{\lambda} \cdot \bar{\mathbf{X}}, \hat{\sigma}) = \sup_{\sigma \in \mathcal{C}} F_0(Y + \hat{\lambda} \cdot \bar{\mathbf{X}}, \sigma) \tag{4.4.14}$$

Then

$$F_0(Y, \hat{\sigma}) = \sup_{\sigma \in \mathcal{C}'} F_0(Y, \sigma) \tag{4.4.15}$$

The solution $\hat{\lambda}$ is unique, since the function

$$h(\bar{\lambda}) = \sup_{\sigma \in \mathcal{C}} F_0(Y + \bar{\lambda} \cdot \bar{\mathbf{X}}, \sigma) - \bar{\lambda} \cdot \bar{\pi} \tag{4.4.16}$$

is convex and has therefore at most one minimum. Furthermore, under first-order conditions on optimality,

$$\frac{\partial}{\partial \bar{\lambda}_i} \left(F_0(Y + \bar{\lambda} \cdot \bar{\mathbf{X}}, \hat{\sigma}) - \bar{\lambda} \cdot \bar{\pi} \right) \bigg|_{\hat{\lambda}_i} = F_0(\bar{X}_i, \hat{\sigma}) - \bar{\pi}_i = 0 \tag{4.4.17}$$

and therefore $F_0(\bar{X}_i, \hat{\sigma}) = \bar{\pi}_i$, for $1 \leq i \leq \bar{k}$.

The position $\hat{\lambda}$ is optimal in the sense that no other position reduces the worst-case residual liability $h(\bar{\lambda})$ by a larger amount. An agent who counterbalances a stake in $\mathbf{X}$ by taking an offsetting position $\hat{\lambda}$ in $\bar{\mathbf{X}}$ needs at most $h(\bar{\lambda})$ additional cash to hedge the combined position, provided the volatility does not leave $\mathcal{C}$. $\hat{\lambda}$ can thus be regarded as the optimal hedge portfolio under the worst-case scenario.

4.2.3 Calibration

Calibration is not the main task of this thesis, although we sketch an online calibrator for the money market in the context of software engineering topics in Chapter 10.2. Calibration naturally motivates, however, further application of the optimization techniques described in the previous section, and shall thus be awarded a few lines.

The goal of calibration is to find an instantiation of the uncertain coefficients that matches observed prices of market instruments exactly. In that sense, the optimal hedge portfolio results from calibrating σ to the market prices $\bar{\pi}$. The method, however, is less satisfactory since it depends on the presence of a book portfolio $\mathbf{X}$. Furthermore, agents cannot introduce subjective prior beliefs about uncertain coefficients; in fact, the resulting scenario σ takes on only extremal values $\sigma_{\min}$ and $\sigma_{\max}$.

For this reason, let us reformulate the problem. Given a portfolio $\bar{\mathbf{X}}$ and a corresponding price vector $\bar{\pi} \in \mathbb{R}^{\bar{k}}_{++}$, choose some (constant) prior $\bar{\sigma} \in \mathcal{C}$ that best reflects your subjective beliefs about the volatility of the underlying asset.

For any $\sigma \in \mathcal{C}$ and for any $\omega \in \Omega$, define the distance of σ to $\bar{\sigma}$ on the path $\{S_t(\omega) \mid 0 \leq t \leq T\}$ as

$$d(\sigma, \omega) = \int_0^T \eta \left(\sigma(S_u(\omega), u)^2 \right) \, \mathrm{d}u \tag{4.4.18}$$

where η is a smooth, finite, strictly convex function which attains its minimum at $\bar{\sigma}^2$, i.e. $\eta(\bar{\sigma}^2) = 0$. η is called *pseudo entropy function* and implements a penalty for deviation from the prior—for instance, take $\eta(\sigma^2) = \frac{1}{2}(\sigma^2 - \bar{\sigma}^2)^2$.

With $\mathcal{C}'$ as defined in (4.4.12), Avellaneda *et al.* (1997) extend Fact 4.8 by showing

Fact 4.9. *Given $\bar{\mathbf{X}}$ and $\bar{\pi}$. Assume $\hat{\lambda} \in \mathbb{R}^{\bar{k}}$ is a finite solution of the optimization problem*

$$\inf_{\bar{\lambda} \in \mathbb{R}^{\bar{k}}} \left\{ \sup_{\sigma \in \mathcal{C}} F_0(-d(\sigma) + \bar{\lambda} \cdot \bar{\mathbf{X}}, \sigma) - \bar{\lambda} \cdot \bar{\pi} \right\} \tag{4.4.19}$$

and let $\hat{\sigma} \in \mathcal{C}$ be the scenario volatility for $\hat{\lambda}$. Then

$$F_0(-d(\hat{\sigma}), \hat{\sigma}) = \sup_{\sigma \in \mathcal{C}'} F_0(-d(\sigma), \sigma) \tag{4.4.20}$$

In other words, $\hat{\sigma}$ minimizes the penalty. Again, the solution $\hat{\lambda}$ is unique.

Computation of $h(\bar{\lambda})$

In the case of the optimal hedge portfolio, $h(\bar{\lambda})$ is computed by solving (4.4.8). This approach needs to be modified for calibration.

For fixed η, define the *flux function*

$$\Phi(x) = \sup_{\sigma} \left(\sigma^2 x - \eta(\sigma^2)\right) \tag{4.4.21}$$

where the supremum is taken over $(\sigma_{\min}, \sigma_{\max})$ and attained at $\sigma = \Phi'(x)$. With $\bar{Y} = \bar{\lambda} \cdot \bar{\mathbf{X}}$ for fixed $\bar{\lambda} \in \mathbb{R}^{\bar{k}}$, define the process $G = \{G_t\}$ as

$$G_t = \sup_{\sigma \in \mathcal{C}} F_t(-d(\sigma) + \bar{Y}, \sigma) \tag{4.4.22}$$

Fact 4.10. *Given G and $\bar{Y}$. Then $G_t = g(S_t, t)$, where g is the solution of the partial differential equation*

$$\frac{\partial g}{\partial t} + \frac{1}{\beta_t} \Phi\left(\frac{\beta_t}{2} S_t^2 \frac{\partial^2 g}{\partial S^2}\right) + r_t\, S_t \frac{\partial g}{\partial S} - r_t g_t = 0 \tag{4.4.23}$$

with boundary condition $g(S_T, T) = \bar{Y}(S_T)$. The supremum in (4.4.22) is realized at

$$\sigma(S_t, t) = \sqrt{\Phi'\left(\frac{\beta_t}{2}\, S_t^2 \frac{\partial^2 g}{\partial S^2}\right)} \tag{4.4.24}$$

By construction, $h(\bar{\lambda}) = G_0$.

Notice that (4.4.23) is not the pricing equation for $\bar{Y}$; the pricing equation for $\bar{Y}$ is obtained by replacing Φ with $\frac{\Phi'}{2}\, S_t^2 \frac{\partial^2 g}{\partial S^2}$.

The PDE (4.4.23) can be solved with finite difference methods. We will get back to calibration issues briefly at the end of Part III, in Chapter 10.2, although there the stage will be the money market (not the equity/FX market), and the numerical tool will be Monte Carlo simulation.

Other Approaches

Calibration or the problem of fitting parameters of stochastic models to market data has been studied for some time. Breeden and Litzenberger (1978) observe that the price of a binary option X with $X = 1$ if $K_1 \leq S_T \leq K_2$ and $X = 0$ otherwise must be

$$\int_{K_1}^{K_2} \frac{\partial^2}{\partial K^2} C(K)\, dK \tag{4.4.25}$$

where $C\colon (0, \infty) \to \mathbb{R}_{++}$, $K \to C(K)$, is the pricing function for a continuum of call options on the asset, with strike price K and expiration date T. This result stems from no-arbitrage arguments involving butterfly spreads and is valid regardless of the stochastic model. The price of any contingent claim can be recovered in a similar way from such a curve C, provided sufficient market data is available.

In a recent study, Jackwerth and Rubinstein (1996) minimize the distance between a prior probability distribution for S_T and a posterior distribution compatible with prices of contingent claims in a one-period setting, using a variety of objective functions. Among others, least-squares, absolute variation, maximum entropy and smoothness criteria are tested, the latter not requiring a prior distribution. The least-squares approach is based on an earlier paper, Rubinstein (1995). Pirkner *et al.* (1999) suggest to model the terminal return density of the stock with a mixture of normal distributions.

Lagnardo and Osher (1997) use a gradient descent procedure to minimize the functional

$$F(\sigma) = \|\, |\nabla\sigma|\, \|_2^2 + \nu(F_0(\bar{\lambda} \cdot \bar{\mathbf{X}}, \sigma) - \bar{\lambda} \cdot \bar{\pi})^2 \tag{4.4.26}$$

where $\nu > 0$ is a constant and controls the rate of convergence in a numerical procedure.

4.3 Scenarios and Nonlinearity

In general, worst-case scenarios lead to nonlinear solutions and may be asymmetric for the buy and sell side.

In economic terms, nonlinearity is due to risk-diversification under mixed convexity. Any position λ in $\mathbf{X}$ has to be priced and hedged as a unit; no "stand-alone" scenario price for X_i can be deduced from $\hat{F}_0$. Sellers of $Y = \lambda \cdot \mathbf{X}$ are hedged against volatility risk within the bounds $\mathcal{C}$ if they charge at least $\hat{F}_0(Y)$. Vice versa, buyers of Y are hedged if they pay at most $-\hat{F}_0(-Y)$. The volatility range $[\sigma_{\min}, \sigma_{\max}]$ thus leads to a corresponding no-arbitrage worst-case price range $[-\hat{F}_0(-Y), \hat{F}_0(Y)]$.

Computationally, nonlinearity requires sophisticated algorithms which handle and (hopefully) reduce the combinatorial complexity that arises if the portfolio under consideration contains exotic, path-dependent options. In the remainder of this thesis, algorithms for barrier and American options are studied in particular.

Part II

Algorithms for Nonlinear Models

5 A Lattice Framework

Nonlinear models that embed Black-Scholes in worst-case scenarios require algorithmic techniques on two levels:

1. Finite difference methods combined with dynamic programming are used to solve individual PDEs of type (4.4.8).

2. A collection of PDEs needs to be solved in the right order if exotic options with barrier or American features are involved. Solutions of subordinate PDEs serve as boundary data for PDEs higher up in the hierarchy. (There is only one PDE if the portfolio under consideration contains only vanilla options.)

The sensitivity of the remaining portfolio to fluctuations in σ changes if options are taken out through knock-out or early exercise. The so altered portfolio, evaluated independently, may yield an instantiation of σ under the worst-case scenario which differs from the one for the original portfolio. Consequently, it may also yield a worst-case value that differs from the contribution of the remaining options to the worst-case value of the original portfolio, had the option(s) not been taken out. The worst-case value of the reduced portfolio, computed separately, must be used as boundary value for the original portfolio where options are removed by knock-out or early exercise.

An example may help to clarify this explanation. Assume a portfolio of two call options X_1 and X_2 which are identical except for the fact X_2 allows early exercise, while X_1 does not. The positions are $\lambda_1 = -1$ and $\lambda_2 = 2$, respectively. Let $Y = \lambda \cdot \mathbf{X}$ be the payoff if X_2 is held until maturity, and let $Y' = \lambda_1 X_1$ be the remaining payoff if X_2 is not held until maturity, but exercised early. Figure 5.1 shows the payoff graphically for both cases.

It is clear that the worst-case volatility is $\sigma = \sigma_{\max}$ if X_2 is held until maturity, for Y and $f_t(S_t, t; Y)$ are both convex in S. (Recall that f is the solution of (4.4.8).) On the other hand, $\sigma = \sigma_{\min}$ from the time on at which X_2 is exercised, for the remainder Y' and thus $f_t(S_t, t; Y')$ are concave in S. In this case, the outlook in terms of exposure to volatility risk is significantly changed. Although the analysis is straightforward in this toy example, the complexity of the problem grows very fast in cases of mixed convexity or exotic options.

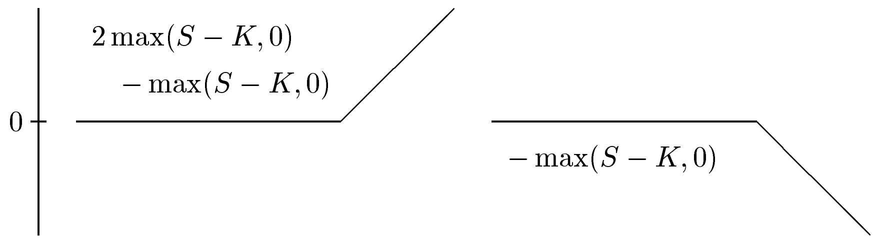


Figure 5.1: The shape of the final payoff $Y = \lambda \cdot \mathbf{X} = 2X_2 - X_1$ on the left side, and $Y' = \lambda_1 X_1 = -X_1$ on the right side

From now on, worst-case pricing—see Sect. 4.2.1—**will be the underlying worst-case scenario.** Results are easily applicable to worst-case hedging and calibration.

The complexity of worst-case pricing and algorithms that cope with it are the focus of the rest of Part II. Chapters 6 and 7 treat in detail the implications arising from the inclusion of barrier and American options into the portfolio. The current chapter focuses on numerical and general data structure aspects of solvers for PDEs of type (4.4.8). As one may need to solve multiple PDE's simultaneously, data structures must support the flow of boundary and decision-support data.

5.1 Multi-lattice Dynamic Programming

The current price of the underlying asset is denoted by $S_0 = s_0$. Let $[s_D, s_U]$ and $[0, T]$ be suitably chosen ranges for the space and time dimensions of the lattice, with $s_D < s_0 < s_U$. Let

$$0 = t_0 < t_1 < \cdots < T_N = T$$

be an equidistant discretization of time, i.e. $t_i = i\,dt$ for $dt = T/N$ and $0 \le i \le N$.

The space dimension need not be uniformly disretized; we will see later that the arbitrary spacing of knock-out barriers requires non-uniformity to avoid slow convergence. Denote the space discretization by

$$s_D = \cdots < s_{-2} < s_{-1} < s_0 < s_1 < s_2 < \cdots = s_U,$$

where for convencience we use the $_D$ and $_U$ subscripts also as numerical index. (In practice, $s_U/s_0 = s_0/s_D \approx 3.5\sqrt{T}$ leads to good results and limits the time complexity in the number of time steps to $O\left(N^{3/2}\right)$. The interested reader is referred to Parás (1995).)

5.1.1 Data Structures

We refer to a lattice node by its space and time labels (s_j, t_i), or simply by its space and time indexes (j, i), whichever is more convenient. All PDEs are based on the same discretization. Each PDE, however, is assigned its own lattice instance in memory. Boundary values are shared by copying (and possibly processing) data from one lattice instance to another.

Each lattice instance L is identified by a partial portfolio $\mathbf{X}_L \subseteq \mathbf{X}$ and a position λ_L (which need not be a partial position of λ). If there are only vanilla options in $\mathbf{X}$, there is only one lattice instance in the computation, identified by top-level $(\mathbf{X}, \lambda)$.

Definition 5.1 (Lattice signature). *Let L be a lattice instance identified by partial portfolio $\mathbf{X}_L \subseteq \mathbf{X}$ and position λ_L. The pair $(\mathbf{X}_L, \lambda_L)$ is called the* signature *of L. The size of L is denoted by $|L| = |\mathbf{X}_L| = |\lambda_L|$.*

Often, λ_L is ommitted, and only $\mathbf{X}_L$ is used to refer to a lattice instance for simplicity. Lattice instances may be added dynamically during the computation. The set of the signatures of active lattice instances is denoted by $\mathcal{L}$. At all times, $(\mathbf{X}, \lambda) \in \mathcal{L}$.

Definition 5.2. *Let $L \in \mathcal{L}$ be a lattice instance with signature $(\mathbf{X}_L, \lambda_L)$, and (j, i) a node. $\hat{V}(j, i; L)$ denotes the finite difference approximation of the worst-case value $\hat{F}_i(\lambda_L \cdot \mathbf{X}_L \mid S_i = s_j)$, and $\hat{v}_k(j, i; L)$ denotes its partial derivative in $(\lambda_L)_k$, $1 \le k \le |L|$:*

$$\begin{aligned} \hat{V}(j,i;L) &= \hat{F}_i(\lambda_L \cdot \mathbf{X}_L \mid S_i = s_j) \\ \hat{v}_k(j,i;L) &= \frac{\partial}{\partial(\lambda_L)_k} \hat{F}_i(\lambda_L \cdot \mathbf{X}_L \mid S_i = s_j) \end{aligned} \tag{5.5.1}$$

(Here and in the following, i and t_i are used interchangingly to index processes such as $\hat{F}$.)

With each node instance $(j, i; L)$ is therefore associated a value/gradient pair

$$\left[\hat{V}(j,i;L), \quad (\hat{v}_k(j,i;L) \mid 1 \le k \le |L|)\right]$$

that is stored in the lattice instance's private memory. If is clear which lattice instance L is meant, or if L is not significant, L is omitted.

Not all value/gradient pairs need to be accessible at the same time. Two general rules must to be observed, however:

Internal consistency: For the finite difference scheme to work, time $i+1$ value/gradient node instances need to be available when time i node instances are computed.

External consistency: A node instance $(j, i; L')$ needs to be available if the computation of node instance $(j, i; L)$, $\mathbf{X}_{L'} \subseteq \mathbf{X}_L$, requires the lookup of a boundary value associated with partial portfolio/position $(\mathbf{X}_{L'}, \lambda_{L'})$.

The second rule motivates the general policy, possibly augmented for special cases, to process existing lattice instance L before lattice instance L' if $|L| < |L'|$. Furthermore, a mechanism must be implemented which automatically inserts a new lattice instance with the appropriate signature into $\mathcal{L}$ if the second rule is violated nevertheless (exception handling).

5.1.2 Dataflow for Explicit Methods

Definition 5.3. *We say that the node instance $(j, i; L)$ belongs to the* continuation region *if no L'-lookup is necessary to determine the worst-case value for it, for any lattice instance $L' \neq L$.*

Figure 5.2 shows the dataflow for an explicit forward Euler one-level scheme for a PDE of type (4.4.8) within the continuation region.

If a node instance $(j, i; L)$ turns out to require boundary data from $L' \neq L$, the scheme in Fig. 5.2 may or may not be bypassed, depending on whether $\hat{V}(j, i; L)$ can be determined unconditionally (knock-out) or not (agent's choice like early exercise).

Notice that data flows from time $i + 1$ to time i slices for both instantiation of the uncertain coefficient and actual rollback.

5.1.3 Dataflow for Mixed Explicit/Implicit Methods

Mixed explicit/implicit methods such as Crank-Nicholson introduce a lag of one time slice between the instantiation of the uncertain coefficient and the actual rollback, as shown pictorially in Fig. 5.3.

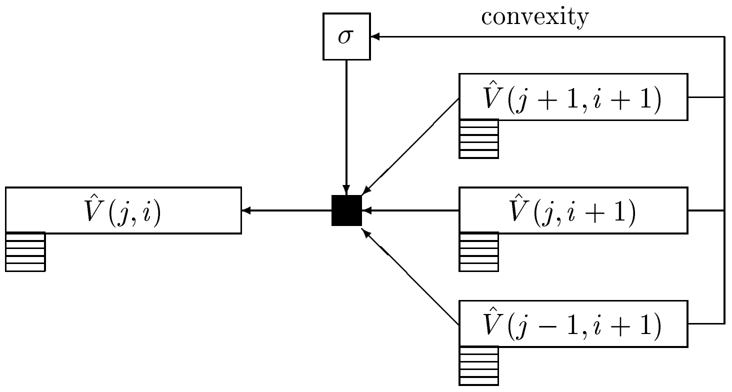


Figure 5.2: Dataflow for explicit one-level finite differencing in the continuation region. Values at time $i+1$ nodes are first used to compute the worst-case volatility. The black box signifies the finite difference approximation for the PDE. The compartmentalized node attachments symbolize the gradient $(\hat{v}_k(\cdot,\cdot))_k$

This discrepancy is necessary to preserve the simplicity of the tridiagonal system of linear equations that obtains in the rollback step from time $i+1$ to time i. The nonlinearity introduced by the worst-case scenario is taken care of entirely in the explicit instantiation of σ.

Mixed methods cause more problems if the transfer of boundary values between lattice instances is not unconditional. Iterative refinement methods such as SOR must then be employed since the replacement of one $\hat{V}(j,i;L)$ affects all other $\hat{V}(\cdot,i;L)$'s, through their implicit connection.

5.2 Numerical Issues

Standard procedure is to solve PDEs of the Black-Scholes type on a lattice whose space dimension is discretized uniformly after logarithmic scaling. It is also well-known to practitioners that barriers should coincide with spatial levels of the lattice whenever possible. Since a) the number of distinct barriers in the portfolio is not limited, b) all instruments and thus all barriers must be watched simultaneously under worst-case

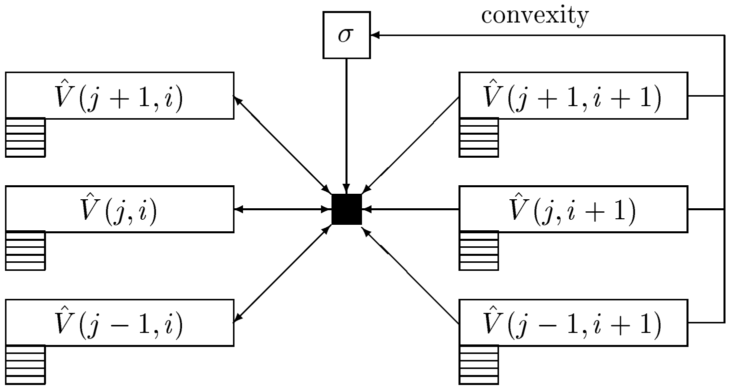


Figure 5.3: Dataflow for mixed explicit/implicit one-level finite differencing in the continuation region. Values at time $i+1$ nodes are first used in an explicit fashion to compute the worst-case volatility at all space levels. The black box represents one equation in the linear system of equations, instantiated with the local worst-case volatility. The bi-directional arrows on the left side indicate the implicit nature of the system

scenarios, and c) uniform spacing can match at most one barrier and s_0, or two barriers at the same time (Rubinstein and Reiner (1991) and Cheuk and Vorst (1996)), it is reasonable to modify the standard procedure to allow non-uniformity.

Secondly, to guarantee stability, explicit forward Euler schemes require the von Neumann condition $dt/(\Delta x)^2 \leq 1/2$ to hold. Here, Δx is the spatial step size after a suitable variable transformation. Equivalently, one may require the transition weights assigned to the arrows in Fig. 5.2 to remain positive (see Thomas (1995) or Wilmott *et al.* (1993)).

We present an algorithm that matches all barriers except those that are *very* close together, retains uniform spacing between barriers, and obyes the von Neumann stability condition (this condition is lifted for Crank-Nicholson, since it is not necessary for mixed epxlicit/implicit methods).

The following exposition is taken from Avellaneda and Buff (1998).

Let the factors $U_j = s_{j+1}/s_j$ resp. $D_j = s_{j-1}/s_j$ represent the size of the up resp. down moves at each spatial level. Instead of using the increments U_j and D_j directly, however, we switch to their logarithms and work with quantities $\overline{\sigma}_U^j$ and $\overline{\sigma}_D^j$ satisfying

$$\begin{aligned} U_j &= 1/D_{j+1} = \mathrm{e}^{\overline{\sigma}_U^j \sqrt{dt}} \\ D_j &= 1/U_{j-1} = \mathrm{e}^{-\overline{\sigma}_D^j \sqrt{dt}} \end{aligned} \tag{5.5.2}$$

for $D \leq j \leq U$. dt is the time increment determined from an initial target increment $dt_{\max}$.

Equation (4.4.8) is formulated with the riskneutral drift r_t. We generalize and write $\mu_t = r_t - d_r$ instead, where d_t denotes a dividend rate, foreign interest rate or storage cost, depending on the properties of the underlying asset. It is assumed that lower and upper bounds

$$\mu_{\min} \leq \mu_t \leq \mu_{\max} \qquad (0 \leq t \leq T) \tag{5.5.3}$$

are known.

For simplicity, we assume that there are n up-and-out barriers

$$s_0 < b_1 < b_2 < \cdots < b_n < \infty,$$

and no down-and-out barriers. Extension to down-and-out barriers in both algorithms and proposition is straightforward. By convention, $s_0 = b_0$ is also treated as a barrier.

Proposition 5.4. *Given barriers $b_0, \ldots, b_n$, a target time step $dt_{\max}$, volatility bounds $\sigma_{\min}$, $\sigma_{\max}$, and drift bounds $\mu_{\min}$, $\mu_{\max}$. If the algorithm in Fig, 5.4 is used to compute spatial increments $\overline{\sigma}_U^j$, $\overline{\sigma}_D^j$ for $D \leq j \leq U$ together with a possible adjustment of $dt_{\max}$ to dt, then the explicit forward Euler approximation of* (4.4.8) *shown in Fig. 5.5 obeys the von Neumann stability condition. In particular, the variables P_U and P_D satisfy*

$$\begin{aligned} P_U, P_D &> 0 \\ P_U + P_D &< \frac{1}{2} \end{aligned} \tag{5.5.4}$$

Furthermore, the barriers $b_0, b_1, \ldots, b_n$ are all matched if the algorithm in Fig. 5.5 does not stop with an error.

Proof. See Avellaneda and Buff (1998) for a full proof. Here, let us only apply the transformation $X = \log(S)$ to (4.4.8) to get

$$\frac{\partial f}{\partial t} + \frac{1}{2}\Sigma^2\left\{\mathrm{e}^{-2X}\left(\frac{\partial^2 f}{\partial X^2} - \frac{\partial f}{\partial X}\right)\right\}\left(\frac{\partial^2 f}{\partial X^2} - \frac{\partial f}{\partial X}\right) + \mu_t \frac{\partial f}{\partial X} - r_t f_t = 0 \tag{5.5.5}$$

```
Input: barriers $b_0, \dots, b_n$, $dt_{\max}$, $\sigma_{\min}$, $\sigma_{\max}$, $\mu_{\min}$, $\mu_{\max}$
Output: $dt$, $\overline{\sigma}_U^j$, $\overline{\sigma}_D^j$ for $0 \leq j \leq U$
(extension to cover down-and-out barriers as well is straightforward)

1. Set $\overline{\mu} := \max\{|\mu_{\min}|, |\mu_{\max}|\}$
2. Set $dt := dt_{\max}$. This is the initial guess, to be adjusted later
3. Repeat for $i = 0, \dots, n$:
   (a) Set $\overline{\sigma} := 2\sigma_{\max}$ (see remark in text)
   (b) If $i = n$ then skip the next step (there are no more barriers above $b_n$)
   (c) Increase $\overline{\sigma}$ such that $b_i e^{k\overline{\sigma}\sqrt{dt}} = b_{i+1}$ for some $k \in \mathbb{N}$. If no such $k$ exists (i.e.,
       $\ln(b_{j+1}/b_j) < \overline{\sigma}\sqrt{dt}$), abort and report an error (see remark in text)
   (d) Set
           $$dt' := \left[\frac{2\sigma_{\min}^2}{\overline{\sigma}(\sigma_{\max}^2 + 2\overline{\mu})}\right]^2$$
       Check if $dt < dt'$. If yes, skip the next step ($dt$ has passed the test)
   (e) $dt$ is too big: choose a new $dt > 0$ such that $dt < dt'$ and start over with step 3
       (for instance, set $dt = 0.9dt'$)
   (f) For all $s_j$ such that $b_i < s_j < b_{i+1}$ (or simply $b_i < s_j$ if $i = n$), set $\overline{\sigma}_U^j := \overline{\sigma}_D^j :=$
       $\overline{\sigma}$. In addition, set $\overline{\sigma}_U^{j_0} := \overline{\sigma}$ where $s_{j_0} = b_i$, and if $i < n$, set $\overline{\sigma}_D^{j_1} := \overline{\sigma}$ where
       $s_{j_1} = b_{i+1}$
```

Figure 5.4: Discretizing space while preserving the von Neumann condition. The input $dt_{\max}$ indicates the desirable time step, from which spatial increments $\overline{\sigma}_U^j$ and $\overline{\sigma}_D^j$ are derived. The output dt is equal to $dt_{\max}$ if no adjustments are necessary, smaller otherwise (see step 3e). The algorithm matches one barrier at a time, starting with $b_0 = s_0$

Input: Lattice instance L, time $i\ dt$, $\overline{\sigma}_U^j$, $\overline{\sigma}_D^j$ for $D \le j \le U$

Output: $\hat{V}(j,i;L)$ for $D \le j \le U$

1. Repeat for $D < j < U$:

   (a) Define

$$P_U(\sigma) = \frac{\sigma^2}{\overline{\sigma}_U^j \overline{\sigma}_D^j + \left(\overline{\sigma}_U^j\right)^2} \left(1 - \frac{\overline{\sigma}_D^j \sqrt{dt}}{2}\right) + \frac{\mu\, \overline{\sigma}_D^j \sqrt{dt}}{\overline{\sigma}_U^j \overline{\sigma}_D^j + \left(\overline{\sigma}_U^j\right)^2}$$

$$P_D(\sigma) = \frac{\sigma^2}{\overline{\sigma}_U^j \overline{\sigma}_D^j + \left(\overline{\sigma}_D^j\right)^2} \left(1 + \frac{\overline{\sigma}_U^j \sqrt{dt}}{2}\right) - \frac{\mu\, \overline{\sigma}_U^j \sqrt{dt}}{\overline{\sigma}_U^j \overline{\sigma}_D^j + \left(\overline{\sigma}_D^j\right)^2}$$

$$P_M(\sigma) = 1 - P_U(\sigma) - P_D(\sigma)$$

   (b) Set

$$\begin{aligned}\hat{V}(j,i;L) :=& \mathrm{e}^{-r_{t_i} dt} \max_\sigma \Big\{ P_U(\sigma)\, \hat{V}(j+1, i+1; L) \\ & + P_M(\sigma)\, \hat{V}(j, i+1, L) + P_D(\sigma)\, \hat{V}(j-1, i+1; L) \Big\}\end{aligned}$$

   where the maximum is taken over $\{\sigma_{\min}, \sigma_{\max}\}$

2. Extrapolate to get $\hat{V}(D,i;L)$ and $\hat{V}(U,i;L)$

Figure 5.5: The explicit forward Euler scheme to compute the worst-case value $\hat{V}(j,i;L)$ at all spatial levels $s_D, \dots, s_U$ from the $\hat{V}(\cdot, i+1; L)$. The gradient is computed similarly. This algorithm corresponds to Fig. 5.2

with

$$\Sigma^2\{C\} = \begin{cases} \sigma^2_{\max} & \text{if } C \geq 0 \\ \sigma^2_{\min} & \text{if } C < 0 \end{cases} \tag{5.5.6}$$

The explicit finite difference approximations for (5.5.5) are as follows. For the time axis, the forward difference

$$\frac{\partial f}{\partial t} \doteq \frac{\hat{V}(j,i+1) - \hat{V}(j,i)}{dt} \tag{5.5.7}$$

is used. On the space axis, centered differences for both the first and second partial derivatives are used. Since the upward and downward displacement might differ, the formulas are slightly more complex than usual:

$$\begin{aligned} \frac{\partial f}{\partial X} &\doteq \frac{1}{\alpha\sqrt{dt}}\Big[\left(\overline{\sigma}^j_D\right)^2 \hat{V}(j+1,i+1) \\ &\quad - \left(\overline{\sigma}^j_U\right)^2 \hat{V}(j-1,i+1) - \left(\left(\overline{\sigma}^j_D\right)^2 - \left(\overline{\sigma}^j_U\right)^2\right)\hat{V}(j,i+1)\Big] \\ \frac{\partial^2 f}{\partial X^2} &\doteq \frac{2}{\alpha\, dt}\Big[\overline{\sigma}^j_D \hat{V}(j+1,i+1) \\ &\quad + \overline{\sigma}^j_U \hat{V}(j-1,i+1) - (\overline{\sigma}^j_D + \overline{\sigma}^j_U)\hat{V}(j,i+1)\Big] \end{aligned} \tag{5.5.8}$$

where

$$\alpha = \overline{\sigma}^j_U \left(\overline{\sigma}^j_D\right)^2 + \overline{\sigma}^j_D \left(\overline{\sigma}^j_U\right)^2 \tag{5.5.9}$$

Algebra shows that the weights P_U, P_M and P_D computed in the algorithm in Fig. 5.4 replicate the approximation (5.5.7) and (5.5.8). They furthermore satisfy (5.5.4) by construction: crucial is step 3d.

The barriers are matched by construction as well. □

P_U, P_D and $P_M = 1 - P_U - P_D$ can be regarded as probabilities. The property $P_U + P_D < \frac{1}{2}$ guarantees that the middle weight is always at least $\frac{1}{2}$; this has been found empirically to lead to a significant improvement in accuracy (a small P_M effectively turns the explicit scheme into a binomial tree method).

Note that the algorithm in Fig. 5.5 matches the barriers regardless of the validity of the von Neumann condition. The algorithm can thus be used unmodified for mixed explicit/implicit schemes (and indeed is). The algorithm in Fig. 5.4 can be significantly simplified in the mixed case (the test with dt' can be ommitted).

Two further remarks should be made. Firstly, step 3a in the algorithm in Fig. 5.5 can safely be replaced by

3a$'$ Set $\overline{\sigma} := \sqrt{2}\,\sigma_{\max}$

In this case $P_U + P_D < \frac{1}{2}$ only if $\overline{\sigma}^j_U = \overline{\sigma}^j_D$. For $\overline{\sigma}^j_U \neq \overline{\sigma}^j_D$, the upper bound becomes $P_U + P_D < 1$ instead. This still guarantees $P_M > 0$ and therefore does not break the probability framework of the derivation. Moreover, $\overline{\sigma}^j_U \neq \overline{\sigma}^j_D$ for at most n spatial levels of the lattice (n is the number of barriers). The ratio of the number of "good" j's ($\overline{\sigma}^j_U = \overline{\sigma}^j_D$) over the number of "bad" j's ($\overline{\sigma}^j_U \neq \overline{\sigma}^j_D$) is therefore negligible as the granularity of the lattice gets finer.

Secondly, the algorithm may trigger an error in step 3c. If two barriers are too close to each other, one of them must be ignored and the algorithm is restarted with the number of barriers reduced by one.

6 Algorithms for Barrier Options

Consider a portfolio $\mathbf{X}$ consisting of position λ_1 in barrier option X_1 with knock-out barrier $b > s_0$ and positions $\lambda_2, \dots, \lambda_k$ in $k-1$ vanilla options $X_2, \dots, X_k$. Let all options mature at time T. The payoff at time T is path-dependent: depending on whether the underlying asset has reached the barrier b in the time interval $[0, T]$ or not, the owner of the portfolio receives $\sum_{i=2}^{k} \lambda_i X_i$ or $\lambda \cdot \mathbf{X}$, respectively.

The situation is shown pictorially in Fig. 6.1. Path 1 crosses the barrier u at time t, path 2 doesn't. When path 1 hits the barrier, X_1 becomes worthless. As the portfolio is reduced by one instrument, its sensitivity to volatility fluctuations between times t and T is likely to differ from the sensitivity of the original, unaltered portfolio $(\mathbf{X}, \lambda)$. The worst-case volatility from time t on is therefore likely to be different for the partial and the original portfolio. Hence, two instances of the worst-case pricing problem must be solved, for $(\mathbf{X}, \lambda)$ and for $(\mathbf{Y}, \lambda')$, respectively, where $\mathbf{Y} = (X_2, \dots, X_k)^{\mathrm{T}}$ and $\lambda' = (\lambda_2, \dots, \lambda_k)^{\mathrm{T}}$.

Two worst-case pricing problems correspond to two lattice instances L_1 and L_2, each assigned to solve a PDE of type (4.4.8). The boundary conditions imposed on the two PDE's, however, differ. L_2 is used to solve an initial-value problem with initial value

$$\hat{f}(S_T, T; \lambda', \mathbf{Y}) = \lambda' \cdot \mathbf{Y}(S_T) \tag{6.6.1}$$

as the partial portfolio $(\mathbf{Y}, \lambda')$ contains only vanilla options. L_1 is used to solve an initial-boundary-value problem with initial value

$$\hat{f}(S_T, T; \lambda, \mathbf{X}) = \lambda \cdot \mathbf{X}(S_T) \tag{6.6.2}$$

and boundary value

$$\hat{f}(u, t; \lambda, \mathbf{X}) = \hat{f}(u, t; \lambda', \mathbf{Y}) \tag{6.6.3}$$

for $0 \le t \le T$. Under the assumption that L_1 and L_2 match the barrier u at level j_u, (6.6.3) is reflected within the finite difference framework by the identity of values

$$\hat{V}(j_u, i; L_1) = \hat{V}(j_u, i; L_2) + 0 \tag{6.6.4}$$

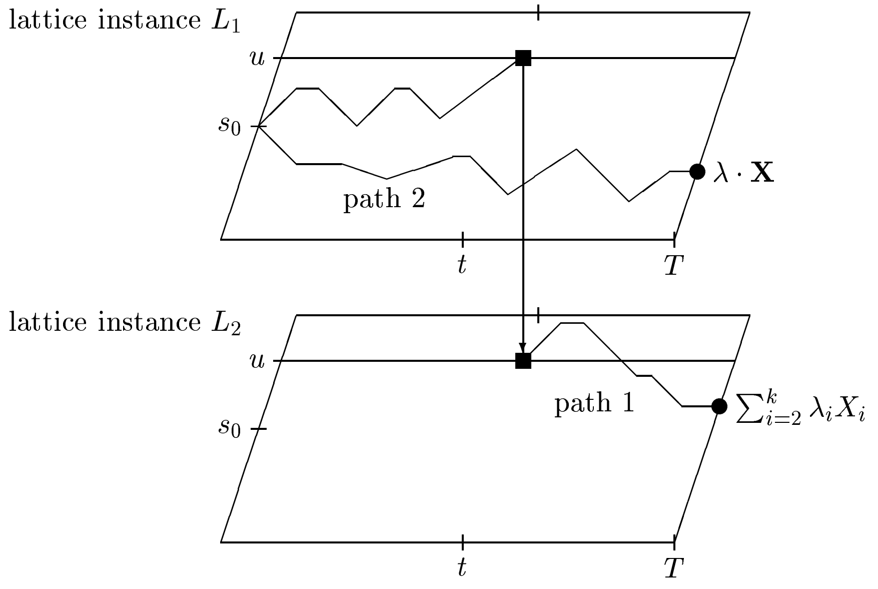


Figure 6.1: Two paths taken by the underlying asset. Path 1 crosses the barrier u and looses the position in X_1 at time t, consequently shifting to lattice instance L_2 whose signature does not contain X_1. Path 2 stays below the barrier and leaves the portfolio intact until expiration

and the identity of gradients

$$\begin{aligned} \hat{v}_1(j_u, i; L_1) &= 0 \\ \hat{v}_2(j_u, i; L_1) &= \hat{v}_1(j_u, i; L_2) \\ &\cdots \\ \hat{v}_k(j_u, i; L_1) &= \hat{v}_{k-1}(j_u, i; L_2) \end{aligned} \tag{6.6.5}$$

for all time slices $0 \leq i \leq N$. We write "$+\,0$" in (6.6.4) to indicate that any potential premium received as a result of X_1's knock-out must be added to the residual worst-case liability $\hat{V}(j_u, i; L_2)$. The identities (6.6.4) and (6.6.5) replace the transactions shown in Figs. 5.2 and 5.3. External consistency requires furthermore that nodes of L_2 are always processed before corresponding nodes of L_1.

A remark regarding path-dependency The candidate set $\mathcal{C}$ defined in Def. 4.3 contains only non-path-dependent elements $\sigma\colon (\mathbb{R}_{++} \times [0, T]) \to \mathbb{R}_{++}$. To solve separate pricing problems on distinct lattice instances and transfer boundary data essentially makes the volatility path-dependent and thus leads to a worst-case volatility scenario that may not be part of $\mathcal{C}$. Without proof, however, we point out that the worst-case volatility is path-independent (i.e., recombining on a discrete lattice) for paths that remain within a single lattice instance. Only where paths hit boundaries and a jump between lattice instances occurs may volatilities diverge. Each realized path can experience only a finite number of such jumps.

We refrain from changing Def. 4.3 formally, but ask the reader to keep this remark in mind. At any rate, the subsequent definitions that contain terms such as $\sup_{\sigma \in \mathcal{C}}(\mathrm{E}_{Q(\sigma)}(\dots))$ remain consistent under either interpretation of $\mathcal{C}$, since jumps are always explicitely reflected in recursive boundary terms or terms that contain independent σ'.

This remark applies to both worst-case pricing of barrier options (this chapter) and American options (treated in Chapter 7).

6.1 The Hierarchy of PDEs

We have seen that two lattice instances have to be created if the portfolio contains one barrier option, thus doubling the cost of solving the worst-case pricing problem. The immediate question is: what is the number of lattice instances in the general case, and what are their signatures? How expensive is it to compute worst-case values for portfolios that contain more than one barrier option?

We answer this question for any portfolio that contains up-and-out, down-and-out and double-barrier knock-out options. Up-and-out barrier options knock out if the asset price reaches a barrier $u > s_0$, as in the example in Fig. 6.1. Down-and-out barrier options knock out if the asset prices falls to a level $d < s_0$. Double-barrier options knock out as soon as the asset reaches a barrier $u > s_0$, *or* falls to a level $d < s_0$: the interval $[d, u]$ defines a corridor in which the double-barrier option is alive.

The following sections closely follow Avellaneda and Buff (1998).

6.1.1 Construction

Let each instrument of the portfolio $\mathbf{X}$ be associated with an up-and-out barrier $u(X_i)$ and a down-and-out barrier $d(X_i)$, $1 \leq i \leq k$. Vanilla options are modeled by setting $d(X_i) = 0$ and $u(X_i)$ to a very large number (preferrably larger than s_U, the upper boundary of the finite difference lattice). For a single up-and-out barrier option with barrier b, $d(X_i) = 0$ and $u(X_i) = b$. For a single down-and-out barrier option with barrier b, $d(X_i) = b$ and $u(X_i)$ very large. For double barrier options, $d(X_i)$ and $u(X_i)$ are both set to the respective barriers.

The open asset-price interval in which the instrument X_i is possibly alive is denoted by $a(X_i) = (d(X_i), u(X_i))$, $1 \leq i \leq k$. Let $\mathbf{Y} \subseteq \mathbf{X}$ be a partial portfolio with $k' \leq k$ instruments . Define

$$A(\mathbf{Y}) = \bigcap_{i=1}^{k'} a(Y_i) \tag{6.6.6}$$

$A(\mathbf{Y})$ is also open. Let

$$\begin{aligned} U(\mathbf{Y}) &= \sup A(\mathbf{Y}) = \min_{i=1}^{k'} u(Y_i) \\ D(\mathbf{Y}) &= \inf A(\mathbf{Y}) = \max_{i=1}^{k'} d(Y_i) \end{aligned} \tag{6.6.7}$$

$[D(\mathbf{Y}), U(\mathbf{Y})]$ is the closure of $A(\mathbf{Y})$. $U(\mathbf{Y})$ is the smallest up-and-out barrier of the instruments in $\mathbf{Y}$. Similarly, $D(\mathbf{Y})$ is the largest down-and-out-barrier in $\mathbf{Y}$. If the underlying asset stays within $A(\mathbf{Y})$, an initial position in $\mathbf{Y}$ will remain intact until expiration.

Definition 6.1 (Extensions). *Let $\mathbf{Y} \subseteq \mathbf{X}$ be a partial portfolio with $k' \leq k$ instruments.*

$$B_U(\mathbf{Y}) = \left\{1 \leq i \leq k' \mid u(Y_i) > U(\mathbf{Y})\right\} \tag{6.6.8}$$

is called the upper extension *of $\mathbf{Y}$. Correspondingly,*

$$B_D(\mathbf{Y}) = \left\{1 \leq i \leq k' \mid d(Y_i) < D(\mathbf{Y})\right\} \tag{6.6.9}$$

is called the lower extension *of $\mathbf{Y}$. The vectorized versions of B_U and B_D are*

$$\begin{aligned} \mathbf{B}_U(\mathbf{Y}) &= \operatorname{select}(\mathbf{Y}, B_U(\mathbf{Y})) \\ \mathbf{B}_D(\mathbf{Y}) &= \operatorname{select}(\mathbf{Y}, B_D(\mathbf{Y})) \end{aligned} \tag{6.6.10}$$

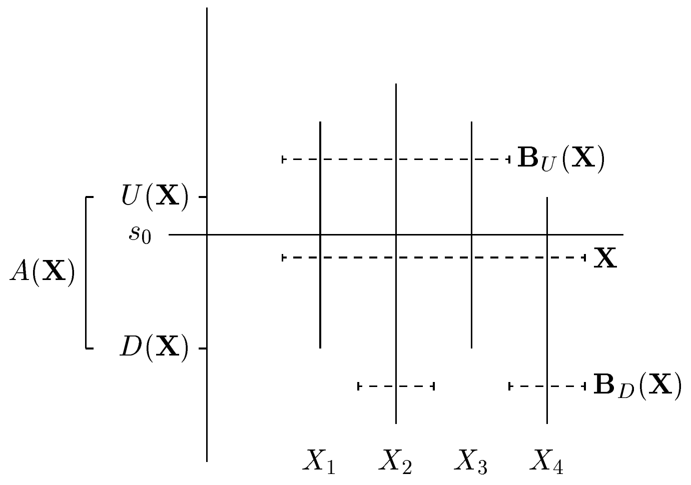


Figure 6.2: A portfolio $\mathbf{X}$ consisting of $k = 4$ options and its upper and lower extensions, $\mathbf{B}_U(\mathbf{X})$ and $\mathbf{B}_D(\mathbf{X})$. The vertical axis marks the price of the underlying asset. $U(\mathbf{X})$ is the smallest up-and-out barrier among the up-and-out barriers in $\mathbf{X}$. Similarly, $D(\mathbf{X})$ is the smallest down-and-out barrier among the down-and-out barriers in $\mathbf{X}$. Each option X_i is represented by a vertical bar whose endpoints are its barriers $u(X_i)$ and $d(X_i)$

Similarly, a position λ' in $\mathbf{Y}$ is reduced to

$$\begin{aligned}\lambda'_U(\mathbf{Y}) &= \text{select}\left(\lambda', B_U(\mathbf{Y})\right)\\ \lambda'_D(\mathbf{Y}) &= \text{select}\left(\lambda', B_D(\mathbf{Y})\right)\end{aligned} \tag{6.6.11}$$

$B_U(\mathbf{Y})$ resp. $B_D(\mathbf{Y})$ indicate which instruments in $\mathbf{Y}$ remain possibly alive when the price of the underlying asset crosses $U(\mathbf{Y})$ resp. $D(\mathbf{Y})$. $B_U(\mathbf{Y})$ and $B_D(\mathbf{Y})$ are sets; the corresponding partial portfolios $\mathbf{B}_U(\mathbf{Y})$ and $\mathbf{B}_D(\mathbf{Y})$ are possibly empty. $(\mathbf{B}_U(\mathbf{Y}), \lambda'_U(\mathbf{Y}))$ and $(\mathbf{B}_D(\mathbf{Y}), \lambda'_D(\mathbf{Y}))$ are the signatures of the lattice instances that feed the boundary data at $U(\mathbf{Y})$ and $D(\mathbf{Y})$. For empty $\mathbf{B}_U(\mathbf{Y})$ or $\mathbf{B}_D(\mathbf{Y})$, no lookup is necessary.

If $U(\mathbf{Y})$ is very large (as is the case if $\mathbf{Y}$ consists of vanilla options only), it will lie outside the finite lattice. Similarly, $D(\mathbf{Y}) = 0$ also falls outside the lattice. In these cases, no additional lattice instances need to be maintained.

Figure 6.2 gives an example for $k = 4$. The sequences of up-and-out and down-and-out barriers are

$$\begin{aligned} &s_0 < u(X_4) < u(X_1) = u(X_3) < u(X_2) \\ &s_0 > d(X_1) = d(X_3) > d(X_2) = d(X_4) \end{aligned} \tag{6.6.12}$$

The upper and lower extensions $\mathbf{B}_U(\mathbf{X})$ and $\mathbf{B}_D(\mathbf{X})$ of the full portfolio contain themselves barrier options. Thus, additional lattice instances covering the extensions of $\mathbf{B}_U(\mathbf{X})$ and $\mathbf{B}_D(\mathbf{X})$ need to be created as well. Recursion leads to the four partial portfolios shown in Figure 6.3. Altogether, four lattice instances are needed to solve the worst-case pricing problem for the example portfolio.

Partial portfolio	Upper extension	Lower extension
$(X_1, X_2, X_3, X_4)^{\mathrm{T}}$	$(X_1, X_2, X_3)^{\mathrm{T}}$	$(X_2, X_4)^{\mathrm{T}}$
$(X_1, X_2, X_3)^{\mathrm{T}}$	$(X_2)^{\mathrm{T}}$	$(X_2)^{\mathrm{T}}$
$(X_2, X_4)^{\mathrm{T}}$	$(X_2)^{\mathrm{T}}$	empty
(X_2)	empty	empty

Figure 6.3: The extension hierarchy created by the example portfolio $\mathbf{X}$ of Fig. 6.2

Definition 6.2 (Extension hierarchy). *Let* $\mathbf{X}$ *be a portfolio with* $k > 0$ *instruments. Let* $\mathcal{B}$ *denote the set of all partial portfolios of* $\mathbf{X}$. *The* extension hierarchy *of* $\mathbf{X}$, *written* $\mathcal{B}(\mathbf{X})$, *is defined as the smallest subset of* $\mathcal{B}$ *such that*

- $\mathbf{X} \in \mathcal{B}(\mathbf{X})$, *and*
- $\mathbf{Y} \in \mathcal{B}(\mathbf{X})$ *implies* $\mathbf{B}_U(\mathbf{Y}) \in \mathcal{B}(\mathbf{X})$ *and* $\mathbf{B}_D(\mathbf{Y}) \in \mathcal{B}(\mathbf{X})$, *assuming those are nonempty*

Figure 6.4 sketches the algorithm to find the extension hierarchy $\mathcal{B}(\mathbf{X})$ of any given portfolio $\mathbf{X}$ on a very high level. The sketch is inefficient, but finding $\mathcal{B}(\mathbf{X})$ is the least costly operation in solving the worst-case for $\mathbf{X}$. (In our actual implementation, we do employ a more efficient procedure.)

Once $\mathcal{B}(\mathbf{X})$ is known, lattice instances can be created, with appropriately instantiated signatures.

$\mathcal{B}(\mathbf{X})$ is exhaustive. No more lattice instances are required to solve the worst-case pricing problem for $\mathbf{X}$. The solution itself is obtained by solving worst-case pricing

```
Input: portfolio $\mathbf{X}$
Output: extension hierarchy $\mathcal{B}(\mathbf{X})$

1. Set $\mathcal{B}(\mathbf{X}) := \{\mathbf{X}\}$
2. Repeat the following:
    (a) Set $\mathcal{B}' := \bigcup_{\mathbf{Y} \in \mathcal{B}(\mathbf{X})} \{\mathbf{B}_D(\mathbf{Y}), \mathbf{B}_U(\mathbf{Y})\}$
    (b) Set $\mathcal{B}' := \mathcal{B}' \setminus (\mathcal{B}(\mathbf{X}) \cup \{\emptyset\})$
    (c) Set $\mathcal{B}(\mathbf{X}) := \mathcal{B}(\mathbf{X}) \cup \mathcal{B}'$
   until $\mathcal{B}' = \emptyset$
```

Figure 6.4: Finding the extension hierarchy $\mathcal{B}(\mathbf{X})$ amounts to computing a closure. In step 2a, we make sure we know all extensions immediately reachable from the current configuration. In step 2b, extensions that are already known are discarded, as well as the empty extension for which no lattice instance is created

problems on all lattice instances, transferring boundary data where necessary. The policy outlined in Sect. 5.1 to ensure external consistency leads to the approach shown in Fig. 6.5, outlined on a very high level. (In a concrete implementation, step 3c is done time slice by time slice; the inner loop implicit in step 3c and the outer loop in step 3 change places.)

6.1.2 Complexity

The example in Fig. 6.2 requires four lattice instances for the solution of the worst-case problem. Now consider a second portfolio $\mathbf{X}'$ also consisting of 4 double-barrier options, with the barriers rearranged as shown in Fig. 6.6. In this case, the application of the algorithm in Fig. 6.4 yields an extension hierarchy of 10 elements, listed in Figure 6.7.

It turns out that $10 = 4 \times (4 + 1)/2$ is indeed the maximum size of any extension hierarchy for a portfolio with 4 instruments. This result can be generalized to the following proposition, taken from Avellaneda and Buff (1998).

```
Input: extension hierarchy $\mathcal{B}(\mathbf{X})$

1. Set $n := |\mathcal{B}(\mathbf{X})|$
2. Find an ordering $\mathbf{Y}_{l_1}, \mathbf{Y}_{l_2}, \ldots, \mathbf{Y}_{l_n}$ of $\mathcal{B}(\mathbf{X})$ such that $|\mathbf{Y}_{l_i}| \leq |\mathbf{Y}_{l_j}|$ for $i < j$
3. Repeat for $i = 1, \ldots, n$:
   (a) If $B_U(\mathbf{Y}_{l_i}) \neq \emptyset$ then access the lattice instance for partial portfolio $\mathbf{B}_U(\mathbf{Y}_{l_i})$,
       and use it for the boundary condition at $U(\mathbf{Y}_{l_i})$
   (b) If $B_D(\mathbf{Y}_{l_i}) \neq \emptyset$ then access the lattice instance for partial portfolio $\mathbf{B}_D(\mathbf{Y}_{l_i})$,
       and use it for the boundary condition at $D(\mathbf{Y}_{l_i})$
   (c) Solve (4.4.8) for $(\mathbf{Y}_{l_i}, \lambda_{l_i})$, using the data produced in the previous two steps
```

Figure 6.5: Solving the worst-case pricing problem for $\mathbf{X}$ requires solving subordinate worst-case problems in the right order. The particular ordering in step 2 implies that $\mathbf{Y}_{l_n} = \mathbf{X}$

Proposition 6.3. *Given a portfolio* $\mathbf{X}$ *of* $k \geq 1$ *instruments such that*

$$u(X_1) > u(X_2) > \cdots > u(X_k) \tag{6.6.13}$$

$$d(X_i) \neq d(X_j) \qquad (1 \leq i, j \leq k, i \neq j) \tag{6.6.14}$$

Then $|\mathcal{B}(\mathbf{X})| \leq k(k+1)/2$.

Proof. By induction over k. For $k = 1$, $|\mathcal{B}(\mathbf{X})| = 1$ by inspection and the proposition is true. Thus assume $k > 1$. Define $\mathbf{X}' = (X_1, \ldots, X_{k-1})^{\mathrm{T}}$. $\mathbf{X}'$ is a partial portfolio of $\mathbf{X}$ with $k-1$ instruments and fulfills the premises of the proposition. Figure 6.8 shows an example for $k = 4$.

The idea is to count the lower extensions that must be added to $\mathcal{B}(\mathbf{X}')$ as a consequence of adding X_k to $\mathbf{X}'$. It will turn out that a) all new extensions contain X_k, and b) upper extensions will not cause trouble, thanks to the particular ordering (6.6.13).

Clearly, $\mathbf{X} \in \mathcal{B}(\mathbf{X})$. By assumption (6.6.13), $B_U(\mathbf{X}) = \mathbf{X}'$ which implies $\mathbf{X}' \in \mathcal{B}(\mathbf{X})$, and by transitivity $\mathcal{B}(\mathbf{X}') \subset \mathcal{B}(\mathbf{X})$ (here we refer to the algorithm in Fig. 6.4).

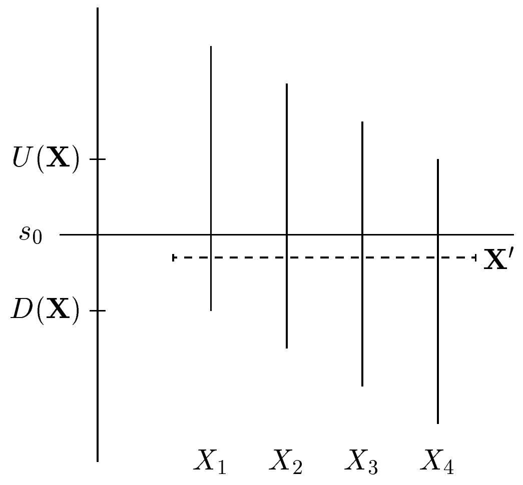


Figure 6.6: A portfolio $\mathbf{X}'$ consisting of $k = 4$ double-barrier options with up-and-out barriers $u(X_1) > u(X_2) > u(X_3) > u(X_4)$ and down-and-out barriers $d(X_1) > d(X_2) > d(X_3) > d(X_4)$. This particular combination requires 10 lattice instances

Now consider the sequence $\mathbf{Y}_0 = \mathbf{X}$, $\mathbf{Y}_1 = \mathbf{B}_D(\mathbf{Y}_0)$, $\mathbf{Y}_2 = \mathbf{B}_D(\mathbf{Y}_1)$, $\dots$, $\mathbf{Y}_{k-1} = \mathbf{B}_D(\mathbf{Y}_{k-2})$, $\mathbf{Y}_k = \emptyset$. This sequence of consecutive lower extensions has $k+1$ distinct elements, because by assumption (6.6.14) the d_i's are all distinct.

Thus, $|\mathbf{Y}_i| = k - i$. For $0 \le i \le k-1$, define

$$\mathbf{B}_i = \text{select}\left(\mathbf{Y}_i, \{1 \le j \le k-i \mid Y_{i,j} \ne X_k\}\right) \tag{6.6.15}$$

$\mathbf{B}_i$ is $\mathbf{Y}_i$, possible reduced by the element X_k if it happens to be part of $\mathbf{Y}_i$. Note that by definition of $\mathbf{X}'$,

$$\mathbf{B}_0 = \mathbf{X}' \tag{6.6.16}$$

We claim that for $0 \le i \le k-1$,

$$\mathbf{B}_i \in \mathcal{B}(\mathbf{X}') \tag{6.6.17}$$

To see this choose $i_0 \in \{0, \dots, k-1\}$ such that $\mathbf{B}_i \subset \mathbf{Y}_i$ (i.e., X_k is in $\mathbf{Y}_i$, and $\mathbf{B}_i$ is a strictly partial portfolio of $\mathbf{Y}_i$) for $i \le i_0$ and $\mathbf{B}_i = \mathbf{Y}_i$ (i.e., X_k is not element of $\mathbf{Y}_i$) for $i > i_0$, and note that for $i < i_0$, the equality $\mathbf{B}_D(\mathbf{B}_i) = \mathbf{B}_{i+1}$ holds.

Partial portfolio	Upper extension	Lower extension
$(X'_1, X'_2, X'_3, X'_4)^{\mathrm{T}}$	$(X'_1, X'_2, X'_3)^{\mathrm{T}}$	$(X'_2, X'_3, X'_4)^{\mathrm{T}}$
$(X'_1, X'_2, X'_3)^{\mathrm{T}}$	$(X'_1, X'_2)^{\mathrm{T}}$	$(X'_2, X'_3)^{\mathrm{T}}$
$(X'_2, X'_3, X'_4)^{\mathrm{T}}$	$(X'_2, X'_3)^{\mathrm{T}}$	$(X'_3, X'_4)^{\mathrm{T}}$
$(X'_1, X'_2)^{\mathrm{T}}$	(X'_1)	(X'_2)
$(X'_2, X'_3)^{\mathrm{T}}$	(X'_2)	(X'_3)
$(X'_3, X'_4)^{\mathrm{T}}$	(X'_3)	(X'_4)
(X'_1)	empty	empty
(X'_2)	empty	empty
(X'_3)	empty	empty
(X'_4)	empty	empty

Figure 6.7: The extension hierarchy created by the example portfolio $\mathbf{X}'$ of Fig. 6.6

Together with (6.6.16) as starting point, this implies that $\mathbf{B}_1 \in \mathcal{B}(\mathbf{X}')$ and recursively $\mathbf{B}_i \in \mathcal{B}(\mathbf{X}')$ for $0 \leq i \leq i_0$. Thus, (6.6.17) is true for at least $0 \leq i \leq i_0$.

That (6.6.17) is also true for $i_0 + 1 \leq i \leq k - 1$ can be derived from $\mathbf{B}_{i_0} = \mathbf{B}_{i_0+1}$ and $\mathbf{B}_i = \mathbf{Y}_i$ for $i > i_0$, both true by choice of i_0, and since $\mathbf{Y}_i \in \mathcal{B}(\mathbf{X}')$ by definition of $\mathbf{Y}_i$.

$u(X_k)$ is the smallest up-and-out barrier, and X_k is thus the first instrument which is dropped. Therefore, $\mathbf{B}_U(\mathbf{Y}_i) = \mathbf{B}_i$, or $\mathbf{B}_U(\mathbf{Y}_i) \in \mathcal{B}(\mathbf{X}')$ for $0 \leq i \leq k-1$ by (6.6.17). This implies that the partial portfolios that are not already part of $\mathcal{B}(\mathbf{X}')$ are exactly those that contain X_k, namely $\mathbf{X}, \mathbf{Y}_1, \ldots, \mathbf{Y}_{i_0}$. Thus, since $i_0 \leq k-1$, it follows that the size of $\mathcal{B}(\mathbf{X})$ is bounded by

$$|\mathcal{B}(\mathbf{X})| \leq |\mathcal{B}(\mathbf{X}')| + 1 + (k-1) \tag{6.6.18}$$

or, by induction,

$$|\mathcal{B}(\mathbf{X})| \leq k(k-1)/2 + 1 + (k-1) = k(k+1)/2 \tag{6.6.19}$$

which completes the proof. □

It is easy to see that the size of the extension hierarchy does not become larger if $u(X_i)$ and $d(X_i)$ are not distinct as postulated in the proposition, making the upper

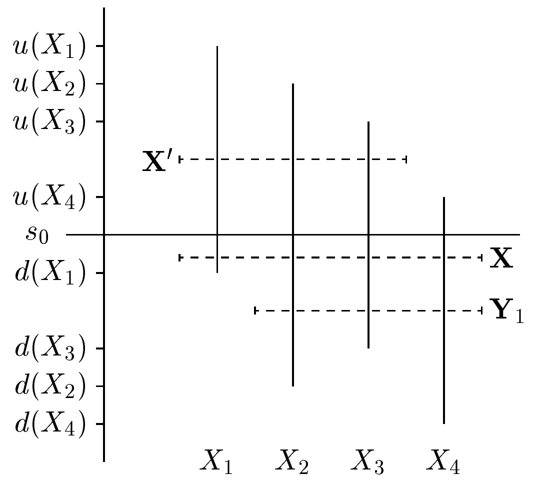


Figure 6.8: Portfolios $\mathbf{X}$ and $\mathbf{X}'$ illustrate the proof of Prop. 6.3 for $k = 4$. $\mathbf{Y}_1$, the lower extension of $\mathbf{X}$, is also marked. Note that $\mathbf{B}_U(\mathbf{Y}_1) = \mathbf{B}_D(\mathbf{X}') = (X_2, X_3)^{\mathrm{T}}$

bound $k(k+1)/2$ the general worst case upper bound for every portfolio of vanilla, single and double barrier options of size k. Motivated by the example in Fig. 6.6, it can be shown that this upper bound is tight.

Corollary 6.4. *Let $\mathbf{X}$ be a portfolio of k double barrier options with barriers $u(X_1) > u(X_2) > \cdots > u(X_k)$ and $d(X_1) > d(X_2) \cdots > d(X_n)$. Then $|\mathcal{B}(\mathbf{X})| = k(k+1)/2$.*

Proof. All elements in the sequence of lower extensions $\mathbf{Y}_0, \ldots, \mathbf{Y}_k$ in the proof of Prop. 6.3 contain X_k. i.e. $i_0 = k - 1$. Since this is true in each inductive step, equality follows in (6.6.19). □

Most practical cases do not involve double barrier options. Proposition 6.3 can be specialized for portfolios that contain only single barrier options.

Proposition 6.5. *Given a portfolio $\mathbf{X}$ of $k \geq 1$ instruments such that*

$$s_0 > d(X_1) > d(X_2) > \cdots > d(X_{k_d}) \tag{6.6.20}$$

and

$$u(X_{k_d+1}) > \cdots > u(X_k) > s_0, \tag{6.6.21}$$

for some $k_d \in \{0,\dots,k\}$. Furthermore, $u(X_1), u(X_2), \dots, u(X_{k_d})$ are very large, and $d(X_{k_d+1}) = \cdots = d(X_k) = 0$. (Thus, there are k_d down-and-out barrier options and $k_u = k - k_d$ up-and-out barrier options in $\mathbf{X}$.) Then

$$|\mathcal{B}(\mathbf{X})| = k_d + k_u + k_d\,k_u. \tag{6.6.22}$$

Proof. A simple counting argument will do. If a path $\{S_t(\omega)\}$, $\omega \in \Omega$ of the underlying crosses barrier $d(X_n)$, $1 \leq n \leq k_d$, it must have crossed barriers $d(X_1), \dots, d(X_{n-1})$ before. Thus, only subsets $X_1, \dots, X_n$ with contiguous indexes can be knocked out at any particular time. Therefore, we count $k_d + 1$ ways to separate the k_d down-and-out barrier options into knocked-out ones and ones which are still alive.

Similarly, we count $k_u + 1$ ways to divide the k_u up-and-out barrier options in $\mathbf{X}$ into knocked-out and alive ones. Since the up-and-out and down-and-out barrier options are unrelated, there are $(k_d + 1)(k_u + 1)$ combinations altogether. Disregarding the empty combination, we get

$$|\mathcal{B}(\mathbf{X})| = (k_d + 1)(k_u + 1) - 1 = k_d + k_u + k_d\,k_u. \tag{6.6.23}$$

□

Proposition 6.5 shows that the number of lattice instances for a portfolio of single barrier options is linear both in the number of up-and-out resp. down-and-out barrier options. Again, barriers are not necessarily distinct as required in the premise of the proposition. If, however, k_d and k_u are set to the number of distinct up-and-out and down-and-out barriers in $\mathbf{X}$, respectively, then (6.6.22) remains precise. If $\mathbf{X}$ also contains vanilla options, one additional lattice instance needs to be created, and $|\mathcal{B}(\mathbf{X})| = k_d + k_u + k_d\,k_u + 1$.

6.2 Performance Results

All tests were performed on a Pentium/166 MHz PC running Windows NT Workstation 4.0/SP 3 and equipped with 128 MB of RAM. The worst-case pricer is written in C++ and compiled with Microsoft Visual C++ 5.0, optimizations activated. The pricer uses the algorithms developed in the previous sections, and is implemented in an object oriented fashion of which more will be heard in Part III.

In the following, the term "Mtg" is used to refer to our pricer.

6.2.1 Convergence

Before we measure the speed of Mtg, we need to convince ourselves that the results it delivers are numerically accurate. We first apply Mtg to a double-barrier option and compare the result with two sources in the literature.

Experiment 1: A Double-barrier Option

Set $\mathbf{X} = (X_1)$ and $\lambda_1 = 1$. X_1 is a double barrier option with variable strike K, up-and-out barrier $u = u(X_1)$ and down-and-out barrier $d = d(X_1)$. We assume $S_0 = 2$ and $T = 365$ days. The interest rate r and volatility σ are constant.

Geman and Yor (1996) use a probabilistic approach to price X_1. Kunitomo and Ikeda (1992) suggest a pricing formula that consists of a sum of an infinite series. Mtg is run with four different time steps $dt_{\max} = 1/(N \times 365)$, where $N = 1$, 5, 20 and 50, respectively, as well as under explicit and Crank-Nicholson schemes (in the explicit scheme $dt = dt_{\max}$ after the algorithm in Fig. 5.4 is run).

Figure 6.9 lists the results for three combinations of r, σ, u and d. Geman and Yor's method is quoted as "G-Y", and Kunitomo and Ikeda's method is quoted as "K-I."

The convergence is very satisfactory. For $N = 5$, the first four digits after the decimal point of the results of all three methods agree. The Crank-Nicholson scheme converges slightly faster than the explicit forward Euler scheme. It is, however, between 30 and 50% slower than the explicit scheme.

For $N = 1$, the result appears almost instantaneously. For $N = 50$, 32 and 44 seconds are needed for the explicit and Crank-Nicholson scheme, respectively. The theoretical time complexity is $O(N^{3/2})$, due to trimming. Measurements for all N validate the theory and yield a running time of approximately $0.1 \times N^{3/2}$ seconds for the explicit scheme, and $0.14 \times N^{3/2}$ seconds for Crank-Nicholson.

Experiment 2: A Portfolio of Single-barrier Puts

To test the algorithm in Fig. 5.5, a portfolio of four down-and-out at-the-money puts is priced, listed in Figure 6.10. All options mature in 30 days. The other parameters are $S_0 = 100$, $r = 0.025$ and $\sigma = 0.2$.

The results for the explicit and the Crank-Nicholson scheme are summarized in Fig. 6.11. Also shown are $\overline{\sigma}_D$ and $\overline{\sigma}_U$ (defined in Sect. 5.2). There are four regions

	N (periods per day)	scheme (CR = Crank-Nicholson)	$\sigma = 0.2$ $r = 0.02$ $K = 2$ $d = 1.5, u = 2.5$	$\sigma = 0.5$ $r = 0.05$ $K = 2$ $d = 1.5, u = 3$	$\sigma = 0.5$ $r = 0.05$ $K = 1.75$ $d = 1, u = 3$
Mtg	1	explicit	0.040899	0.017666	0.075914
Mtg	1	CR	0.040968	0.017844	0.076146
Mtg	5	explicit	0.041050	0.017819	0.076102
Mtg	5	CR	0.041063	0.017856	0.076149
Mtg	20	explicit	0.041079	0.017848	0.076158
Mtg	20	CR	0.041083	0.017857	0.07617
Mtg	50	explicit	0.041085	0.017853	0.076168
Mtg	50	CR	0.041086	0.017857	0.076173
G-Y			0.0411	0.0178	0.07615
K-I			0.041089	0.017856	0.076172

Figure 6.9: Prices obtained for a double barrier call option with each of the three methods Mtg, G-Y and K-I. There is no uncertainty

in the lattice with differing $\overline{\sigma}_D$ and $\overline{\sigma}_U$; the three interior barriers at 98, 95 and 90 mark the boundaries between these regions. The range of $\overline{\sigma}_D$ and $\overline{\sigma}_U$ narrows as N becomes large, from a range of 0.29–0.38 with an absolute difference of 0.09 for $N = 1$ to a range of 0.28285–0.028591 with an absolute difference of only 0.000306 for $N = 400$. (It is obvious that smaller and thus more numerous spatial increments have to be bent relatively less to match the barriers.)

A closed form formula for down-and-out barrier puts yields 10.287 as the model value. The numerical result is sufficiently close for $N \geq 100$.

For $N = 50$, 100, 200 and 400 the running time is 2, 6, 17 and 51 seconds under the explicit scheme, and 5, 15, 44 and 139 seconds under Crank-Nicholson. This suggests a running time of $0.006 \times N^{3/2}$ (explicit) respectively $0.0162 \times N^{3/2}$ seconds (Crank-Nicholson). Crank-Nicholson trails the explicit scheme by a factor of ≈ 2.7, while not yielding significant higher accuracy.

Note that this example does not exhibit uncertainty. Only one lattice instance is

type	strike	barrier	position
put	100	98	long 200 contracts
put	100	95	long 10 contracts
put	100	90	long 2 conctracts
put	100	85	long 1 contract

Figure 6.10: A portfolio of four down-and-out 30-day at-the-money puts. The position in each put is approximately inverse proportional to the relative value that it contributes to the portfolio

needed to compute the price of the portfolio, and in this case the individual values of the puts might as well have been computed separately and added up. We introduce uncertainty for this particular portfolio below.

6.2.2 Combinatorics

Experiment 3: Two Barrier Options Hedged

We introduce uncertainty in σ for the first time and choose $\sigma_{\min} = 0.1$ and $\sigma_{\max} = 0.2$ as upper and lower bounds. The other parameters are $S_0 = 100$ and $r = 0.02$.

Let $\mathbf{X}$ be a portfolio of 5 instruments. X_1 is a double-barrier call, X_2 is a single-barrier put, and X_3, X_4 and X_5 are vanilla calls whose market prices are known. All options mature in 30 days. Let $\lambda = (1, 1, 0.24, -0.98, 8.47)^{\mathrm{T}}$ be the position in $\mathbf{X}$.

Four lattice instances are necessary to solve the worst-case problem for $(\mathbf{X}, \lambda)$. Their signatures are, respectively,

- the two barrier options plus the vanillas;
- the double barrier option plus the vanillas;
- the single barrier option plus the vanillas;
- the vanillas.

The worst-case price of $(\mathbf{X}, \lambda)$ is shown in Fig. 6.13 for various time steps. Results under the explicit scheme and Crank-Nicholson are in close agreement, although Crank-

N (periods per day)	price		$\overline{\sigma}_D$, $\overline{\sigma}_U$ between 98 and above	$\overline{\sigma}_D$, $\overline{\sigma}_U$ between 98 and 95	$\overline{\sigma}_D$, $\overline{\sigma}_U$ between 95 and 90	$\overline{\sigma}_D$, $\overline{\sigma}_U$ between 90 and below
	explicit	Crank-Nicholson				
1	7.72429	7.76010	0.38597	0.29699	0.34432	0.36400
2	7.79286	7.80849	0.33426	0.34294	0.29819	0.31524
5	9.96666	10.0005	0.28769	0.33205	0.28872	0.30523
10	10.1146	10.1320	0.30514	0.31306	0.29695	0.28777
20	10.2094	10.2185	0.28769	0.29515	0.28872	0.28727
50	10.2532	10.2569	0.30325	0.30001	0.29216	0.28599
100	10.2709	10.2727	0.29690	0.28285	0.28693	0.28737
200	10.2793	10.2803	0.28729	0.28966	0.28643	0.28599
400	10.2832	10.2837	0.28591	0.28285	0.28300	0.28364

Figure 6.11: Results for a portfolio of four down-and-out at-the-money puts. The model value is 10.287. Also shown are $\overline{\sigma}_D$ and $\overline{\sigma}_U$ for the four significant regions of the lattice, determined by the interior barriers 98, 95 and 90 and found by the algorithm in Fig. 5.5

Nicholson turns out to be between 2 and 4 times slower. The running times for the explicit scheme are less than 1, 1, 3 and 12 seconds for $N = 5$, 10, 20 and 50, respectively. The running times for Crank-Nicholson are 1, 2, 12 and 32 seconds, respectively.

Given their market prices, the transaction involving X_3, X_4 and X_5 creates a premium of 84.499. Thus, anyone charging at least 85.5801 for the entire package and at the same time entering the offsetting position in X_3, X_4 and X_5 (thus effectively charging 1.0811 for X_1 and X_2) will break even or make a profit provided the volatility stays within the band $0.1 \leq \sigma \leq 0.2$ over the next 30 days. It can be shown that our particular offsetting position in the vanillas is optimal in the sense that 1.0811 is the smallest surcharge for X_1 and X_2 for any offsetting position. The position $(0.24, -0.98, 8.47)$ in (X_3, X_4, X_5) is the optimal hedge portfolio for the position $(1, 1)$ in (X_1, X_2). See also Sect. 4.2.2.

type	strike	position	U&O barrier	D&O barrier
call	110	long 1 contract	120	90
put	100	long 1 contract	–	95

type	strike	position	quoted price
call	110	long 0.24 contracts	17% implied vol
call	100	short 0.98 contracts	13% implied vol
call	90	long 8.47 contracts	15% implied vol

Figure 6.12: The portfolio consists of two 30-day barrier options and four 30-day vanilla options. The market prices for the vanilla options are quoted as implied volatility. The contribution of X_3, X_4 and X_5, given their market prices and positions, is 84.499

Experiment 4: Single-barrier Portfolios of Various Sizes

According to Prop. 6.5, the running time is $O(k_d + k_u + k_d\, k_u)$ in the number of down-and-out barriers k_d and up-and-out barriers k_u, assuming there are no double-barrier options in the portfolio. We want to validate this formula experimentally.

We augment the portfolio of four down-and-out barrier puts in Fig. 6.10 by four up-and-out barrier calls as shown in Fig. 6.14. For each combination of down-and-out and up-and-out barrier options $(k_d, k_u) \in \{(x, y) \mid 0 \leq x, y \leq 4, x \geq 1, y \leq x\}$, the worst-case price is computed for the portfolio consisting of the first k_d puts and the first k_u calls. Since we are only interested in the running time, we set $\sigma_{\min} = 0.199999$ and $\sigma_{\max} = 0.2$ just to make sure the problem becomes nonlinear. The other parameters are $S_0 = 100$, $r = 0.025$ and $N = 20$ (i.e., $dt = 1/(20 \times 365)$).

Figure 6.15 shows the result in tabulated form. Figure 6.16 plots the running times against the number of lattice instances required for the combinations of Fig. 6.15. The graphic shows that the running time progresses linearly, thereby validating Prop. 6.5 experimentally. (Slight deviations are expected: each partial portfolio has its unique boundary, and the corresponding lattice instance thus a unique continuation region. Numerical processing concentrates on the continuation region, causing slightly different processing times for different lattice instances.)

N (periods per day)	price explicit	price Crank-Nicholson	explicit price minus premium for X_3, X_4, X_5
5	85.5709	85.5720	1.0719
10	85.5762	85.5771	1.0772
20	85.5788	85.5791	1.0798
50	85.5801	85.5803	1.0811

Figure 6.13: Worst-case prices for a double-barrier option, a single-barrier option and three traded vanillas. The last column shows the contribution of X_1 and X_2 to the worst-case price, given that the market prices for X_3, X_4 and X_5 are 17, 13 and 15% implied volatility, respectively

type	strike	barrier	position
put	100	98	long 200 contracts
put	100	95	long 10 contracts
put	100	90	long 2 conctracts
put	100	85	long 1 contract
call	100	102	long 200 contracts
call	100	105	long 10 contracts
call	100	110	long 2 conctracts
call	100	115	long 1 contract

Figure 6.14: The portfolio of four down-and-out 30-day at-the-money puts of Fig. 6.10, augmented by four up-and-out 30-day at-the-money calls

	k_u (explicit)					k_u (Crank-Nicholson)				
k_d	0	1	2	3	4	0	1	2	3	4
1	0.3	1.3				0.5	1.9			
2	1.0	3.1	5.9			1.6	4.2	8.2		
3	1.8	5.2	9.2	13.8		3.2	7.1	12.7	18.1	
4	2.9	7.8	14.1	19.9	26.7	4.9	10.6	17.7	26.5	37.4

Figure 6.15: The running times in seconds under the explicit resp. Crank-Nicholson schemes for various combinations of down-and-out and up-and-out barrier options (k_d and k_u, respectively)

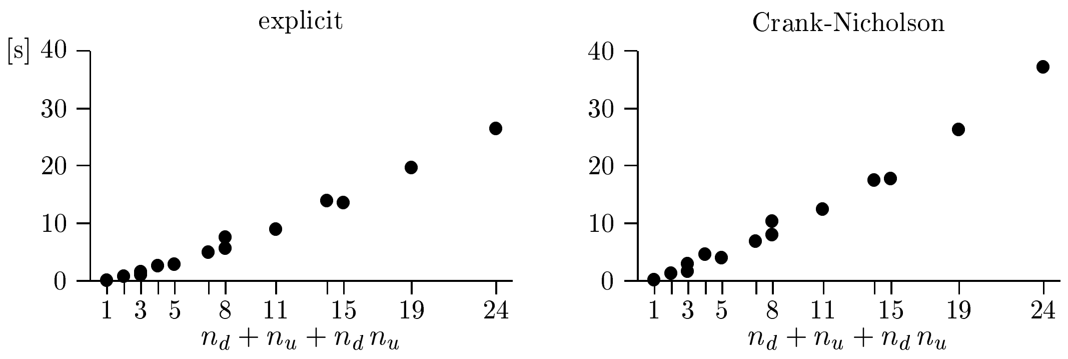


Figure 6.16: The running times in seconds under the explicit resp. Crank-Nicholson schemes, plotted against the number of lattice instances necessary to process various combinations of single-barrier options

7 Algorithms for American Options

Just like barrier options, American options may be subjected to premature termination sometime between the settlement and the face maturity date. This seems to make the techniques discussed in Chapters 5 and 6 applicable to portfolios that contain American options as well.

This is true—in principle. The fundamental difference is that the early exercise of American options is voluntary. The precise date is not known a priori, although it may be assumed that holders of American options time early exercise so as to maximize their expected payoff. How mathematical finance models this behavior can be read in Bensoussan (1984) and Karatzas (1988). Numerically, American options can be evaluated with projected SOR (Successive Over-Relaxation) methods on a lattice or tree, as described in Wilmott *et al.* (1993). Other approaches are also possible (see Longstaff and Schwartz (1998), for instance).

Uncertainty in some of the model coefficients adds another twist to the problem: the early exercise strategy for an individual American option X_1 now depends on the entire position $(\mathbf{X}, \lambda)$, not merely on the contribution of X_1 judged separately. We have encountered this situation with barrier options: once X_1 is exercised, the exposure of the remaining partial portfolio to fluctuations in the uncertain coefficients may be different, and so may the worst-case value. Thus it is not possible to pre-process the components of $\mathbf{X}$ with American early exercise features separately and use the so found early exercise boundaries just like knock-out boundaries are used in the case of barrier options. Rather, the continuation and early exercise regions for each American option in $\mathbf{X}$ must be searched for dynamically, by considering the consequences of all possible early exercise strategies (of which there are plenty if $\mathbf{X}$ contains several American options) on the worst-case value of the entire position.

In this chapter, we show how to implement the dynamic search for continuation and early exercise regions within the framework of worst-case pricing. We also show how to cope with the explosion of combinations: it is possible to reduce the combinatorial complexity in most practical cases considerably.

In some sense, the concept of optional early exercise *is* merely an extension of the notion of forced knock-out. The algorithms in this chapter work for both American and barrier options indiscriminately. In particular, they are capable of pricing a portfolio of

American barrier options!

Once again, σ shall be the only uncertain model coefficient.

7.1 Early Exercise Combinations

We use a lattice approach as proposed in Chapter 5. Let $(\mathbf{X}, \lambda)$ be the portfolio, and assume all k instruments in $\mathbf{X}$ mature at time T, and may be exercised early at any time between now and T.

The local data flow shown in Figs. 5.2 and 5.3 captures the situation only partially if American options are present. $\hat{V}(j, i; L)$, unmodified, represents the worst-case portfolio value under the restriction that no option be exercised at $t = t_i$ and $S_i = s_j$ (recall that S_i is an abbreviation for S_{t_i} and s_j is the jth spatial level of the lattice). $\hat{V}(j, i; L)$ needs to be compared to other worst-case values that arise under viable early exercise combinations, and a proper selection needs to be made. Thus, the original scheme turns into a two-tiered numerical-combinatorial regime:

1. The finite-difference scheme is applied to find the worst-case value under a no-exercise assumption. This is the numerical phase.

2. This preliminary value competes against the worst-case values delivered by all viable early exercise combinations. It may or may not be updated. This is the combinatorial phase.

$\hat{V}(j, i; L)$ is always paired with its gradient vector $(\hat{v}_1(j, i; L), \ldots, \hat{v}_k(j, i; L))^{\mathrm{T}}$. Although sometimes not explicitely mentioned, the gradient is always computed, and modified, together with $\hat{V}(j, i; L)$.

Figure 7.1 presents this approach graphically. Subordinate lattice instances need to be accessed in the combinatorial phase. As some early exercise combinations might be dismissed right away, exactly which lattice instances must be available at a given node instance $(j, i; L)$ is determined at runtime. Clever selection techniques lead to significant speedup.

Note that Fig. 7.1 is correct for explicit finite difference schemes, but not necessarily for mixed explicit/implicit schemes such as Crank-Nicholson. As the update of one $\hat{V}(j, i; L)$ affects all the others implicitely, iterative improvement over both phases 1 and 2, applied

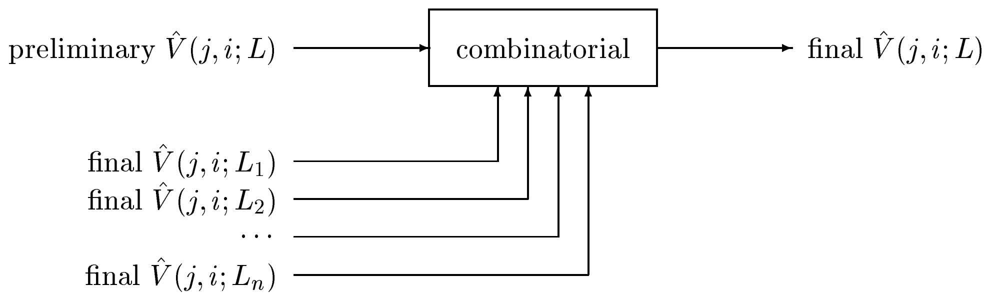


Figure 7.1: The preliminary worst-case value $\hat{V}(j,i;L)$ at node instance $(j,i;L)$, result of the dataflow in Fig. 5.2, enters the combinatorial post-processor which selects a suitable early exercise combination. To do that it needs to access lattice instances $L_1,\dots,L_n$, all carrying partial portfolios of $\mathbf{X}$

to all node instances $(\cdot,i;L)$ of the current time slice, is required. A modified projective SOR method, for instance, may do.

For this reason, all experimental results were obtained with explicit forward Euler. Although we have implemented Crank-Nicholson (and projected SOR), we focus on combinatorial aspects in this chapter and ignore the numerical side as much as possible.

7.1.1 Long and Short Positions

Which early exercise combinations should be adopted at (j,i,L)? Which combination is "suitable", in the language of Fig. 7.1? Simply choosing the one that represents the largest worst-case value is not sufficient, since control lies not only with the other party to whom some of the American instruments in the portfolio have been sold, but also with the agent who may *own* some of the American options.

The Worst-case Price Revisited

To clarify this point, we remind ourselves that

> the worst-case price of $(\mathbf{X},\lambda)$ represents the largest amount of funds that may be necessary to delta-hedge a portfolio $(\mathbf{X},\lambda)$. It thus represents the safe price which, when charged, guarantees that the seller of $(\mathbf{X},\lambda)$ will not incur any losses.

The worst-case price therefore represents the point of view of the sell-side:

- $\lambda_i > 0$ means that the agent has sold λ_i contracts in the ith instrument and does therefore not control the early exercise strategy for this instrument. (When the sell-side sells a long position in X_i, it effectively goes short X_i.)

- $\lambda_i < 0$ means that the agent has bought $|\lambda_i|$ contracts in the ith instrument and therefore controls its early exercise strategy.

The worst-case value of $(\mathbf{X}, \lambda)$ is the worst-case liability of the sell-side. Positive values mean that additional funds must be provided to hedge against the worst-case. They represent an upper bound for the price (i.e., the most desirable price) the seller can justifiably charge the buyer, assuming the buyer agrees with the seller on the uncertain model coefficients.

Similarly, negative values indicate a net flow of funds from the seller to the buyer; the absolute value represents a lower bound (corresponding to the unaltered value being an upper bound) on the amount of funds p the seller needs to transfer together with $(\mathbf{X}, \lambda)$. Any amount smaller than p is no longer competitive under the particular uncertainty assumptions of the model. Thus, p is the most desirable cost for the seller.

The Best Worst-case Price

In the previous paragraph we gave an economic justification for the maximum-principle in worst-case pricing. So far, however, agents from whose perspective the worst-case price is computed have been unable to modify their risk profile after the position $(\mathbf{X}, \lambda)$ has been set up.

This is different if $\mathbf{X}$ contains American instruments. Suppose X_i is American, and $\lambda_i < 0$. Define $\mathbf{X}' = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_k)^{\mathrm{T}}$, and λ' accordingly. Define $\mathbf{X}^e = \mathbf{X}$, with X_1 modified to preclude early exercise if $t = t_i$ (early exercise is still allowed for $t \geq t_{i+1}$). Let the signature of lattice instance L^e and L' be $(\mathbf{X}^e, \lambda)$ and $(\mathbf{X}', \lambda')$, respectively.

Consider the two quantities

$$\begin{aligned} V_1 &= \hat{V}(j, i; L^e) \\ V_2 &= \hat{V}(j, i; L') + \lambda_i X_i \end{aligned} \tag{7.7.1}$$

Both are worst-case prices: V_1 under the restriction that X_1 must not be exercised now, but may be later, and V_2 under the constraint that X_1 be forcibly exercised now. Let us assume that there are no other American options in $\mathbf{X}$. Then V_1 and V_2 exhaust the possible early exercise combinations for X_1 and the remainder of the portfolio, and we may express $\hat{V}(j,i;L)$ in terms of V_1 and V_2.

Since the agent may *choose* between V_1 and V_2, it is advisable to choose the *minimum.* This strategy reduces the worst-case liability of the agent and assures that the agent selects the most competitive price when $(\mathbf{X},\lambda)$ is offered for sale, while still being hedged against volatility fluctuations.

Definition 7.1 (Best worst-case). *The* best worst-case *value of portfolio* $(\mathbf{X},\lambda)$ *is the minimal worst-case value of* $(\mathbf{X},\lambda)$*, where the minimum is taken over the early exercise strategies open to the seller of* $(\mathbf{X},\lambda)$.

We illustrate the principle of best worst-case evaluation with an example, and give a formalization in Sect. 7.1.2.

Assume $\mathbf{X} = (X_1, X_2)^{\mathrm{T}}$ and $\lambda = (-1, 1)^{\mathrm{T}}$. Both instruments are American. The seller of $(\mathbf{X},\lambda)$ controls early exercise for X_1, but is subjected to any early exercise decisions made by the holder of X_2. Suppose the outcomes of the four early exercise combinations at node instance (j,i) are those shown in Fig 7.2:

- 40 if X_1 and X_2 are both exercised (payoff of X_1 plus payoff of X_2);
- 10 if only X_1 is exercised (payoff of X_1, plus worst-case value of X_2 under the restriction that X_2 not be exercised at node (j,i));
- 20 if only X_2 is exercised (payoff of X_2, plus best worst-case value of X_1 under the restriction that X_1 not be exercised at node (j,i));
- 30 if neither instrument is exercised (best worst-case value under the restriction that X_1 and X_2 not be exercised at node (j,i)).

Definition 7.1 requires a strategy for X_1 that guarantees the lowest value under all possible decisions of the holder of X_2. The agent therefore selects the strategy with the lowest row maximum. In the example, the agent postpones the exercise of X_1 at least until time t_{i+1} and thus guarantees a worst-case value of 30.

	exercise X_2	don't exercise X_2
exercise X_1	**40**	10
don't exercise X_1	20	$\boxed{\mathbf{30}}$

Figure 7.2: Early exercise combinations at node (j, i) and their corresponding values. Bold values are row maxima; the framed value is the best worst-case

7.1.2 Best Worst-case Evaluation Formalized

Most of the exposition in this section is taken from Buff (1999a) and reformulated for a discrete setting.

Some notational remarks. In general, if nothing else is said, the lattice instance associated with signature $(\mathbf{X}, \lambda)$ is denoted by L, and vice versa. Its size is denoted $|L| = |\mathbf{X}| = |\lambda| = k$. In some cases, however, $\mathbf{X}_L$ and λ_L express this relationship explicitely. If $(\mathbf{X}', \lambda') \subseteq (\mathbf{X}, \lambda)$ is partial, and L' is the corresponding lattice instance, we refer to L' as a sub-lattice instance. L is sometimes called the root-lattice instance.

Definition 7.2 (Separation into long and short). *Let $(\mathbf{X}', \lambda') \subseteq (\mathbf{X}, \lambda)$ be a partial portfolio of size k'. Then*

$$\begin{aligned} \operatorname{long}\left(\mathbf{X}', \lambda'\right) &= \left\{1 \leq n \leq k' \mid \lambda'_n < 0 \textit{ and } X'_n \textit{ is American}\right\} \\ \operatorname{short}\left(\mathbf{X}', \lambda'\right) &= \left\{1 \leq n \leq k' \mid \lambda'_n > 0 \textit{ and } X'_n \textit{ is American}\right\} \\ \operatorname{am}\left(\mathbf{X}'\right) &= \left\{1 \leq n \leq k' \mid \lambda'_n \neq 0 \textit{ and } X'_n \textit{ is American}\right\} \end{aligned} \tag{7.7.2}$$

separate the American instruments in $\mathbf{X}'$ into long and short positions. (Recall that for the sell-side, $\lambda'_n > 0$ translates to X'_n being sold.)

Definition 7.3 ("Europeanization"). *Let $\mathbf{X}$ be a portfolio, and $X_n \in \operatorname{am}(\mathbf{X})$ one of its American instruments. Then X_n^E is its "europeanized" version: early exercised is precluded everywhere. If G is any process involving X_n, then we write G^E for the corresponding process involving X_n^E (where "corresponding" depends on the context). Similarly, L^E is a lattice instance whose signature contains europeanized versions $X_1^E, \ldots$ of American instruments $X_1, \ldots$.*

Definition 7.4 (Residual lattice instance). *Let L be a lattice instance with signature $(\mathbf{X}, \lambda)$, and let $M \subseteq \{1, \dots, |L|\}$. Then $\neg M = \{1, \dots, |L|\} \setminus M$, and $L_{\neg M}$ with signature*

$$(\text{select}\,(\mathbf{X}, \neg M)\,, \text{select}\,(\lambda, \neg M)) \tag{7.7.3}$$

is called the residual lattice instance *of L.*

Definition 7.5 (Payoff from early exercise). *Given a lattice instance L with signature $(\mathbf{X}, \lambda)$ and an enumeration of instruments $M \subseteq \text{am}\,(\mathbf{X})$. Then the* payoff from early exercise *of the instruments in M is given by the linear combination*

$$\text{payoff}\,(L, M) = \text{select}\,(\lambda, M) \cdot \text{select}\,(\mathbf{X}, M) \tag{7.7.4}$$

The Best Worst-case Price Process

Definition 7.6 (Local fixation of early exercise). *Let $\hat{F} = \{\hat{F}_i\}$ be any discrete process defined on the space of node instances. $\hat{F}_i(L)$ is a random variable; $\hat{F}_i(j; L)$ its value at node instance $(j, i; L)$. Assume $(\mathbf{X}, \lambda)$ is the signature of L, and choose $\sigma \in \mathcal{C}$.*

Then we define the local fixation *of $\hat{F}$ for $M \subseteq \text{am}\,(\mathbf{X})$ as follows:*

$$F_i(L, M, \sigma) = \frac{1}{\beta_i}\,\mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L_{\neg M}) \mid \mathcal{F}_i\right) + \text{payoff}\,(L, M) \tag{7.7.5}$$

where $L_{\neg M}$ is the residual lattice instance of L. $F_i(L, M, \sigma)$ is also a random variable, and $F_i(j; L, M, \sigma)$ its value at node instance $(j, i; L)$.

In some sense, the local fixation $F_i(L, M, \sigma)$ "harnesses" the power of $\hat{F}_i(L)$ by fixing the volatility σ as well as the early exercise strategy for one time period. The parameter M in Def. 7.6 has the effect of modifying the features of $\mathbf{X}$ locally. The maturity date of the instruments covered by M is advanced to t_i, and for all other instruments the earliest date on which early exercise is permissible is set to t_{i+1}. It is easy to see that F is adaptable.

The following definitions remove the restrictions on the parameters M and σ in $F(L, M, \sigma)$ again. The result will not be the original $\hat{F}$, but a version that incorporates the best worst-case paradigm in Def. 7.1.

Definition 7.7 (Local uncertainty reintroduced). *Let $F(L, M, \sigma)$ be a local fixation. Then $\hat{F}(L, M) = \{\hat{F}_i(L, M)\}$, defined as*

$$\hat{F}_i(L, M) = \sup_{\sigma \in \mathcal{C}} F_i(L, M, \sigma) \tag{7.7.6}$$

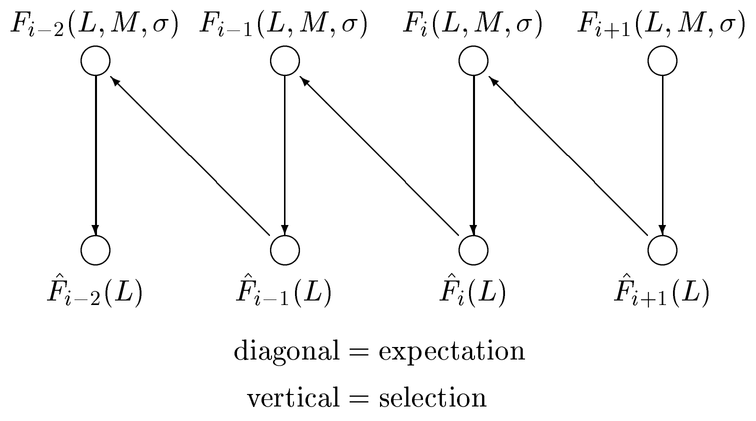


Figure 7.3: An illustration of Def. 7.6. The process $F(L,M,\sigma) = \{F_i(L,M,\sigma)\}$ depends on the values the base process $\hat{F}(L) = \{\hat{F}_i(L)\}$ takes on in subsequent time slices, by taking the (discounted) expectation over all possible transitions. Although not implied in Def. 7.6, we shall see later that $\hat{F}(L)$ in turn may depend on $F(L,M,\sigma)$, by selecting among feasable instantiations of M and σ

reintroduces uncertainty *in σ locally.*

Definition 7.8 (Local optionality reintroduced). *Let $\hat{F} = \{\hat{F}_i\}$ be any discrete process defined on the space of node instances. Let $\hat{F}(L,M,\sigma)$ be its local fixation, and let $\hat{F}(L,M)$ be the process defined in Def. 7.7. Let*

$$\begin{aligned} A(L) &= \operatorname{long}(\mathbf{X},\lambda) \\ B(L) &= \operatorname{short}(\mathbf{X},\lambda) \end{aligned} \tag{7.7.7}$$

denote the American instruments on lattice instance L with signature $(\mathbf{X},\lambda)$. Then the process $\hat{G}(L) = \{\hat{G}_i(L)\}$, defined as

$$\hat{G}_i(L) = \min_{A \subseteq A(L)} \max_{B \subseteq B(L)} \hat{F}_i(L, A \cup B) \tag{7.7.8}$$

is said to reintroduce optionality *locally.*

Definition 7.9 (Best worst-case process). *Given $(\mathbf{X},\lambda)$. Let $\hat{F} = \{\hat{F}_i\}$ be a discrete process defined on the space of node instances belonging to the set of root- and sub-lattice instances for $(\mathbf{X},\lambda)$, and let $\hat{G}_i$ reintroduce optionality locally. Then $\hat{F}$ is called a* best worst-case *process if*

$$\hat{F}_i(L) = \hat{G}_i(L) \tag{7.7.9}$$

for all lattice instances L with signatures $(\mathbf{X}', \lambda')$, $\mathbf{X}' \subseteq \mathbf{X}$ and $\lambda' \in \mathbb{R}^{|\mathbf{X}'|}$, and $0 \le i \le N - 1$. If furthermore the payoff condition

$$\hat{F}_N(L) = \lambda_L \cdot \mathbf{X}_L \tag{7.7.10}$$

holds for all those L, then $\hat{F}$ is called a best worst-case price *process for $(\mathbf{X}, \lambda)$.*

We require (7.7.9) and the payoff condition to hold for all possible positions, not only for those which can be constructed by removing elements from λ. This is necessary because auxiliary positions (especially in singleton partial portfolios) may be required to support the computation.

The vertical arrows in Fig. 7.3 now make sense: the process $\hat{F}$ in the picture is a best worst-case price process.

Notice how Defs. 7.7 and 7.8 implement the proper hierarchy of local minimization and maximization, corresponding to the following sequence of moves between the agent, the client(s) who hold the instruments sold by the agent, and the market:

1. The agent chooses instruments to exercise. Candidates are found in long $(\mathbf{X}, \lambda)$.
2. The client(s) decide likewise for the instruments enumerated in short $(\mathbf{X}, \lambda)$.
3. The market acts by "selecting" the volatility of the underlying asset until the subsequent time slice.

No other order is plausible. The minimization operator in (7.7.8) guarantees that the agent makes the best possible choice, assuming maximal adversion by the client(s) and by the market.

Proposition 7.10 (Uniqueness). *There is only one best worst-case price process for $(\mathbf{X}, \lambda)$.*

Proof. Let $\hat{F}$ and $\hat{G}$ be both best worst-case price processes for $(\mathbf{X}, \lambda)$. Since, by definition, both processes fulfill the payoff condition (7.7.10), they agree for $i = N$. Induction over $i = N - 1, \ldots, 0$ and the definiteness of Defs. 7.7 and 7.8, including (7.7.8), imply uniqueness. $\square$

Compatibility with Traditional Formulations

It is easy to verify that the definition of the best worst-case price process $\hat{F}$ for $(\mathbf{X}, \lambda)$ implies that

$$\hat{F}_i(L) = \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L) \mid \mathcal{F}_i \right) \tag{7.7.11}$$

if long $(\mathbf{X}, \lambda) =$ short $(\mathbf{X}, \lambda) = \emptyset$, i.e. if all instruments in $\mathbf{X} = \mathbf{X}^E$ are European. Expansion leads to

$$\begin{aligned} \hat{F}_i(L) &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\sup_{\sigma' \in \mathcal{C}} \mathrm{E}_{Q(\sigma')} \left(\beta_{i+2} \hat{F}_{i+2}(L) \mid \mathcal{F}_{i+1} \right) \mid \mathcal{F}_i \right) \\ &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\mathrm{E}_{Q(\sigma')} \left(\beta_{i+2} \hat{F}_{i+2}(L) \mid \mathcal{F}_{i+1} \right) \mid \mathcal{F}_i \right) \end{aligned} \tag{7.7.12}$$

where we exploit the fact that the outer and inner expectations are taken over distinct periods of time. The outer supremum is insensitive to changes of the values of the function σ for $t \notin [t_i, t_{i+1})$, and the inner supremum is insensitive to changes of the values of σ' for $t \notin [t_{i+1}, t_{i+2})$. Both operators can thus be merged. Telescoping leads to

$$\begin{aligned} \hat{F}_i(L) &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+2} \hat{F}_{i+2}(L) \mid \mathcal{F}_i \right) \\ & \cdots \\ &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_N \hat{F}_N(L) \mid \mathcal{F}_i \right) \\ &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_N \left(\lambda \cdot \mathbf{X} \right) \mid \mathcal{F}_i \right) \end{aligned} \tag{7.7.13}$$

This is the discrete version of (4.4.10) in Fact 4.6.

Now assume long $(\mathbf{X}, \lambda) = \emptyset$, but short $(\mathbf{X}, \lambda) \neq \emptyset$, i.e. some of the sold options are American, but none of the options held long (by the sell-side) are. In this case, the minimization operator in (7.7.8) is superfluous, and we get, after unrolling the definitions,

$$\hat{F}_i(L) = \max_{B \subseteq B(L)} \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L_{\neg B}) \mid \mathcal{F}_i \right) + \text{payoff}\,(L, B) \tag{7.7.14}$$

There are two cases:

1. The maximum is attained at $B = \emptyset$, or $\hat{F}_i(L) = \hat{F}_i(L, \emptyset)$. In this case, early exercise

of any American instrument does not make the situation worse, and we conclude

$$\begin{aligned}\hat{F}_i(L) &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1}\, \hat{F}_{i+1}(L) \mid \mathcal{F}_i\right) \\ &= \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \max_{B \subseteq B(L)} \hat{F}_{i+1}(L, B) \;\middle|\; \mathcal{F}_i\right)\end{aligned} \tag{7.7.15}$$

If, in turn, $\hat{F}_{i+1}(L) = \hat{F}_{i+1}(L, \emptyset)$ for all s_j, conditioned on $\mathcal{F}_i$, then

$$\hat{F}_i(L) = \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+2} \max_{B \subseteq B(L)} \hat{F}_{i+2}(L, B) \;\middle|\; \mathcal{F}_i\right) \tag{7.7.16}$$

and so on. Most of the time this will not be so, however, and branching into case 2 may occur, depending on $\mathcal{F}_{i+1}$.

2. The maximum is attained for some $B \neq \emptyset$, or $\hat{F}(L) \neq \hat{F}(L, \emptyset)$. Let $L' = L_{\neg B}$ be the residual lattice instance, with signature $(\mathbf{X}', \lambda') = (\mathbf{X}_B, \lambda_B)$. Now choose $B' \subseteq \operatorname{short}(\mathbf{X}', \lambda')$ such that a)

$$\hat{F}_i(L') = \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1}\, \hat{F}_{i+1}(L'_{\neg B'}) \mid \mathcal{F}_i\right) + \operatorname{payoff}\left(L', B'\right) \tag{7.7.17}$$

and b) no subset B'' for which (7.7.17) also holds has fewer elements. Then $B' = \emptyset$ must necessarily be true, for otherwise one could identify the instruments enumerated in B' in the original portfolio $\mathbf{X}$ (the indexing might be different). They wouldn't be covered by B since $B' \cap B = \emptyset$, and exercising them in addition to those in B would increase the worst-case value, contradicting the maximality under B in (7.7.14). (The argument uses the associativity of the maximum operator.) Thus, $\hat{F}_i(L') = \hat{F}_i(L', \emptyset)$, and

$$\hat{F}_i(L) = \hat{F}_i(L', \emptyset) + \operatorname{payoff}(L, B) \tag{7.7.18}$$

We have found the *free boundary.*

These two cases can be combined with the introduction of a discrete stopping time τ_i such that

$$\tau_i(L) = \inf\left\{i \leq u \leq N \mid \hat{F}_u(L) \neq \hat{F}_u(L, \emptyset) \text{ or } u = N\right\} \tag{7.7.19}$$

$\tau_i(L)$ marks the first time case 2 is encountered on lattice instance L. Combining cases 1 and 2, (7.7.14) can be re-written

$$\hat{F}_i(L) = \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{\tau_i(L)} \max_{B \subseteq B(L)} \left[\hat{F}_{\tau_i(L)}(L_{\neg B}) + \text{payoff}\,(L, B) \right] \,\middle|\, \mathcal{F}_i \right) \tag{7.7.20}$$

which is the form for the early exercise problem that can be found in standard textbooks, such as Duffie (1996) (in a linear setting).

Subadditivity Revalidated

Fact 4.7 states that worst-case prices are subadditive. In the following paragraphs, we show that best worst-case prices have the same property.

Definition 7.11 (Signature arithmetic). *Given $c \in \mathbb{R}_{++}$, a portfolio $\mathbf{X}$ of size k and two positions $\lambda, \lambda' \in \mathbb{R}^k$. The following symbols for lattice instances and signatures are associated:*

symbol	*signature*
L	$(\mathbf{X}, \lambda)$
L'	$(\mathbf{X}, \lambda')$
cL	$(\mathbf{X}, c\lambda)$
$-L$	$(\mathbf{X}, -\lambda)$
$L + L'$	$(\mathbf{X}, \lambda + \lambda')$

Proposition 7.12 (Subadditivity). *Given $c \in \mathbb{R}_{++}$, a portfolio $\mathbf{X}$ of size k and two orthogonal positions λ and λ', i.e., $\lambda_n \lambda'_n = 0$ for $1 \leq n \leq k$. Let $\hat{F}$ be the best worst-case process for $(\mathbf{X}, \lambda)$..*

Then, in the notation of Def. 7.11, the following holds for $0 \leq i \leq N$:

$$\begin{aligned} \hat{F}_i(cL) &= c\,\hat{F}_i(L) \\ \hat{F}_i(L + L') &\leq \hat{F}_i(L) + \hat{F}_i(L') \\ \hat{F}_i(L + L') &\geq \hat{F}_i(L) - \hat{F}'_i(-L') \end{aligned} \tag{7.7.21}$$

Proof. We only prove the second inequality, by induction over i. For $i = N$, equality holds in (7.7.21) throughout, as all instruments mature at time $t = t_N$.

So let $i < N$, and assume (7.7.21) is true for $i + 1$:

$$\hat{F}_{i+1}(L + L') \leq \hat{F}_{i+1}(L) + \hat{F}_{i+1}(L') \tag{7.7.22}$$

Let $M \subseteq \operatorname{am}(\mathbf{X})$ be a subset of American instruments on L (and on L' and $L+L'$, for that matter). If $M = \emptyset$ the residual lattice of $L+L'$ is $L+L'$ itself. In this case, direct application of (7.7.22) leads to

$$\begin{aligned} F_i(L+L', \emptyset, \sigma) &= \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \hat{F}_{i+1}(L+L') \mid \mathcal{F}_i\Big) \\ &\le \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \Big(\hat{F}_{i+1}(L) + \hat{F}_{i+1}(L')\Big) \mid \mathcal{F}_i\Big) \\ &= \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \hat{F}_{i+1}(L) \mid \mathcal{F}_i\Big) + \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \hat{F}_{i+1}(L') \mid \mathcal{F}_i\Big) \\ &= F_i(L, \emptyset, \sigma) + F_i(L', \emptyset, \sigma) \end{aligned} \tag{7.7.23}$$

If $M \neq \emptyset$ we have $(L+L')_{\neg M} \subset L + L'$. (7.7.22) does not directly validate

$$\hat{F}_{i+1}((L+L')_{\neg M}) \le \hat{F}_{i+1}(L_{\neg M}) + \hat{F}_{i+1}(L'_{\neg M}) \tag{7.7.24}$$

but we may hold (7.7.24) to be true, by nested application of the proposition for a portfolio of smaller size $k' = |(L+L')_{\neg M}| < k$. (For $k' = 0$, equality obviously holds.) Thus,

$$\begin{aligned} &F_i(L+L', M, \sigma) \\ &\quad= \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \hat{F}_{i+1}((L+L')_{\neg M}) \mid \mathcal{F}_i\Big) + \text{payoff}\,\big(L+L', M\big) \\ &\quad\le \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \Big(\hat{F}_{i+1}(L_{\neg M}) + \hat{F}_{i+1}(L'_{\neg M})\Big) \mid \mathcal{F}_i\Big) + \text{payoff}\,\big(L+L', M\big) \\ &\quad= \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \hat{F}_{i+1}(L_{\neg M}) \mid \mathcal{F}_i\Big) + \text{payoff}\,(L, M) \\ &\qquad\quad + \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \Big(\beta_{i+1} \hat{F}_{i+1}(L'_{\neg M}) \mid \mathcal{F}_i\Big) + \text{payoff}\,\big(L', M\big) \\ &\quad= F_i(L, M, \sigma) + F_i(L', M, \sigma) \end{aligned} \tag{7.7.25}$$

and subadditivity is shown for the local fixation. Since this is true for all $\sigma \in \mathcal{C}$, we get

$$\begin{aligned} \sup_{\sigma \in \mathcal{C}} F_i(L+L', M, \sigma) &\le \sup_{\sigma \in \mathcal{C}} \big[F_i(L, M, \sigma) + F_i(L', M, \sigma)\big] \\ &\le \sup_{\sigma \in \mathcal{C}} F_i(L, M, \sigma) + \sup_{\sigma' \in \mathcal{C}} F_i(L', M, \sigma') \end{aligned} \tag{7.7.26}$$

or

$$\hat{F}_i(L+L', M) \le \hat{F}_i(L, M) + \hat{F}_i(L', M) \tag{7.7.27}$$

Thus, reintroducing local uncertainty does not violate subadditivity.

The remainder of the proof is concerned with retaining subadditivity through the application of the minimum and maximum operators. Just as in (7.7.7), define

$$\begin{aligned} A(L) &= \operatorname{long}(\mathbf{X}, \lambda) & B(L) &= \operatorname{short}(\mathbf{X}, \lambda) \\ A(L') &= \operatorname{long}(\mathbf{X}, \lambda_{L'}) & B(L') &= \operatorname{short}(\mathbf{X}, \lambda_{L'}) \\ A(L+L') &= \operatorname{long}(\mathbf{X}, \lambda_{L+L'}) & B(L+L') &= \operatorname{short}(\mathbf{X}, \lambda_{L+L'}) \end{aligned} \tag{7.7.28}$$

The orthogonality of λ and λ' and Def. 7.2 imply that

$$\begin{aligned} A(L+L') &= A(L) \cup A(L') \\ B(L+L') &= B(L) \cup B(L') \end{aligned} \tag{7.7.29}$$

where the union is direct, i.e. $A(L) \cap A(L') = \emptyset$ and $B(L) \cap B(L') = \emptyset$.

Now partition $M = A \cup B$ with $A \subseteq A(L+L')$ and $B = B(L+L')$. Then

$$\begin{aligned} \operatorname{payoff}(L, M) &= \operatorname{payoff}(L, A \cup B) \\ &= \operatorname{payoff}(L, (A \cap A(L)) \cup (B \cap B(L))) \\ \operatorname{payoff}\left(L', M\right) &= \operatorname{payoff}\left(L', A \cup B\right) \\ &= \operatorname{payoff}\left(L', \left(A \cap A(L')\right) \cup \left(B \cap B(L')\right)\right) \end{aligned} \tag{7.7.30}$$

and consequently

$$\begin{aligned} \hat{F}_i(L, M) &= \hat{F}_i(L, A \cup B) \\ &= \hat{F}_i(L, (A \cap A(L)) \cup (B \cap B(L))) \\ \hat{F}_i(L', M) &= \hat{F}_i(L', A \cup B) \\ &= \hat{F}_i(L', \left(A \cap A(L')\right) \cup \left(B \cap B(L')\right)) \end{aligned} \tag{7.7.31}$$

Although we do not show this in every detail, (7.7.30) and (7.7.31) are easy to validate, since $A \setminus A(L)$ and $B \setminus B(L)$ respectively $A \setminus A(L')$ and $B \setminus B(L')$ refer to vanishing positions: the correspondings λ's respectively λ''s are all zero. It is straightforward to equate payoff terms and signatures of lattice instances that differ only on instruments whose position is zero.

Reintroducing local optionality, it follows from (7.7.27) that

$$\begin{aligned} \hat{F}_i(L+L') &= \min_{A \subseteq A(L+L')} \max_{B \subseteq B(L+L')} \hat{F}_i(L+L', A \cup B) \\ &\leq \min_{A \subseteq A(L+L')} \max_{B \subseteq B(L+L')} \left(\hat{F}_i(L, A \cup B) + \hat{F}_i(L', A \cup B)\right) \end{aligned} \tag{7.7.32}$$

With (7.7.31),

$$\begin{aligned}
&\min_{A\subseteq A(L+L')} \max_{B\subseteq B(L+L')} \Big(\hat{F}_i(L, A\cup B) + \hat{F}_i(L', A\cup B)\Big) \\
&\qquad = \min_{A\subseteq A(L+L')} \max_{B\subseteq B(L+L')} \Big(\hat{F}_i\big(L, (A\cap A(L))\cup(B\cap B(L))\big) \\
&\qquad\qquad + \hat{F}_i\big(L', \big(A\cap A(L')\big)\cup\big(B\cap B(L')\big)\big)\Big) \qquad (7.7.33) \\
&\qquad \le \min_{A\subseteq A(L+L')} \Big(\max_{B\subseteq B(L+L')} \hat{F}_i\big(L, (A\cap A(L))\cup(B\cap B(L))\big) \\
&\qquad\qquad + \max_{B\subseteq B(L+L')} \hat{F}_i\big(L', \big(A\cap A(L')\big)\cup\big(B\cap B(L')\big)\big)\Big)
\end{aligned}$$

Some of the candidate sets searched by the maximum operators can be dropped, and the candidate sets for the minimum operator can be partitioned:

$$\begin{aligned}
&\min_{A\subseteq A(L+L')} \Big(\max_{B\subseteq B(L+L')} \hat{F}_i\big(L, (A\cap A(L))\cup(B\cap B(L))\big) \\
&\qquad\qquad + \max_{B\subseteq B(L+L')} \hat{F}_i\big(L', \big(A\cap A(L')\big)\cup\big(B\cap B(L')\big)\big)\Big) \\
&\qquad = \min_{A\subseteq A(L+L')} \Big(\max_{B\subseteq B(L)} \hat{F}_i\big(L, (A\cap A(L))\cup B\big) \\
&\qquad\qquad + \max_{B\subseteq B(L')} \hat{F}_i\big(L', \big(A\cap A(L')\big)\cup B\big)\Big) \\
&\qquad = \min_{A_1\subseteq A(L)} \min_{A_2\subseteq A(L')} \Big(\max_{B\subseteq B(L)} \hat{F}_i(L, A_1\cup B) + \max_{B\subseteq B(L')} \hat{F}_i(L', A_2\cup B)\Big) \qquad (7.7.34)
\end{aligned}$$

Rearranging terms yields

$$\begin{aligned}
&\min_{A_1\subseteq A(L)} \min_{A_2\subseteq A(L')} \Big(\max_{B\subseteq B(L)} \hat{F}_i(L, A_1\cup B) + \max_{B\subseteq B(L')} \hat{F}_i(L', A_2\cup B)\Big) \\
&\qquad = \min_{A_1\subseteq A(L)} \Big(\max_{B\subseteq B(L)} \hat{F}_i(L, A_1\cup B) + \min_{A_2\subseteq A(L')} \max_{B\subseteq B(L')} \hat{F}_i(L', A_2\cup B)\Big) \qquad (7.7.35) \\
&\qquad = \min_{A_1\subseteq A(L)} \max_{B\subseteq B(L)} \hat{F}_i\big(L, A_1\cup B\big) + \min_{A_2\subseteq A(L')} \max_{B\subseteq B(L')} \hat{F}_i(L', A_2\cup B)
\end{aligned}$$

Since, by definition,

$$\min_{A_1\subseteq A(L)} \max_{B\subseteq B(L)} \hat{F}_i(L, A_1\cup B) = \hat{F}_i(L) \qquad (7.7.36)$$

and

$$\min_{A_2\subseteq A(L')} \max_{B\subseteq B(L')} \hat{F}_i(L', A_2\cup B) = \hat{F}_i(L') \qquad (7.7.37)$$

we conclude from the sequence (7.7.32) through (7.7.35) that

$$\hat{F}_i(L + L') \leq \hat{F}_i(L) + \hat{F}_i(L') \tag{7.7.38}$$

This completes the induction step and the proof. $\square$

It should be obvious that the assertions of Prop. 7.12 also hold simultaneously for all partial portfolios $(\mathbf{X}', \lambda') \subseteq (\mathbf{X}, \lambda)$.

In the proposition, the portfolio $\mathbf{X}$ is split into halves $(X_n \mid \lambda_n \neq 0)^{\mathrm{T}}$ and $(X_n \mid \lambda_n = 0)^{\mathrm{T}}$. A different formulation uses two partial portfolios $(\mathbf{X}_i, \lambda_i) \subseteq (\mathbf{X}, \lambda)$, $i = 1, 2$, that do not overlap: $(\mathbf{X}_i, \lambda_i) = (\mathrm{select}\,(\mathbf{X}, M_i)\,, \mathrm{select}\,(\lambda, M_i))$ with $M_1 \cap M_2 = \emptyset$ and $M_1 \cup M_2 = \{1, \ldots, |\mathbf{X}|\}$. L_1, L_2 and $-L_2$ being the lattice instances with signatures $(\mathbf{X}_1, \lambda_1)$, $(\mathbf{X}_2, \lambda_2)$ and $(\mathbf{X}_2, -\lambda_2)$, respectively, (7.7.21) reads

$$\begin{aligned} \hat{F}_i(L) &\leq \hat{F}_i(L_1) + \hat{F}_i(L_2) \\ \hat{F}_i(L) &\geq \hat{F}_i(L_1) - \hat{F}'_i(-L_2) \end{aligned} \tag{7.7.39}$$

We will refer to the assertions of the proposition in either form, depending on the context.

Implementation

In principle, we have already seen in Fig. 7.1 how the best worst-case process for $(\mathbf{X}, \lambda)$ can be implemented by applying dynamic programming principles locally. The variables $\hat{V}(j, i; L)$ are discrete approximizations of the values $\hat{F}_i(L \mid S_i = s_j)$ of the best worst-case process. The results achieved so far motivate the algorithm in Fig. 7.4 to compute the "suitable early exercise combination" mentioned in the caption of Fig. 7.1.

7.2 Speedup Techniques

The term

$$\hat{F}_i(L) = \min_{A \subseteq A(L)} \max_{B \subseteq B(L)} \hat{F}_i(L, A \cup B) \tag{7.7.40}$$

has $2^{|A(L)|+|B(L)|}$ subexpressions. If corresponds to step 3b in Fig. 7.4. If the signature of L is $(\mathbf{X}, \lambda)$ and there are $n \leq k$ American instruments in $\mathbf{X}$, the running time of the worst-case pricer becomes $O(2^n)$, which is quite unacceptable.

In this section, we explore two ways of improving this performance impasse:

Input: Lattice instance L with signature $(\mathbf{X}, \lambda)$, time i

Output: $\hat{V}(j, i; L)$ for $D \leq j \leq U$

1. Repeat for all spatial levels $D \leq j \leq U$, possibly with the algorithm in Fig. 5.5:
   (a) Set $\hat{V}(j, i; L, \emptyset) \doteq \sup_{\sigma \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L) \mid S_j = s_i \right)$
   (b) Set the gradient $\hat{v}_n(j, i; L, \emptyset)$ accordingly, for $1 \leq n \leq |L|$:
   $$\hat{v}_n(j, i; L, \emptyset) = \frac{\partial}{\partial \lambda_n} \hat{V}(j, i; L, \emptyset)$$
2. Set $A(L) = \text{long}\,(\mathbf{X}, \lambda)$ and $B(L) = \text{short}\,(\mathbf{X}, \lambda)$
3. Solve the local minmax problem:
   (a) For all $\emptyset \neq M \subseteq A(L) \cup B(L)$, check whether $\hat{V}(j, i; L', \emptyset)$ has already been computed, where the signature of L' is $(\text{select}\,(\mathbf{X}, M), \text{select}\,(\lambda, M))$. If not, interrupt and recurse
   (b) Find $\hat{A} \subseteq A(L)$ and $\hat{B} \subseteq B(L)$ such that
   $$\hat{V}(j, i; L, \hat{A} \cup \hat{B}) = \min_{A \subseteq A(L)} \max_{B \subseteq B(L)} \hat{V}(j, i; L, A \cup B)$$
   and set $M = \hat{A} \cup \hat{B}$, $\hat{\mathbf{X}} = \text{select}\,(\mathbf{X}, M)$, and $\hat{\lambda} = \text{select}\,(\lambda, M)$
   (c) If $M = \emptyset$ set $\hat{V}(j, i; L) = \hat{V}(j, i; L, \emptyset)$ and $\hat{v}_n(j, i; L) = \hat{v}_n(j, i; L, \emptyset)$, $1 \leq n \leq |L|$. Otherwise let $\hat{L}$ be the lattice instance with signature $(\hat{\mathbf{X}}, \hat{\lambda})$, set $\hat{V}(j, i; L) = \hat{V}(j, i; \hat{L}) + \text{payoff}\,(L, M)$, and copy $\hat{v}_n(j, i; L)$ from $\hat{L}$ for $n \in \{1, \ldots, |L|\} \setminus M$, after proper reindexing. For all other n, set $\hat{v}_n(j, i; L) = X_n$

Figure 7.4: The algorithm to track the best worst-case process on the lattice. In a real implementation, step 1 is one round in an explicit or mixed explicit/implict finite difference scheme, based on PDE's of type (4.4.8). Step 2 offers potential for improvement. The temporary vector $\hat{V}(\cdot, i; L, \emptyset)$ is the discrete version of $\hat{F}(L, \emptyset)$

1. Sometimes it can be said with certainty that a particular instrument X_n on a lattice instance L must or must not be exercised, regardless of the remaining position. Only where such certainty cannot be gained is it necessary to consider both possibilities in concert with all other instruments.

2. In the linear case the simple cutoff rule

$$X_n \geq \hat{v}_n(i, j; L, \emptyset) \tag{7.7.41}$$

comparing the potential payoff with the prospective future profit determines the early exercise boundary. This formula is no longer true in the nonlinear case, but it might well be useful as heuristic.

In both cases, space is partitioned onto three regions.

Definition 7.13 (Corridor of uncertainty). *Let L be a lattice instance with signature $(\mathbf{X}, \lambda)$. Choose $n \in \mathrm{am}(\mathbf{X})$. Let $(j, i; L)$ be a node instance.*

If early exercise of X_n is a priori not pursued at $(j, i; L)$ then $(j, i; L)$ belongs to the continuation region *of X_n.*

If early exercise of X_n is a priori opted for at $(j, i; L)$ then $(j, i; L)$ belongs to the exercise region *of X_n.*

In early exerise is a priori neither avoided nor opted for at $(j, i; L)$ then $(j, i; L)$ belongs to the corridor of uncertainty *of X_n. In its corridor of uncertainty, X_n contributes to the exponential complexity of* (7.7.40)*.*

Notice that the terminology is operational: the continuation region of X_n is not the region in which not exercising X_n is optimal in the sense of the minmax formulation. Rather, continuation region, exercise region and corridor of uncertainty are established externally; then (7.7.40) is applied at each node instance (j, i, L) that belongs to the corridor of uncertainty of at least one instrument.

The computational complexity is still exponential in n where n corridors of uncertainty overlap. Figure 7.5 shows cases with non-overlapping and overlapping corridors of uncertainty.

It is crucial to keep the corridor of uncertainty as small as possible. The first speedup approach uses a worst-case and best-case price band for each individual American instrument to estimate the corridor of uncertainty. This approach never misses the correct combination $(\hat{A}, \hat{B})$ in step 3b of Fig. 7.4 and is thus equivalent.

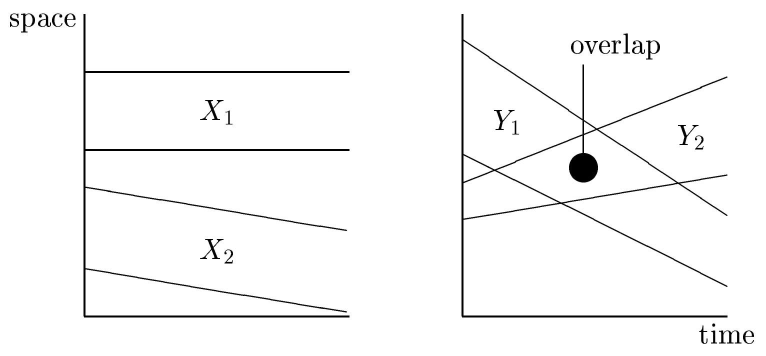


Figure 7.5: Shown on the left side are the non-overlapping corridors of uncertainty for two American options X_1 and X_2. Within each corridor 2 early exercise alternatives must be considered: execise X_1 resp. X_2 versus don't exercise X_1 resp. X_2. On the right side, the corridors of uncertainty for the American options Y_1 and Y_2 overlap. In the overlap region, 4 early exercise alternatives must be considered. (This picture is conceptual. For an actual example, see Fig. 7.8)

The second speedup approach collapses the corridor of uncertainty. The formula (7.7.41) is used to separate space into regions of continuation and exercise, respectively. The corridor of uncertainty is empty. This approach is no longer guaranteed to be correct; it is a heuristic. It handles American options much like barrier options. For instance, it can be made to process barrier options with irregular barriers, by replacing (7.7.41) with $s_j \geq U(X_n, i)$ where U maps to a time-dependent up-and-out barrier for X_n.

These techniques require a refinement of steps 2 and 3 in Fig. 7.4. A general template of the necessary changes is offered in Fig. 7.6.

7.2.1 Maintaining the Corridor of Uncertainty

So far we have created partial portfolios $(\mathbf{X}', \lambda') \subseteq (\mathbf{X}, \lambda)$ only when they were necessary as sources of boundary values. In this section we shall see how the separate computation of best worst-case prices for $(X_n, 1)$ and $(X_n, -1)$, $1 \leq n \leq k$, can help to eliminate early exercise combinations without sacrificing correctness. (The notation $(X_n, \pm 1)$ is used as shorthand for the vector pair $((X_n), (\pm 1))$ throughout this section and the next.)

2$*$ Partition $\text{long}\,(\mathbf{X}, \lambda) = A_C \cup A_U \cup A_E$ and $\text{short}\,(\mathbf{X}, \lambda) = B_C \cup B_U \cup B_E$, where the subscripts C, U and E stand for continuation region, uncertainty corridor and exercise region, respectively

3$*$ Solve the local minmax problem:

(a) For all $M \subseteq A_U \cup B_U$, interrupt and recurse if $\hat{V}(j, i; L', \emptyset)$ has not been computed yet. Here, the signature of L' is

$$(\text{select}\,(\mathbf{X}, M \cup A_E \cup B_E)\,, \text{select}\,(\lambda, M \cup A_E \cup B_E))$$

(b) Find $\hat{A} \subseteq A_U$ and $\hat{B} \subseteq B_U$ such that

$$\begin{aligned}&\hat{V}(j, i; L, \hat{A} \cup \hat{B} \cup A_E \cup B_E)\\ &\qquad = \min_{A \subseteq A_U} \max_{B \subseteq B_U} \hat{V}(j, i; L, A \cup B \cup A_E \cup B_E)\end{aligned}$$

and set $M = \hat{A} \cup \hat{B} \cup A_E \cup B_E$, $\hat{\mathbf{X}} = \text{select}\,(\mathbf{X}, M)$, and $\hat{\lambda} = \text{select}\,(\lambda, M)$

(c) $\ldots$ (just as step 3c in Fig. 7.4)

Figure 7.6: Steps 2$*$ and 3$*$ replace steps 2 and 3 in Fig. 7.4. Step 2$*$ is still generic: it does not provide guidelines as to how to partition the long and short positions. Sections 7.2.1 and 7.2.2 fill in the details

Proposition 7.14 (Corridor of uncertainty I). *Let L be a lattice instance of size k with signature $(\mathbf{X}, \lambda)$, and let $\hat{F}$ be the best worst-case price process. For $n \in \text{am}\,(\mathbf{X})$, let L^U be the lattice instance with signature $(X_n, 1)$, and let L_D be the lattice instance with signature $(X_n, -1)$. Then*

$$-\hat{F}_i(L_D) \leq \hat{F}_i(L_U) \tag{7.7.42}$$

for $0 \leq i \leq N$.

Proof. By induction over i. For $i = N$ we have equality, as X_n is always exercised at the maturity date t_N, and

$$-\hat{F}_N(L_D) = \hat{F}_N(L_U) = X_N \tag{7.7.43}$$

For $i < N$, by unrolling Defs. 7.5, 7.6, 7.7 and 7.8, we get

$$\begin{aligned} -\hat{F}_i(L_D) &= -\min\left\{-X_n,\ \sup_{\sigma\in\mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L_D) \mid \mathcal{F}_i\right)\right\} \\ &= \max\left\{X_n,\ -\sup_{\sigma\in\mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L_D) \mid \mathcal{F}_i\right)\right\} \end{aligned} \tag{7.7.44}$$

Using the induction hypothesis and linearity of expectation,

$$\begin{aligned} -\sup_{\sigma\in\mathcal{C}} &\frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L_D) \mid \mathcal{F}_i\right) \\ &= \inf_{\sigma\in\mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(-\beta_{i+1}\,\hat{F}_{i+1}(L_D) \mid \mathcal{F}_i\right) \\ &\le \inf_{\sigma\in\mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L^U) \mid \mathcal{F}_i\right) \\ &\le \sup_{\sigma\in\mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L^U) \mid \mathcal{F}_i\right) \end{aligned} \tag{7.7.45}$$

Together,

$$-\hat{F}_i(L_D) \le \max\left\{X_n,\ \sup_{\sigma\in\mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)}\left(\beta_{i+1}\,\hat{F}_{i+1}(L^U) \mid \mathcal{F}_i\right)\right\} = \hat{F}_i(L^U) \tag{7.7.46}$$

□

Proposition 7.14 shows that $-\hat{F}(L_D)$ and $\hat{F}(L^U)$ span a non-empty corridor between them. The following proposition shows that this corridor can be used to separate the continuation and exercise regions.

Proposition 7.15 (Corridor of uncertainty II). *Given a lattice instance L of size k with signature $(\mathbf{X}, \lambda)$. Let $\hat{F}$ be the best worst-case process for $(\mathbf{X}, \lambda)$. Choose $n \in \mathrm{am}(\mathbf{X})$. Set $A(L) = \mathrm{long}(\mathbf{X}, \lambda)$ and $B(L) = \mathrm{short}(\mathbf{X}, \lambda)$. Let $A'(L) = A(L) \setminus \{n\}$ and $B'(L) = B(L) \setminus \{n\}$ be their reduced versions. Let L^U be the lattice instance with signature $(X_n, 1)$, and let L_D be the lattice instance with signature $(X_n, -1)$.*

If $\hat{F}_i(L^U) \le X_n$ then

$$\hat{F}_i(L) = \min_{A\subseteq A'(L)} \max_{B\subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \tag{7.7.47}$$

i.e. X_n is exercised for sure.

If, on the other hand, $-\hat{F}_i(L_D) > X_n$ then

$$\hat{F}_i(L) = \min_{A\subseteq A'(L)} \max_{B\subseteq B'(L)} \hat{F}_i(L, A \cup B) \tag{7.7.48}$$

i.e. X_n is not exercised.

Proof. The proof uses Prop. 7.12. Set $M = \{1, \dots, n-1, n+1, \dots, k\}$ and

$$(\mathbf{X}', \lambda') = (\text{select}(\mathbf{X}, M), \text{select}(\lambda, M)) \tag{7.7.49}$$

Let L' be the lattice instance with signature $(\mathbf{X}', \lambda')$, and notice that

$$\hat{F}_i(L') = \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L', A \cup B) \tag{7.7.50}$$

Let L^+ be the lattice instance with signature (X_n, λ_n), and let L^- be the lattice instance with signature $(X_n, -\lambda_n)$.

Case 1 Assume $\hat{F}_i(L^U) \leq X_n$ and $\lambda_n > 0$, i.e. $n \in B(L)$ and $B'(L) \subset B(L)$. Then $\lambda_n \hat{F}(L^U) = \hat{F}(L^+)$ by the first assertion of Prop. 7.12. Furthermore, by the second assertion of Prop. 7.12,

$$\begin{aligned}
\hat{F}_i(L) &\leq \hat{F}_i(L^+) + \hat{F}_i(L') \\
&= \lambda_n \hat{F}_i(L^U) + \hat{F}_i(L') \\
&\leq \lambda_n X_n + \hat{F}_i(L') \\
&= \lambda_n X_n + \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L', A \cup B) \\
&= \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\})
\end{aligned} \tag{7.7.51}$$

The last transformation follows because a) $\lambda_n X_n$ can be pulled inside the payoff term of $\hat{F}_i(L', A \cup B)$, and b) the residual lattice instances of L with respect to $A \cup B \cup \{n\}$ and L' with respect to $A \cup B$ are identical.

For fixed A we have $\max_{B \subseteq B(L)} \hat{F}_i(L, A \cup B) \geq \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\})$, since in the latter term the maximum is taken over less values. Since $A(L) = A'(L)$,

$$\begin{aligned}
\hat{F}_i(L) &= \min_{A \subseteq A(L)} \max_{B \subseteq B(L)} \hat{F}_i(L, A \cup B) \\
&\geq \min_{A \subseteq A(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \\
&= \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\})
\end{aligned} \tag{7.7.52}$$

(7.7.51) and (7.7.52) together prove (7.7.47).

Case 2 If $\lambda_n < 0$, i.e. $n \in A(L)$ and $A'(L) \subset A(L)$, we reason similarly. By the first assertion of Prop. 7.12, $-\lambda_n \hat{F}(L^U) = \hat{F}(L^-)$. By the third assertion of Prop. 7.15,

$$\begin{aligned} \hat{F}_i(L) &\geq -\hat{F}_i(L^-) + \hat{F}_i(L') \\ &= -\left(-\lambda_n \hat{F}_i(L^U)\right) + \hat{F}_i(L') \\ &\geq \lambda_n X_n + \hat{F}_i(L') \\ &= \lambda_n X_n + \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L', A \cup B) \\ &= \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \end{aligned} \tag{7.7.53}$$

In the other direction we use $B(L) = B'(L)$ and get

$$\begin{aligned} \hat{F}_i(L) &= \min_{A \subseteq A(L)} \max_{B \subseteq B(L)} \hat{F}_i(L, A \cup B) \\ &\leq \min_{A \subseteq A'(L)} \max_{B \subseteq B(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \\ &= \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \end{aligned} \tag{7.7.54}$$

Again, (7.7.47) follows readily.

Case 3 Now assume $-\hat{F}_i(L_D) > X_n$ and $\lambda_n > 0$, i.e. $n \in B(L)$, $B'(L) \subset B(L)$. Then $\lambda_n \hat{F}(L_D) = \hat{F}(L^-)$ by the first assertion of Prop. 7.12. With the third assertion of the proposition and $A(L) = A(L')$,

$$\begin{aligned} \hat{F}_i(L) &\geq -\hat{F}_i(L^-) + \hat{F}_i(L') \\ &= -\lambda_n \hat{F}_i(L_D) + \hat{F}_i(L') \\ &> \lambda_n X_n + \hat{F}_i(L') \\ &= \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \end{aligned} \tag{7.7.55}$$

Choose $\hat{A} \subseteq A(L)$ and $\hat{B} \subseteq B(L)$ such that $\hat{F}_i(L) = \hat{F}_i(L, \hat{A} \cup \hat{B})$. If $n \in \hat{B}$ the strict inequalitiy in (7.7.55) leads to a contradiction. Thus, $n \notin \hat{B}$. Therefore, $\hat{B} \subseteq B(L) \backslash \{n\} = B'(L)$, and

$$\hat{F}_i(L) = \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B) \tag{7.7.56}$$

2∗∗ Set

$$\begin{aligned} A_C &= \Big\{ n \in \operatorname{long}(\mathbf{X},\lambda) \;\mid\; -\hat{V}(j,i;L_D^n) > X_n \Big\} \\ A_F &= \Big\{ n \in \operatorname{long}(\mathbf{X},\lambda) \;\mid \hat{V}(j,i;L_n^U) \le X_n \Big\} \\ A_M &= \operatorname{long}(\mathbf{X},\lambda) \setminus (A_C \cup A_F) \end{aligned}$$

and

$$\begin{aligned} B_C &= \Big\{ n \in \operatorname{short}(\mathbf{X},\lambda) \;\mid\; -\hat{V}(j,i;L_D^n) > X_n \Big\} \\ B_F &= \Big\{ n \in \operatorname{short}(\mathbf{X},\lambda) \;\mid \hat{V}(j,i;L_n^U) \le X_n \Big\} \\ B_M &= \operatorname{short}(\mathbf{X},\lambda) \setminus (B_C \cup B_F) \end{aligned}$$

Figure 7.7: An elaboration of step 2∗ in Fig. 7.6, based on Prop. 7.15. L_n^U is the lattice instance with signature $(X_n, 1)$, and L_D^n is the lattice instance with signature $(X_n, -1)$, insofar X_n is American. Proposition 7.14 guarantees that $A_C \cap A_F = B_C \cap B_F = \emptyset$

Case 4 If $\lambda_n < 0$, i.e. $n \in A(L)$ and $A'(L) \subset A(L)$, we find that $-\lambda_n \hat{F}(F_D) = \hat{F}(L^+)$ by the first assertion of Prop. 7.12. The second assertion of Prop. 7.12 and $B(L) = B(L')$ imply

$$\begin{aligned} \hat{F}_i(L) &\le \hat{F}_i(L^+) + \hat{F}_i(L') \\ &= -\lambda_n \hat{F}_i(L_D) + \hat{F}_i(L') \\ &< \lambda_n X_n + \hat{F}_i(L') \\ &= \min_{A \subseteq A'(L)} \max_{B \subseteq B'(L)} \hat{F}_i(L, A \cup B \cup \{n\}) \end{aligned} \tag{7.7.57}$$

The argument that concluded case 3 works in this case as well, and thus (7.7.47) is shown. This completes the proof. □

Figure 7.7 shows how Prop. 7.15 can be used to initialize the corridor of uncertainty in step 2∗ in Fig. 7.6. The discrete formulation in terms of node instances is straightforward. Note that up to $2k$ additional lattice instances must be maintained. The technique is thus not entirely overhead free, but the overhead is linear in the number of American options.

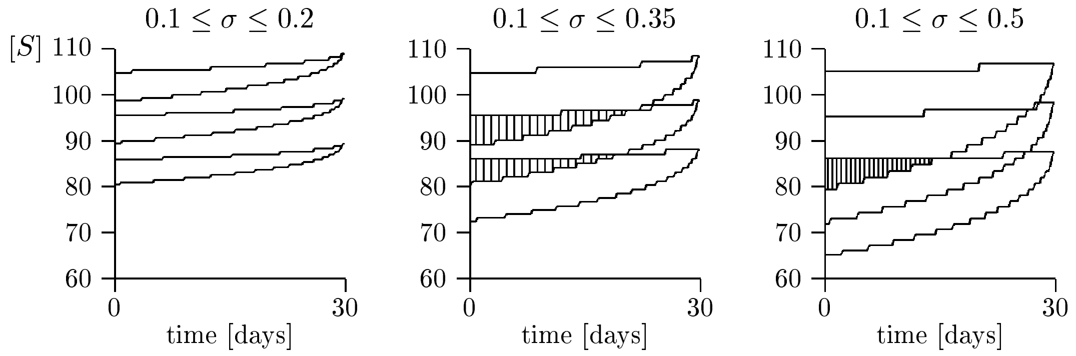


Figure 7.8: The corridor of uncertainty for three 30-day American puts, with strike 90, 100 and 110, respectively (from bottom to top). The interest rate is $r = 0.03$. The volatility range gets wider from left to right. In the left picture, there is no overlap; in the middle picture, corridors of uncertainty overlap pairwise; in the right picture, all corridors overlap in the shaded region

(In fact, it can be shown that the total number of lattice instances is bounded from above by $2^k + k - 1$. The exhausitve set of 2^k partial portfolios includes already k of the additional singleton lattice instances, and "-1" comes from the fact that the empty partial portfolio need not be carried on a lattice instance at all.)

Figure 7.8 shows the location of the corridor of uncertainty for three 30-day American puts with strikes 90, 100 and 110. Under a scenario in which the volatility stays within 10 and 20% the corridors do not overlap. If L is a lattice instance with signature $(\mathbf{X}, \lambda)$, and the three American puts are part of $\mathbf{X}$ (but no other American instruments are), then $|A_M| + |B_M| = 1$ in step 2∗∗, Fig. 7.7, in each corridor, and 0 otherwise. Under a 10–35% scenario, the corridors for the puts with strikes 90 and 100, and the corridors for the puts with strikes 100 and 110 overlap, respectively. Here, $|A_M| + |B_M| = 2$ in the shaded regions, leading to 4 combinations in the minmax term in step 3∗b, Fig. 7.6. Under a 10–50% scenario all corridors overlap, and $|A_M| + |B_M| = 3$ in the shaded region, leading to 8 combinations in the minmax term. The example demonstrates that the corridor of uncertainty is a powerful tool to reduce the combinatorial complexity of the best worst-case pricing problem if the volatility range is not too wide. It also shows that the method reverts to exponential complexity if the volatility range is extraordinarily wide. In the next section, a heuristic is presented that tries to alleviate this dependency.

7.2.2 Collapsing the Corridor of Uncertainty

The complexity of best worst-case pricing is potentially exponential if the corridors of uncertainty are nonempty and overlap. To collapse the corridor of uncertainty means to select a definite early exercise boundary, possibly within the corridor, that divides the lattice into continuation and exercise regions for each American option. The way in which this is done in the following paragraphs makes the approach heuristic: it does not guarantee that the resulting early exercise combinations reflect the local best worst-case selections adequately. We present, however, experimental results that show that the discrepancy is negligible in most cases.

The idea is to apply the rule commonly used in linear lattice-based pricing of American options: if the early exercise payoff exceeds the expected future payoff (which includes possible future early exercise), then the expected future payoff is locally replaced by the early exercise payoff. This cut-off rule dynamically assigns each lattice node to either the continuation or exercise region of the lattice.

Under nonlinearity, we do not have an isolated expected future payoff for an American option X_n which is part of a larger portfolio $(\mathbf{X}, \lambda)$. We do have, however, the discrete version of the gradient of the best worst-case value of $(\mathbf{X}, \lambda)$ with respect to the position of X_n, namely

$$\hat{v}_n(j, i; L) \doteq \frac{\partial}{\partial \lambda_n} \hat{F}_i(L \mid S_i = s_j) \tag{7.7.58}$$

(Discrete and continuous terms with analogue interpretations are equated with "$\doteq$".) As we will see in the following proposition, this gradient together with λ provides sufficient information to reconstruct the best worst-case value locally.

In this section, we adopt a candidate set

$$\mathcal{C} = \{\sigma \mid \sigma_{\min} \leq \sigma \leq \sigma_{\max}\} \tag{7.7.59}$$

with constants $0 < \sigma_{\min} \leq \sigma_{\max}$ for simplicity.

Definition 7.16 (Local stability). *Let L be a lattice instance with signature $(\mathbf{X}, \lambda)$. Choose $n \in \{1, \ldots, |L|\}$. We say $\hat{F}$ is* locally stable with respect to λ_n *if*

$$\frac{\partial}{\partial \lambda_n} \hat{F}_i(L) = \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, M) \tag{7.7.60}$$

whenever

$$\hat{F}_i(L) = \hat{F}_i(L, M) \tag{7.7.61}$$

for some $M \subseteq \{1, \dots, |L|\}$.

We are not too concerned with the situation in which $\hat{F}$ is not locally stable. In all practical cases, there are only finitely many regions in λ-space with distinct best worst-case exercise combinations at a given node instance (j, i). Finiteness suggests that whenever $\hat{F}_i(\cdot)$ switches from $\hat{F}_i(\cdot, M_1)$ to $\hat{F}_i(\cdot, M_2)$ with $M_1 \neq M_2$, say at $\lambda_n = \alpha$ with all other λ's unchanged, then there are intervals $(\alpha - \epsilon, \alpha)$ and $(\alpha, \alpha + \epsilon)$ in which M_1 respectively M_2 remain the best worst-case exercise combination. If we assume that partial derivatives in λ_n exist for $\hat{F}_i(\cdot)$, then by approching α from above and below it follows that $\hat{F}$ is locally stable with respect to λ_n.

The subsequent propositions are meant to *motivate* heuristic (7.7.58). We therefore leave it at this rather informal argument and, by assuming the existence of partial derivatives in λ, at the same time implicitely assert that Def. 7.16 applies to $\hat{F}$.

Proposition 7.17. *Let L be a lattice instance of size k with signature $(\mathbf{X}, \lambda)$. Assume the partial derivatives with respect to λ of the best worst-case price process $\hat{F}$ exist. Then the following identity holds:*

$$\sum_{n=1}^{k} \lambda_n \frac{\partial}{\partial \lambda_n} \hat{F}_i(L) = \hat{F}_i(L) \tag{7.7.62}$$

Proof. By induction over i. For $i = N$ we have

$$\hat{F}_N(L) = \text{payoff}\,(L, \{1, \dots, k\}) = \lambda \cdot \mathbf{X} \tag{7.7.63}$$

which, as linear combination of individual payoffs, obviously satisfies (7.7.62).

Now assume $i < N$. By induction hypothesis,

$$\sum_{n=1}^{k} \lambda_n \frac{\partial}{\partial \lambda_n} \hat{F}_{i+1}(L) = \hat{F}_{i+1}(L) \tag{7.7.64}$$

Case 1 Assume that $\hat{F}_i(L) = \hat{F}_i(L, \emptyset)$, i.e. no instruments are exercised in the best worst-case. Now recall that $\hat{F}_i(L, \emptyset \mid S_i = s_j) = \hat{f}(s_j, i)$, where $\hat{f}$ satisfies a PDE of type

(4.4.8) between times i and $i+1$ (during which interval no exercise takes place), and the boundary value is given by $\hat{F}_{i+1}(L)$. The worst-case spot volatility is determined by the spot convexity $x = \frac{\partial^2 \hat{f}}{\partial S^2}$ and the function

$$\Sigma(x) = \begin{cases} \sigma_{\max} & \text{if } x \geq 0 \\ \sigma_{\min} & \text{if } x < 0 \end{cases} \tag{7.7.65}$$

In particular,

$$\frac{\partial}{\partial \lambda_n} \Sigma(x) = 0 \tag{7.7.66}$$

for $x \neq 0$, and $\Sigma(x)\, x = 0$ for $x = 0$. Differentiating the PDE (4.4.8) with respect to λ_n shows that $\frac{\partial}{\partial \lambda_n} \hat{f}$ satisfies the same PDE, and so does any linear combination of partial derivatives.

We set the boundary at time $i+1$, and conclude furthermore from (7.7.64) that the boundary condition is the same for PDE (4.4.8) and the linear combination of its partial derivatives. Thus, by the uniqueness of the solution of (4.4.8),

$$\sum_{n=1}^{k} \lambda_n \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, \emptyset) = \hat{F}_i(L, \emptyset) \tag{7.7.67}$$

This completes the proof for case 1.

Case 2 Now assume $\hat{F}_i(L) = \hat{F}_i(L, M)$ for some $M \neq \emptyset$. Then, with $L' = L_{\neg M}$ being the residual lattice instance,

$$\begin{aligned} \hat{F}_i(L) &= \sup_{\sigma \in \mathcal{C}} F_i(L, M, \sigma) \\ &= \left[\sup_{\sigma \in \mathcal{C}} F_i(L', \emptyset, \sigma) \right] + \text{payoff}\,(L, M) \\ &= \hat{F}_i(L', \emptyset) + \text{payoff}\,(L, M) \end{aligned} \tag{7.7.68}$$

by the definition of $\hat{F}_i(L, M)$ and $F_i(L, M, \sigma)$ in Defs. 7.6 and 7.7, and switching between lattice instances.

Let $k' = |L| - |M| = |L'|$. The induction hypothesis (7.7.64) makes no statement for the smaller lattice instance L'. We may, however, apply the proposition on L' (inheriting all the premises) and conclude

$$\sum_{n=1}^{k'} (\lambda_{L'})_n \frac{\partial}{\partial (\lambda_{L'})_n} \hat{F}_i(L', \emptyset) = \hat{F}_i(L', \emptyset) \tag{7.7.69}$$

(A similar argument has been used in Prop. 7.12. The obvious base case $k' = 0$, or even $k' = 1$, can be worked out directly.)

Let $\{n_1, \dots, n_{|M|}\}$ be an enumeration of M. As in (7.7.63),

$$\text{payoff}\,(L, M) = \sum_{l=1}^{|M|} \lambda_{n_l} \frac{\partial}{\partial \lambda_{n_l}} \text{payoff}\,(L, M) \tag{7.7.70}$$

and hence, by adding (7.7.69) and (7.7.70) which sum over distinct positions,

$$\sum_{n=1}^{k} \lambda_n \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, M) = \hat{F}_i(L, M) \tag{7.7.71}$$

This completes the proof. □

The next two propositions establish upper and lower bounds for the partial derivatives of the best worst-case process.

Proposition 7.18 (Upper bounds for partial derivatives). *Let L be a lattice instance with signature $(\mathbf{X}, \lambda)$. Assume the partial derivatives of the best worst-case price process $\hat{F}$ in λ exist and are uniformly continuous. Choose $n \in \{1, \dots, |L|\}$ and let L^U be the lattice instance with signature $(X_n, 1)$. Then*

$$\frac{\partial}{\partial \lambda_n} \hat{F}_i(L) \leq \hat{F}_i(L^U) \tag{7.7.72}$$

Proof. By induction over i. If $i = N$ all instruments are forcibly exercised, and basic algebra shows that equality holds in (7.7.72). Thus assume $i < N$, and reason as in the proof of Prop. 7.17.

Case 1 Assume that $\hat{F}_i(L) = \hat{F}_i(L, \emptyset)$, i.e. no instruments are exercised in the best worst-case. Fix σ. Then, by uniform continuity and the induction hypothesis,

$$\begin{aligned}
\frac{\partial}{\partial \lambda_n} F_i(L, \emptyset, \sigma) &= \frac{\partial}{\partial \lambda_n} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L) \mid \mathcal{F}_i \right) \\
&= \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \frac{\partial}{\partial \lambda_n} \hat{F}_{i+1}(L) \mid \mathcal{F}_i \right) \\
&\leq \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L^U) \mid \mathcal{F}_i \right)
\end{aligned} \tag{7.7.73}$$

The local fixation of $\hat{F}_i(L^U)$ with respect to $\emptyset$ is

$$F_i(L^U, \emptyset, \sigma) = \frac{1}{\beta_i} \operatorname{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L^U) \mid \mathcal{F}_i \right) \tag{7.7.74}$$

and therefore

$$\begin{aligned} \hat{F}_i(L^U) &= \max \left\{ X_n, \sup_{\sigma' \in \mathcal{C}} F_i(L^U, \emptyset, \sigma') \right\} \\ &\geq F_i(L^U, \emptyset, \sigma) \end{aligned} \tag{7.7.75}$$

Together, (7.7.73), (7.7.74) and (7.7.75) show that

$$\frac{\partial}{\partial \lambda_n} F_i(L, \emptyset, \sigma) \leq \hat{F}_i(L^U) \tag{7.7.76}$$

(7.7.76) is true for every $\sigma \in \mathcal{C}$. Now let $\sigma_1, \sigma_2, \ldots$ be a sequence such that

$$\lim_{l \to \infty} F_i(L, \emptyset, \sigma_l) = \hat{F}_i(L, \emptyset) \tag{7.7.77}$$

Then,

$$\begin{aligned} \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, \emptyset) &= \frac{\partial}{\partial \lambda_n} \lim_{l \to \infty} F_i(L, \emptyset, \sigma_l) \\ &= \lim_{l \to \infty} \frac{\partial}{\partial \lambda_n} F_i(L, \emptyset, \sigma_l) \\ &\leq \lim_{l \to \infty} \hat{F}_i(L^U) \\ &= \hat{F}_i(L^U) \end{aligned} \tag{7.7.78}$$

As $\hat{F}_i(L) = \hat{F}_i(L, \emptyset)$, we conclude

$$\frac{\partial}{\partial \lambda_n} \hat{F}_i(L) \leq \hat{F}_i(L^U) \tag{7.7.79}$$

Case 2 Now assume $\hat{F}_i(L) = \hat{F}_i(L, M)$ for some $M \neq \emptyset$. Then, with $L' = L_{\neg M}$ being the residual lattice instance,

$$\hat{F}_i(L) = \hat{F}_i(L', \emptyset) + \text{payoff}\,(L, M) \tag{7.7.80}$$

as shown before in (7.7.68). If $n \notin M$ we apply the proposition for $\hat{F}_i(L', \emptyset)$ with smaller lattice instance L'. If $n \in M$ we take from (7.7.75) that

$$\begin{aligned} \hat{F}_i(L^U) &\geq X_n \\ &= \frac{\partial}{\partial \lambda_n} \text{payoff}\,(L, M) \\ &= \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, M) \end{aligned} \tag{7.7.81}$$

This completes the proof. □

The lower bounds for the partial derivatives of the best worst-case prices are weaker:

Proposition 7.19 (Lower bounds for partial derivatives). *Let L be a lattice instance with signature $(\mathbf{X}, \lambda)$. Assume the partial derivatives of the best worst-case price process $\hat{F}$ in λ exist and are uniformly continuous. Pick $n \in \{1, \ldots |L|\}$. Let L_D^E be the lattice instance with signature $(X_n^E, -1)$, where X_n^E denotes the European version of X_n (see Def. 7.3). Let $\hat{F}^E$ be the best worst-case price process for $(X_n^E, -1)$. Then*

$$-\hat{F}_i^E(L_D^E) \leq \frac{\partial}{\partial \lambda_n} \hat{F}_i(L) \tag{7.7.82}$$

Proof. By induction over i. At maturity ($i = N$), all instruments are exercised, and equality holds in (7.7.82). Therefore assume $i < N$.

Case 1 Assume that $\hat{F}_i(L) = \hat{F}_i(L, \emptyset)$, i.e. no instruments are exercised in the best worst-case. Fix σ. Then, by uniform continuity and the induction hypothesis,

$$\begin{aligned}
\frac{\partial}{\partial \lambda_n} F_i(L, \emptyset, \sigma) &= \frac{\partial}{\partial \lambda_n} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}(L) \mid \mathcal{F}_i \right) \\
&= \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \frac{\partial}{\partial \lambda_n} \hat{F}_{i+1}(L) \mid \mathcal{F}_i \right) \\
&\geq \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \left(-\hat{F}_{i+1}^E(L_D^E) \right) \mid \mathcal{F}_i \right) \\
&= -\frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}^E(L_D^E) \mid \mathcal{F}_i \right)
\end{aligned} \tag{7.7.83}$$

Furthermore

$$\begin{aligned}
-\hat{F}_i^E(L_D^E) &= -\sup_{\sigma' \in \mathcal{C}} \frac{1}{\beta_i} \mathrm{E}_{Q(\sigma')} \left(\beta_{i+1} \hat{F}_{i+1}^E(L_D^E) \mid \mathcal{F}_i \right) \\
&= \inf_{\sigma' \in \mathcal{C}} \left[-\frac{1}{\beta_i} \mathrm{E}_{Q(\sigma')} \left(\beta_{i+1} \hat{F}_{i+1}^E(L_D^E) \mid \mathcal{F}_i \right) \right] \\
&\leq -\frac{1}{\beta_i} \mathrm{E}_{Q(\sigma)} \left(\beta_{i+1} \hat{F}_{i+1}^E(L_D^E) \mid \mathcal{F}_i \right)
\end{aligned} \tag{7.7.84}$$

Together, (7.7.83) and (7.7.84) show that, for every $\sigma \in \mathcal{C}$,

$$\frac{\partial}{\partial \lambda_n} F_i(L, \emptyset, \sigma) \geq \hat{F}_i^E(L_D^E) \tag{7.7.85}$$

Let $\sigma_1, \sigma_2, \ldots$ be a sequence such that

$$\lim_{l \to \infty} F_i(L, \emptyset, \sigma_l) = \hat{F}_i(L, \emptyset) \tag{7.7.86}$$

Then,

$$\begin{aligned} \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, \emptyset) &= \frac{\partial}{\partial \lambda_n} \lim_{l \to \infty} F_i(L, \emptyset, \sigma_l) \\ &= \lim_{l \to \infty} \frac{\partial}{\partial \lambda_n} F_i(L, \emptyset, \sigma_l) \\ &\geq \lim_{l \to \infty} \hat{F}_i^E(L_D^E) \\ &= \hat{F}_i^E(L_D^E) \end{aligned} \tag{7.7.87}$$

Finally,

$$\frac{\partial}{\partial \lambda_n} \hat{F}_i(L) \geq \hat{F}_i^E(L_D^E) \tag{7.7.88}$$

Case 2 The case $\hat{F}_i(L) = \hat{F}_i(L, M)$ for some $M \neq \emptyset$ is handled just as case 2 in the proof of Prop. 7.18. □

Propositions 7.18 and 7.19 show that $\frac{\partial}{\partial \lambda_n} \hat{F}_i(L)$ lies in the interval

$$\hat{F}_i^E(L_D^E) \leq \frac{\partial}{\partial \lambda_n} \hat{F}_i(L) \leq \hat{F}_i(L^U) \tag{7.7.89}$$

for X_n. This band is wider than the corridor of uncertainty $\left[\hat{F}_i(L_D), \hat{F}_i(L^U)\right]$. We were unable to prove that the lower corridor bound is also a lower bound for the partial derivative (which we conjecture nevertheless).

Of course, $\frac{\partial}{\partial \lambda_n} \hat{F}_i(L)$ is not available in a program unless all early exercise combinations have already been examined. This can be avoided by substituting $\frac{\partial}{\partial \lambda_n} \hat{F}_i(L, \emptyset)$ for $\frac{\partial}{\partial \lambda_n} \hat{F}_i(L)$. Although not shown here, the previous results can be extended to (and partially include already) the estimate

$$\hat{F}_i^E(L_D^E, \emptyset) \leq \frac{\partial}{\partial \lambda_n} \hat{F}_i(L, \emptyset) \leq \hat{F}_i(L^U, \emptyset) \tag{7.7.90}$$

Figure 7.9 instantiates the algorithm of Fig. 7.6 to collapse the corridor of uncertainty.

2∗∗ Set

$$A_C = \{n \in \operatorname{long}(\mathbf{X}, \lambda) \mid \hat{v}_n(j, i; L, \emptyset) > X_n\}$$
$$A_F = \operatorname{long}(\mathbf{X}, \lambda) \setminus A_C$$

and

$$B_C = \{n \in \operatorname{short}(\mathbf{X}, \lambda) \mid \hat{v}_n(j, i; L, \emptyset) > X_n\}$$
$$B_F = \operatorname{short}(\mathbf{X}, \lambda) \setminus B_C$$

Figure 7.9: An elaboration of step 2∗ in Fig. 7.6 that uses partial derivatives to estimate the early exercise boundary. The variables $\hat{v}_n(j, i; L, \emptyset)$ have already been computed

7.2.3 Other Issues

Sections 7.2.1 and 7.2.2 have presented the big picture. In this section we review some minor or unresolved issues which are interesting purely from a computational point of view. They are of no financial or numerical concern.

Dynamic Maintenance of the Corridor of Uncertainty

The algorithm in Fig. 7.7 relies of the existence of $\hat{V}(j, i; L_n^U)$ and $-\hat{V}(j, i; L_D^n)$ to partition $\operatorname{long}(\mathbf{X}, \lambda)$ respectively $\operatorname{short}(\mathbf{X}, \lambda)$ into A_C, A_F, A_M respectively B_C, B_F, B_M. Here, L_n^U is the lattice instance with signature $(X_n, 1)$, and L_D^n is the lattice instance with signature $(X_n, -1)$.

Depending on the shape of the lattice (box or tree shape?), its position (what is s_0?), the width of the volatility range and the characteristics of the instruments, A_C or A_F respectively B_C or B_F may sometimes be empty, corresponding to the respective boundaries lying outside the region covered by the lattice.

Thus L_n^U or L_D^n may sometimes be superfluous. In order to not maintain lattice instances which are of no use, we employ the dynamic lookup approach that reduces the number of lattice instances carrying partial portfolios in the first place. The recursion that adds lattice instances when needed is activated in step 3∗a in the algorithm in Fig. 7.6.

The idea is to use the partial derivative of $\hat{V}(j, i; L, \emptyset)$ with respect to λ_n to make a first choice. If $X_n < \hat{v}_n(j, i; L)$ then $\hat{V}(j, i; L_n^U) \leq X_n$ cannot be the case, due to Prop. 7.18 that says, in its discrete approximation, $\hat{v}_n(j, i; L) \leq \hat{V}(j, i; L_n^U)$. This necessarily implies $x \notin A_F \cup B_F$. Thus, lookup of L_n^U can be avoided in some cases. As $\hat{v}_n(j, i; L)$ is not yet available when the comparison needs to be made, $\hat{v}_n(j, i; L, \emptyset)$ may be used instead.

In the other direction the situation is more subtle. Prop. 7.19 only states that $-\hat{F}_i^E(L_D^E) \leq \frac{\partial}{\partial \lambda_n} \hat{F}_i(L)$ which is too weak to allow conclusions from $X_n > \hat{v}_n(j, i; L)$. However, under the conjecture $-\hat{F}_i(L_D) \leq \frac{\partial}{\partial \lambda_n} \hat{F}_i(L)$, the initial comparison with the partial derivative may indeed lead to the avoidance of the L_D^n lookup for some X_n. This strategy is pursued in our implementation.

A careful look at the data in Fig. 7.12 reveals that this avoidance strategy has practical impact. The number of lattice instances reported in the table are smaller than the ones that follow from the schematic view in Fig. 7.14, for $\sigma_{\max} \leq 0.4$.

Recursion Leads to Domino Effect

There is also the possibility of the recursion in step 3∗a in Fig. 7.6 causing a domino effect that restarts the rollback of the time slices in the finite difference scheme for many lattice instances. If $\hat{V}(j, i; L, A_1 \cup B_1 \cup A_E \cup B_E)$ is not available in the computation of $\hat{V}(j, i; L)$ for some node instance $(j, i; L)$, then lattice instance L_1 with signature $(\text{select}(\mathbf{X}, M_1), \text{select}(\lambda, M_1))$ needs to be created. Here, $M_1 = \{1, \ldots, |L|\} \setminus \{A_1 \cup B_1 \cup A_E \cup B_E\}$. The finite difference scheme computes $\hat{V}(\cdot, i'; L_1)$ for all $N \geq i' \geq i$ and resumes the computation of $\hat{V}(j, i; L)$.

A memory aware implementation of the finite difference scheme does not keep all the values of $\hat{V}(\cdot, i'; L_1)$, and the other lattice instances. Rather, data is kept for the current and the previous time slices i' and $i' + 1$, in order to reduce the space complexity for one lattice instance from $O(N \times (U - D))$ to $O(U - D)$. Here, D and U are the spatial levels of the lattice boundaries.

For this reason, subordinate values $\hat{V}(\cdot, i'; L_1, A_2 \cup B_2 \cup A_E \cup B_E)$ required in turn as data for L_1 need not be available, even if the associated lattice instance L_2 exists. Each lattice instance is equipped to provide data for one "current" time slice, and no others.

We have briefly mentioned in the beginning of Sect. 7.2 that the algorithms for American options are applicable to barrier options with regular or irregular barriers as well.

B	BC	AB	ABC
			30 ✓, 29 recurse for BC
	30,29 ✓		resume 29
	28–20 ✓		28–20 ✓
	19 recurse for B		
30–19 ✓	resume 191		19 recurse for AB
		$\boxed{\text{30 ✓, 29 recurse for } B}$	
$\boxed{\text{30,29 ✓}}$		resume 29	
28–19 ✓		28–19 ✓	resume 19
18–0 ✓	18–0 ✓	18–0 ✓	18–0 ✓

Figure 7.10: Rolling back the lattice for a 30-day up-and-out barrier option A, a 25-day vanilla option B and a 20-day down-and-out barrier option C. Numbers indicate time slices i for which values $\hat{V}(\cdot, i, L)$ are being computed, and labels indicate actions triggered due to lookup misses. ("19 recurse for AB", for instance, means that the lattice instance for portfolio AB does not exist or cannot provide the data for the desired time slice $i = 19$.) The computation proceeds row by row, and within rows from left to right columns. The boxes represent the single case in which the creation of a new lattice instance leads to a waste of compute time

The tools developed in Chapter 6, in particular the algorithm in Fig. 6.4 to compute the extension hierarchy, go beyond the general approach of Fig. 7.6 in that they guarantee that $\hat{V}(j, i; L, A_E \cup B_E)$ is always available if the barriers are canonical.

For illustration purposes, we assume that the computation of the extension hierarchy is turned off in the following example shown in Fig. 7.10. The portfolio consists of a 30-day up-and-out barrier option A, a 25-day vanilla option B and a 20-day down-and-out barrier option C. The time step is one day: $dt = 1/365$.

Figure 7.10 monitors the finite difference scheme time slice by time slice. "Recurse" and "resume" labels indicate where recursion is triggered and work is resumed, and for what time slice. Initially, only the lattice instance for the entire portfolio is maintained. The boxed cells represent a situation in which a total restart is required: the lattice instance $L(B)$ for B is carried unimpeded through time slices 30, … ,19, when the creation of the lattice instance for AB requires access to time 30-values on $L(B)$. These have been discarded long ago, and so $L(B)$ has to be restarted, resulting in double work for 12 time slices on $L(B)$.

In general, if the directed acyclic graph implied by lookup operations, where vertices model lattices instances and edges model data flow, is recombining, then the domino effect may occur. Edges in the dag are created at different times and may connect to vertices whose lattice instances are not synchronized. Singleton partial portfolios are likely to lead to the domino effect, for instance..

The domino effect can have serious consequences for the running time. There are two solutions to this problem:

1. Develop a tool that precomputes an anlogue of the extension hierarchy for American options. It is in principle possible to evaluate all singleton portfolios first and create a data structure with geometric information on overlapping corridors of uncertainty. The resulting extension hierarchy would be exact. This is a preemptive solution.

2. Periodically checkpoint by saving the values $\hat{V}(\cdot, i; L)$ and related information such as the gradient in separate memory space. A restart can then be based on the data collected during the most recent checkpoint. This solution tries to alleviate the effect of a restart.

Neither approach has been implemented in our system, however. Although the domino effect plays no role in our laboratory test cases, we realize that an industrial-strength product must implement at least one to yield competitive results, as far as speed is concerned.

Intermediate Results in the Minmax Computation

Step 3∗b in Fig. 7.6 requires the computation of

$$V(L) = \min_{A \subseteq A_U} \max_{B \subseteq B_U} \hat{V}(j, i; L, A \cup B \cup A_E \cup B_E) \tag{7.7.91}$$

where the existence of $\hat{V}(j,i;L,A\cup B\cup A_E\cup B_E)$ is guaranteed by step 3$*$a. Taking the minmax term apart, we observe

$$V(L)=\min\Bigg\{\max\Big[\hat{V}(j,i;L,A_E\cup B_E),\max_{\substack{B\subseteq B_U\\ B\neq\emptyset}}\hat{V}(j,i;L,B\cup A_E\cup B_E)\Big],\\ \min_{\substack{A\subseteq A_U\\ A\neq\emptyset}}\max_{B\subseteq B_U}\hat{V}(j,i;L,A\cup B\cup A_E\cup B_E)\Bigg\} \tag{7.7.92}$$

Furthermore,

$$\begin{aligned}&\max_{\substack{B\subseteq B_U\\ B\neq\emptyset}}\hat{V}(j,i;L,B\cup A_E\cup B_E)\\ &\quad=\max_{n\in B_U}\max_{B\subseteq B_U\setminus\{n\}}\hat{V}(j,i;L,B\cup A_E\cup B_E\cup\{n\})\\ &\quad=\max_{n\in B_U}\left(\max_{B\subseteq B_U\setminus\{n\}}\hat{V}(j,i;L_n,B\cup A_E\cup B_E)+\lambda_nX_n\right)\end{aligned} \tag{7.7.93}$$

where the signature of L_n is

$$\big(\text{select}\left(\mathbf{X},\{1,\ldots,n-1,n+1,\ldots,k\}\right),\text{select}\left(\lambda,\{1,\ldots,n-1,n+1,\ldots,k\}\right)\big)$$

Here we assume the computation of A_E, B_E, A_U and B_U is independent of the lattice instance: switching to lattice instance L_n must not change these sets, apart from $\{n\}$. In the algorithms presented so far this is indeed so.

Now consider A_U:

$$\begin{aligned}&\min_{\substack{A\subseteq A_U\\ A\neq\emptyset}}\max_{B\subseteq B_U}\hat{V}(j,i;L,A\cup B\cup A_E\cup B_E)\\ &\quad=\min_{m\in A_U}\min_{A\subseteq A_U\setminus\{m\}}\max_{B\subseteq B_U}\hat{V}(j,i;L,A\cup B\cup A_E\cup B_E\cup\{m\})\\ &\quad=\min_{m\in A_U}\left(\hat{V}(j,i;L_m)+\lambda_mX_m\right)\end{aligned} \tag{7.7.94}$$

Together,

$$V(L)=\min\Bigg\{\max_{n\in B_U}\left(\max_{B\subseteq B_U\setminus\{n\}}\hat{V}(j,i;L_n,B\cup A_E\cup B_E)+\lambda_nX_n\right),\\ \min_{m\in A_U}\left(\hat{V}(j,i;L_m)+\lambda_mX_m\right)\Bigg\} \tag{7.7.95}$$

Thanks to step 3∗a, $\hat{V}(j,i;L_m)$ is avaliable. It is reasonable to assume that the values $\hat{V}(j,i;L_n,B \cup A_E \cup B_E)$ are available as well; they can be readily stored as intermediate results when L_n is processed. Equally accessible (and computed only once) is $\max_{B \subseteq B_U \setminus \{n\}} \hat{V}(j,i;L_n,B \cup A_E \cup B_E)$, which can be constructed "bottom-up" when the hierarchy of lattice instances is processed, and therefore need not involve all exponentially many combinations.

With that in mind (7.7.95) is equivalent to (7.7.91), but with only $|A_U| + |B_U|$ terms instead of $2^{|A_U|+|B_U|}$. The complexity of step 3∗b (and likewise of step 3∗a) can thus be reduced dramatically, at the cost of additional storage of intermediate results. Figure 7.11 gives an idea of the savings for the case $|A_U| = |B_U| = 3$.

	B_U							
A_U	ccc	ccE	cEc	Ecc	cEE	EcE	EEc	EEE
ccc	max	max	max	max	max			
ccE	min							
cEc	min							
Ecc	min							
cEE								
EcE								
EEc								
EEE								

Figure 7.11: The table shows all $2^{3+3} = 64$ early exercise combinations if $|A_U| = |B_U| = 3$. Rows represent combinations selected from A_U, and columns represent combinations picked from B_U. A lower-case "c" means <u>c</u>ontinuation or no exercise, an upper-case "E" means <u>E</u>xercise. The formula in (7.7.95) examines the filled in table elements only. "max" indicates a subterm of (7.7.93). "min" indicates a subterm of (7.7.94) (the column position is slightly misleading in this case, as the correct selection from B_U might differ; it is, however, already reflected in $\hat{V}(j,i;L_m)$)

It is important to keep in mind, however, that the overall number of lattice instances is not reduced by this algebraic trick. Again, think of lattice instances as vertices and associations through lookup as directed edges. The result of the algebraic transformation is to reduce the number of edges, but the set of vertices remains unaltered.

In an early stage of our research, (7.7.91) was replaced by (7.7.95). Although no rigorous tests have been made, the speedup appeared to be marginal if the number of instruments was very small, but noticeable once the number of instruments increased. Since there was no obvious drawback and the additional overhead in memory management seemed to be outweighed by the benefit in all cases, (7.7.95) has been used ever since.

7.3 Performance Results

All tests were performed on a Pentium/166 MHz PC running Windows NT Workstation 4.0/SP 3 and equipped with 128 MB of RAM. The best worst-case pricer, which we call Mtg in the following, is written and compiled with Microsoft Visual C++ 5.0, optimizations activated.

7.3.1 Complexity

We investigate the computational complexity that arises from the positive correlation between the width of the volatility band and the number of lattice instances required for the solution of the best worst-case pricing problem. In this section, scenarios and portfolios are constructed unter "lab conditions", to probe certain performance characteristics while perturbing the setup as little as possible.

Statistical tests on a large number of generated situations are reported in Sect. 7.3.2.

Experiment 1: Three American Puts

The portfolio consists of three 30-day American puts with strikes 90, 100 and 110, respectively. Market parameters are $S_0 = 100$ and $r = 0.03$. The size of the time step is $dt = 1/(5 \times 365)$, or five periods per day, 150 periods overall. All results are obtained with the explicit method.

Figure 7.12 gives an overview of the running time when the corridors of uncertainty are maintained (as described in Sect. 7.2.1) and collapsed (as described in Sect. 7.2.2), respectively. In the linear case ($\sigma_{\min} = \sigma_{\max}$) Mtg takes 0.4 seconds to compute the result. If corridors are maintained, the running time is stable in the intervals $0.12 \le \sigma_{\max} \le 0.24$

(small $\sigma_{\max}$), $0.26 \leq \sigma_{\max} \leq 0.40$ (medium $\sigma_{\max}$) and $0.42 \leq \sigma_{\max} \leq 0.5$ (large $\sigma_{\max}$), with jumps of about 1.5, 1 and 1.5 seconds preceding the intervals. Jumps correspond to the introduction of more lattice instances, as wider volatility bands lead to more overlap among corridors of uncertainty.

	maintaining		collapsing
$\sigma_{\max}$	time [s]	# of lattices	time [s]
0.10	0.4	1	–
0.12	1.9	6	1.0
...			
0.24	1.9	6	1.0
0.26	2.9	7	1.0
0.28	3.0	8	1.0
...			
0.40	2.7	8	1.1
0.42	4.2	10	1.0
...			
0.50	3.9	10	1.0

Figure 7.12: Results for a portfolio of three American 30-day puts with strikes 90, 100 and 110, evaluated under a volatility band of $[0.1, \sigma_{\max}]$, with $\sigma_{\max}$ ranging from 0.1 to 0.5 in steps of 0.02. Shown is the running time if corridors of uncertainty are maintained, together with the number of lattice instances created (of those, up to 6 lattice instances are used to monitor the corridors of uncertainty). Also shown is the running time if corridors of uncertainty are collapsed

Figure 7.13 contains a graph of the running times. Figure 7.14 displays the location and extent of the corridors of uncertainty for the three puts schematically for the qualitatively different small, medium and large volatility ranges. For medium and large $\sigma_{\max}$, the labels in the picture indicate the non-singleton residual portfolios that are part of the early exercise combinations considered. In the medium scenario, 3 partial non-singleton portfolios need to be maintained. In the large scenario, 4 partial non-singleton portfolios need to be maintained. In addition, some singleton lattice instances may need to be

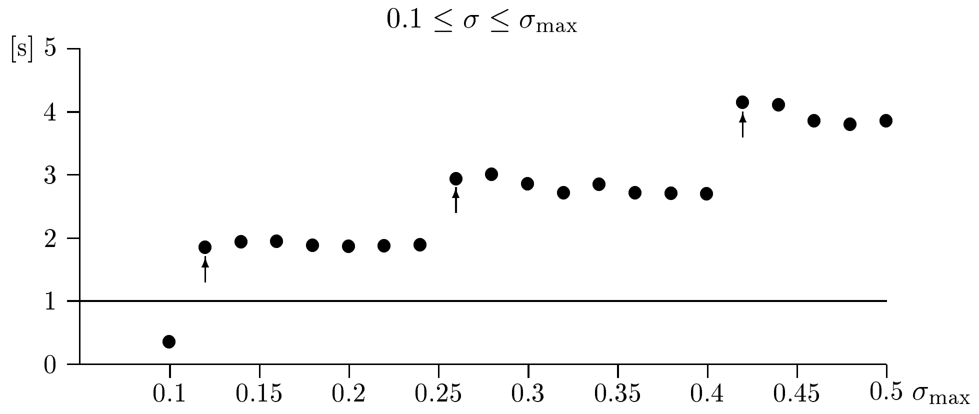


Figure 7.13: Running times in seconds for the three 30-day American puts with strikes 90, 100 and 110, respectively, evaluated under different volatility ranges $[0.1, \sigma_{\max}]$, $0.1 \leq \sigma_{\max} \leq 0.5$. The arrows mark jumps in the number of lattice instances, due to increasing overlap of the corridors of uncertainty. The line at 1 second indicates the average running time if the corridors of uncertainty are collapsed

maintained to feed boundary values, and some to provide the upper or lower boundary for the corridor of uncertainty.

Also compare with Fig. 7.8, which shows the actual corridor shape for some sample values of $\sigma_{\max}$. Notice that the number of combinations cited there, 4 respectively 8, refers to the maximum number of early exercise combinations that enter the minmax term at any given node instance. This number is a lower bound for the overall number of lattice instances.

The prices computed by both methods (maintaining versus collapsing) were identical. Collapsing the corridors of uncertainty, however, turns out to reduce the running time considerably and to make it independent from $\sigma_{\max}$. The speedup is approximately $3.9/1.0 \approx 4$ for $\sigma_{\max} = 0.5$.

Experiment 2: Increasing the Number of Puts

In the previous experiment the size of the portfolio remained stable, while the width of the volatility range increased. In experiment 2 the number of puts in the portfolio is

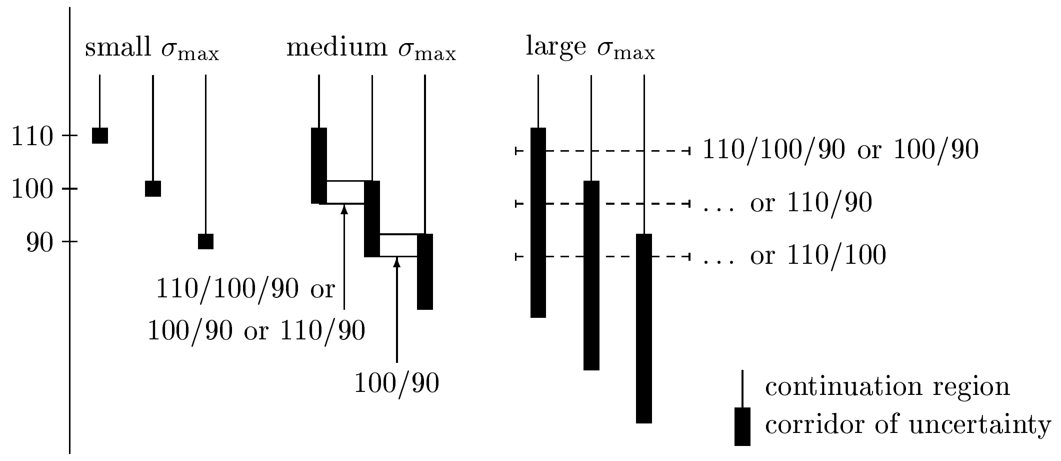


Figure 7.14: A schematic view of the extent of the corridors of uncertainty for the three American puts mentioned in the text, under three qualitatively different volatility ranges. The vertical axis marks the value of the underlying. The labels indicate the residual non-singleton portfolios (at least 2 puts); some singleton residual portfolios are evaluated in addition to maintain the corridors of uncertainty. Instruments are identified by their strike, which is 110, 100 and 90, respectively

varied. All puts mature in 30 days and differ only by their strikes. The extremal strikes are 80 and 120, and all other strikes are equidistantly spaced between those endpoints. Thus, a portfolio of size 5 contains the strikes 80, 90, 100, 110 and 120.

The number of puts varies between 2 (strikes 80 and 120 only) and 21 (10 strikes below 100, 10 strikes above 100, and 100 itself). The experiment is repeated for a linear scenario ($\sigma = 0.1$) and two nonlinear scenarios, with $\sigma_{\min} = 0.1$ and $\sigma_{\max} = 0.125$ and 0.15, respectively. All other parameters are unchanged: $S_0 = 100$, $r = 0.03$ and $dt = 1/(5 \times 365)$.

Figure 7.15 shows the running times and number of lattice instances created for all three volatility scenarios and all 20 portfolio sizes. Figure 7.16 presents the same data pictorially. The superimposed step function shows the relative increase in the number of lattice instances, normalized to fit into the plot. This function approximately follows the trend in the running times.

	$\sigma = 0.1$	$0.1 \leq \sigma \leq 0.125$		$0.1 \leq \sigma \leq 0.15$	
# of puts	time [s]	time [s]	# of lattices	time [s]	# of lattices
2	0.28	0.8	3	1.2	4
3	0.37	1.6	5	2.1	6
4	0.54	2.8	7	3.5	8
5	0.59	3.9	9	4.6	10
6	0.72	5.4	11	6.5	12
7	0.83	7.0	13	8.2	14
8	0.91	9.1	15	10.0	16
9	1.04	10.4	17	12.2	18
10	1.13	12.5	18	14.5	20
11	1.29	15.4	20	17.2	22
12	1.47	18.1	22	19.7	24
13	1.53	22.4	24	38.9	27
14	1.67	27.6	26	115.9	34
15	1.79	28.8	28	134.9	38
16	1.89	33.6	30	168.7	43
17	1.93	38.4	32	172.9	47
18	2.37	41.7	33	208.2	50
19	2.40	80.5	36	211.7	53
20	2.26	88.8	38	239.5	56
21	2.37	89.9	40	258.9	58

Figure 7.15: Running times in seconds for portfolios with a varying number of 30-day American puts under three volatility scenarios. For the nonlinear scenarios, also shown is the number of lattice instances necessary to compute the best worst-case price

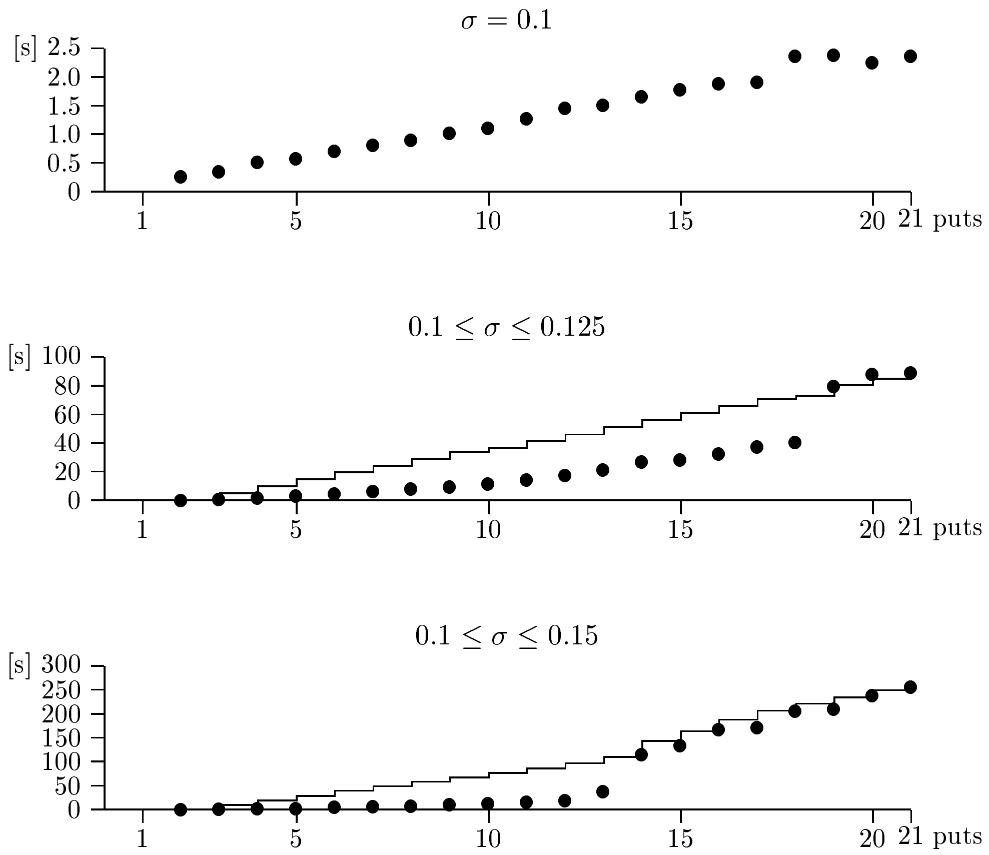


Figure 7.16: The data in Fig. 7.15 presented graphically. Solid disks mark running times. The step function in the bottom graphs tracks the relative (!) growth of the number of lattice instances created. In the top graph there is only one lattice instance

Notice that the running time is not an absolute function of the number of lattice instances. In the scenario $\sigma_{\max} = 0.125$, for instance, 38 lattice instances are processed in 88.8 seconds (20 puts), while in the scenario $\sigma_{\max} = 0.15$, 38 lattice instances are processed in 134.9 seconds (15 puts). The individal width of the corridors of uncertainty, positively correlated with $\sigma_{\max}$, plays a significant role, too. It is here where the early exercise combinations are weighed, causing computational overhead.

The most significant result of this test is the validity of the concept of corridors of uncertainty: there are 2 21 theoretical early exercise combinations if the portfolio contains 21 puts.

7.3.2 A Mass Test

The experiments in the previous section were conducted under laboratory conditions: all parameters but one remained frozen so as to test the influence of the selected parameter on the running time of Mtg. The dimensions along which tests were made were the width of the volatility band and the density of strikes.

In this section we lift the ceteribus paribus condition, and compute best worst-case prices for a set of portfolios with divergent characteristics. Statistical measures are then used to judge performance and accuracy. The hardware is unchanged: a Pentium/166 MHz PC running Windows NT Workstation 4.0/SP 3 and equipped with 128 MB of RAM. Mtg, the pricer, is written in C++.

The results reported here are published in Buff (1999a).

The Random Portfolio Space

The random portfolio space consists of 200 portfolios. Each portfolio consists of 8 options with characteristics determined randomly as follows:

- With equal probability the option is a call or a put.
- For call options, the strike lies in the interval $[80, \ldots, 110]$ with probability

$$\Pr\{\text{strike} = x\} = \begin{cases} \dfrac{1}{42} & \text{if } 80 \leq x \leq 100 \\ \dfrac{1}{20} & \text{if } 101 \leq x \leq 110 \end{cases} \qquad (7.7.96)$$

For put options, the strike lies in the interval $[90, \dots, 120]$ with probability

$$\Pr\{\text{strike} = x\} = \begin{cases} \dfrac{1}{20} & \text{if } 90 \leq x \leq 99 \\ \dfrac{1}{42} & \text{if } 100 \leq x \leq 120 \end{cases} \tag{7.7.97}$$

Thus, in-the-money and out-of-the-money options are equally likely, but the range of possible strikes for in-the-money options is about twice as wide as for out-of-the-money options.

- The maturity is uniformly distributed in the interval $[50, \dots, 100]$, counted in days.
- The position λ_n, $1 \leq n \leq 8$, is ± 1, ± 2 or ± 3 with equal probability.

Figure 7.17 gives a summary.

type	probability	strike	maturity	position
call	$\frac{1}{2}$	betw. 80 and 110	betw. 50 and 100	± 1, ± 2 or ± 3
put	$\frac{1}{2}$	betw. 90 and 120	betw. 50 and 100	± 1, ± 2 or ± 3

Figure 7.17: The random portfolio space. Each option has the characteristics listed in the table, randomly selected. In addition, options are either European or American

The random portfolio space is furthermore divided into two subsets:

- For the first 100 portfolios, 4 options are American. The remaining 4 options are European. We refer to this subset as the 4/4 set of portfolios.
- For the last 100 portfolios, 5 options are American. The remaining 3 options are European. We refer to this subset as the 5/3 set of portfolios.

The random portfolio space extends the theoretical framework observed so far in two aspects:

1. Maturity dates differ; and
2. there are American and European instruments in each portfolio.

These “advanced” features are incorporated to simulate actual situations better. Both features are straightforward to add to the theoretical base. The 4 respectively 3 European options contribute to curvature through superposition of their vanilla payoff structures.

The Evaluation Space

Each of the portfolios $(\mathbf{X}_1^{44}, \lambda_1^{44})$, $\ldots$, $(\mathbf{X}_{100}^{44}, \lambda_{100}^{44})$ in the 4/4 set, and $(\mathbf{X}_1^{53}, \lambda_1^{53})$, $\ldots$, $(\mathbf{X}_{100}^{53}, \lambda_{100}^{53})$ in the 5/3 set was evaluated several times under varying conditions. Always, however, $\sigma_{\text{min}} = 0.1$. The market parameters are $S_0 = 100$, interest rate $r = 0.05$ and dividend rate $q = 0.03$ throughout. (A dividend rate is introduced since the portfolios contain American calls which aren't exercised early if $q = 0$.)

Experiment 1 In the first experiment, all portfolios as well as their negative versions ($-\lambda$ instead of λ) were evaluated under the three volatility scenarios $\sigma_{\text{max}} = 0.2$, 0.4 and 0.6, respectively. Two series of evaluations were performed with maintained corridors of uncertainty, and two series of evaluations were performed with collapsed corridors of uncertainty. The time step in series 1 was $dt = 1/365$, under both methods; in series 2 it was $dt = 1/(2 \times 365)$. This experiment required $2 \times 2 \times 3 \times (2 \times 100 + 2 \times 100) = 4800$ evaluations.

Experiment 2 In the second experiment, all portfolios as well as their negative versions ($-\lambda$ instead of λ) were evaluated under the volatility scenario $\sigma_{\text{max}} = 0.8$. Again, two series of evaluations were performed with maintained corridors of uncertainty, and two series of evaluations were performed with collapsed corridors of uncertainty. The time step in series 1 was $dt = 1/(5 \times 365)$, under both methods; in series 2 it was $dt = 1/(10 \times 365)$. This experiment required $2 \times 2 \times (2 \times 100 + 2 \times 100) = 1600$ evaluations.

The reason for using 5 and 10 as opposed to 1 and 2 time steps per day is stability: the algorithm in Fig. 5.4 rejects 1 respectively 2 time steps per day as too coarse for $\sigma_{\text{max}} = 0.8$.

Experiment 3 In the third experiment, all portfolios as well as their negative versions ($-\lambda$ instead of λ) were evaluated under the volatility scenario $\sigma_{\text{max}} = 1.0$. One series of evaluations was performed with maintained corridors of uncertainty, and one series of evaluations was performed with collapsed corridors of uncertainty. The time step was set to $dt = 1/(16 \times 365)$, after running the algorithm in Fig. 5.4. This experiment required $2 \times 2 \times 100 + 2 \times 100 = 800$ evaluations.

Figure 7.18 provides an overview over all three experiment specifications.

	volatility	series 1	series 2
Experiment 1	$0.1 \leq \sigma \leq 0.2$	$dt = 1/365$	$dt = 1/(2 \times 365)$
	$0.1 \leq \sigma \leq 0.4$	$dt = 1/365$	$dt = 1/(2 \times 365)$
	$0.1 \leq \sigma \leq 0.6$	$dt = 1/365$	$dt = 1/(2 \times 365)$
Experiment 2	$0.1 \leq \sigma \leq 0.8$	$dt = 1/(5 \times 365)$	$dt = 1/(10 \times 365)$
Experiment 3	$0.1 \leq \sigma \leq 1.0$	$dt = 1/(16 \times 365)$	n/a

Figure 7.18: The evaluation space. The time steps are chosen to guarantee numerical stability. Altogether, $4800 + 1600 + 800 = 7200$ evaluations were performed

Maintaining the Corridors of Uncertainty

We give absolute results for the exact approach where corridors of uncertainty are maintained as described in Sect. 7.2.1. In a later paragraph, the benefit and drawback of collapsing the corridors is analyzed relative to the absolute values given here.

Figure 7.19 presents the mean and standard deviation of the running time if corridors of uncertainty are maintained. Only series 1 is analyzed in the first two experiments; no data is avaliable for series 2. The maximum running time in experiment 3 was 302 seconds for the 4/4 set, and 1094 seconds for the 5/3 set.

			4/4 set		5/3 set	
experiment	$\sigma_{\max}$	time step	mean	sdev	mean	sdev
1, series 1	0.2	1/365	1.5	1.0	3.4	2.8
1, series 1	0.4	1/365	3.1	1.1	9.5	4.2
1, series 1	0.6	1/365	3.5	1.1	10.7	4.1
2, series 1	0.8	$1/(5 \times 365)$	36.7	8.9	115.4	36.5
3	1.0	$1/(16 \times 365)$	207.8	45.9	662.2	193.1

Figure 7.19: The running time in seconds if corridors of uncertainty are maintained, broken down for the 4/4 set and the 5/3 set of portfolios. Each entry represents $2 \times 100 = 200$ evaluations, as in (original + negative) $\times$ portfolios

Figure 7.20 presents data on convergence with respect to the time step. The results obtained in series 1 and 2 are matched and compared pair-wise. Shown are the first two central moments, in percentage, of

$$\frac{\hat{V}_2(0,0;L_n^{44}) - \hat{V}_1(0,0;L_n^{44})}{\hat{V}_1(0,0;L_n^{44})} \tag{7.7.98}$$

for $1 \leq n \leq 200$, corresponding to 2×100 portfolios (including the negative λ's) in the 4/4 set, and the first two central moments, in percentage, of

$$\frac{\hat{V}_2(0,0;L_n^{53}) - \hat{V}_1(0,0;L_n^{53})}{\hat{V}_1(0,0;L_n^{53})} \tag{7.7.99}$$

for $1 \leq n \leq 200$, corresponding to 2×100 portfolios in the 5/3 set. $\hat{V}_l$ is the best worst-case price observed in series l. L_n^{44} is the lattice instance with signature $(\mathbf{X}_n^{44}, \lambda_n^{44})$ if $n \leq 100$, and with signature $(\mathbf{X}_{n-100}^{44}, -\lambda_{n-100}^{44})$ if $n \geq 101$. L_n^{53} is interpreted in an analogue fashion.

No data is available for experiment 3, since experiment 3 contains only one series of evaluations.

			4/4 set		5/3 set	
experiment	$\sigma_{\max}$	time steps	mean	sdev	mean	sdev
1	0.2	$1/365 \to 1/(2 \times 365)$	0.1	0.6	0.2	1.6
1	0.4	$1/365 \to 1/(2 \times 365)$	0.3	6.0	-0.8	7.5
1	0.6	$1/365 \to 1/(2 \times 365)$	0.0	11.2	-0.2	2.8
2	0.8	$1/(5 \times 365) \to 1/(10 \times 365)$	-0.5	11.7	-0.2	3.9

Figure 7.20: The relative discrepency in percentage for each matched evaluation in series 1 and 2, respectively, of experiments 1 and 2, broken down by volatility band and portfolio set. The number of time steps is doubled between series 1 and 2

Convergence is better for narrow volatility bands. For $\sigma_{\max} = 0.2$ we may expect stability in the first two leading digits, and thus recommend $dt = 1/(2 \times 365)$ as being adequate. On the other hand, there is considerable variance if $\sigma_{\max} \geq 0.4$. This suggests that for wide volatility ranges dt needs to be further reduced to achieve sufficient numerical stability.

Collapsing the Corridors of Uncertainty: Speed

After establishing a base for comparison, we examine the benefit of collapsing corridors of uncertainty. Let m_n^{44} be the running time if corridors of uncertainty are maintained for portfolio n (where $1 \leq n \leq 200$, and portfolios are counted as described above) in the 4/4 set, and let c_n^{44} be the running time if corridors of uncertainty are collapsed. m_n^{53} and c_n^{53} are interpreted accordingly. Figure 7.21 shows mean and standard deviation for the quantities

$$\frac{c_n^{44}}{m_n^{44}} \quad \text{and} \quad \frac{c_n^{53}}{m_n^{52}} \tag{7.7.100}$$

in percentage, for $1 \leq n \leq 200$, broken down by experiment and series, as well as aggregated over all experiments. Figure 7.22 shows the same data pictorially.

			4/4 set		5/3 set	
experiment	$\sigma_{\max}$	time step	mean	sdev	mean	sdev
1, series 1	0.2	1/365	63.5	10.8	55.5	11.7
1, series 1	0.4	1/365	66.1	13.6	55.1	18.2
1, series 1	0.6	1/365	62.5	16.3	52.6	18.2
2, series 1	0.8	$1/(5 \times 365)$	65.1	15.8	54.3	18.7
3	1.0	$1/(16 \times 365)$	71.0	16.5	60.1	19.6
all			65.6	15.0	55.5	17.7

Figure 7.21: Mean and standard deviation in percentage of the relative running time if corridors of uncertainty are collapsed, broken down by volatility band and portfolio subset. The last row is the average over all previous rows. The inverse of the mean would be the average speedup factor

The relative benefit is remarkably uniform for different volatility bands, although the benefit decreases slightly for very high $\sigma_{\max}$. The standard deviation is under 20% throughout. Relative speed increases if portfolios contain more American instruments .

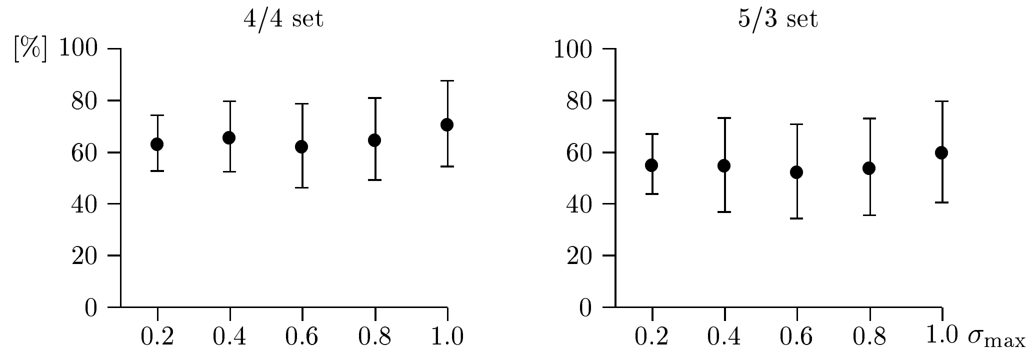


Figure 7.22: Mean ± one standard deviation of the relative running time in percentage when corridors of uncertainty are collapsed, compared to the running time for the benchmark result. The data is the same as in Fig. 7.21

Collapsing the Corridors of Uncertainty: Faithfulness

Collapsing the corridor of uncertainty may lead to false results. The faithfulness of the heuristic measures the gravity of this defect. Let L_n^{44} and L_n^{53} denote lattice instances for portfolios $1 \leq n \leq 200$ in the 4/4 and the 5/3 set, respectively, as defined earlier. Let the benchmark result $\hat{M}(0,0;L_n^{44}) = \hat{V}(0,0;L_n^{44})$ be the best worst-case price on lattice instance L_n^{44} if corridors of uncertainty are maintained, and define $\hat{M}(0,0;L_n^{53})$ accordingly. Let $\hat{C}(0,0;L_n^{44})$ be the best worst-case price if corridors are collapsed, and define $\hat{C}(0,0;L_n^{53})$ accordingly. $\hat{C}(0,0;L_n^{44})$ and $\hat{C}(0,0;L_n^{53})$ may differ from $\hat{V}(0,0;L_n^{44})$ and $\hat{V}(0,0;L_n^{53})$. The faithfulness of the heuristic is reflected in the relative deviation from the benchmark result:

$$\frac{\hat{C}(0,0;L_n^{44}) - \hat{M}(0,0;L_n^{44})}{\hat{M}(0,0;L_n^{44})} \tag{7.7.101}$$

and

$$\frac{\hat{C}(0,0;L_n^{53}) - \hat{M}(0,0;L_n^{53})}{\hat{M}(0,0;L_n^{53})} \tag{7.7.102}$$

Values close to 0 indicate high faithfulness. Large absolute values indicate low faithfulness. Mean and standard deviation in percentage of (7.7.101) and (7.7.102) are shown in Fig. 7.23 for series 2 of experiments 1 and 2. Also shown is the frequency in percentage with which the approximated result deviates no more than 1% from the benchmark result.

			4/4 set			5/3 set		
experiment	$\sigma_{\max}$	time step	mean	sdev	good	mean	sdev	good
1, series 2	0.2	$1/(2 \times 365)$	0.00	0.32	99.0	-0.29	4.06	98.0
1, series 2	0.4	$1/(2 \times 365)$	0.85	7.94	93.0	0.07	2.15	94.0
1, series 2	0.6	$1/(2 \times 365)$	-0.12	9.22	90.5	-0.22	2.44	88.5
2, series 2	0.8	$1/(10 \times 365)$	8.93	122.55	87.0	0.94	18.65	83.0
all			-0.15	3.00	94.2	0.28	6.61	93.5

Figure 7.23: Mean and standard deviation in percentage of the relative deviation from the benchmark result if corridors of uncertainty are collapsed. The column labeled "good" shows the frequency with which the benchmark result is reproduced exactly

Not shown in the figure is the frequency of exactly matching results: 51.0 and 26.0% overall for the 4/4 set, and 46.8 and 19.0% overall for the 5/3 set. Figure 7.24 interpolates the frequency of exactly matching results, or of results that deviate no more than 1 or 5% for experiments 1 and 2. The frequency of "good" results drops consistently as the volatility band gets wider, and slightly if the portfolio contains more American options. Although the heuristic reproduces the benchmark result less than half the time, the frequency at which a 1% relative error bound is achieved is above or close to 90% throughout.

Collapsing the Corridors of Uncertainty: Outliers

There are 4 cases in experiment 1, series 2 in which the absolute deviation from the benchmark result exceeds 50%. The amount by which these cases deviate is shown in Fig. 7.25, together with the composition of one of the outlier portfolios.

The sequence of best worst-case prices for the marked portfolio shows considerable oscillation as the time step decreases, even if the corridors of uncertainty are maintained. Fig. 7.26 shows the values, plotted against the number d of steps per day for both the explicit and mixed explicit/implicit scheme (this is the only case in this chapter on American

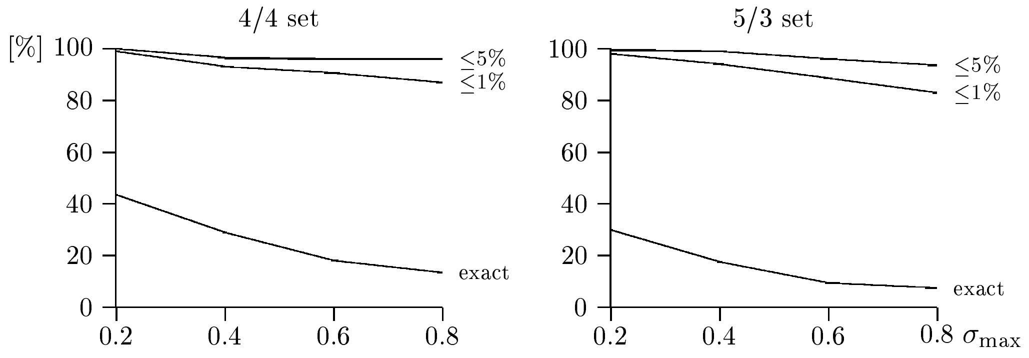


Figure 7.24: Frequency in percentage with which the relative error stays within 0% (exact match), 1% and 5% of the benchmark result if corridors of uncertainty are collapsed, drawn against $\sigma_{\max}$. Data is base on series 2 in experiments 1 and 2

options where Crank-Nicholson combined with iterative refinement is used). 20 lattice instances, the largest possible number, are required to solve the best worst-case pricing problem. The inverse portfolio ($-\lambda_n$ instead of λ_n for $1 \leq n \leq 8$) converges convincingly: the value varies around -36.19, with noise in the fourth digit (values are not shown here). There is no obvious sign that helps to explain what makes the portfolio structurally unusual enough to lead to such instability.

Further comparative convergence analysis with 1, 2, 5 and 20 time steps per day, for all portfolios in the 4/4 subset under the scenario $\sigma_{\max} = 0.6$, shows that there is no correlation between poor numerical convergence and a large deviation from the benchmark result in the series 2 data. The Spearman and Kendall rank coefficients for the association between the absolute relative change of the benchmark result when switching from 1 to 20 time steps per day, and the maximum absolute deviation from the benchmark in the series 2 data are 0.09 and 0.06, respectively. Rank coefficients measure linear and nonlinear monotonic relationships. A value close to zero means there is no such relationship. The linear correlation coefficient is 0.17.

The quality of the result achieved under the heuristic seems therefore unpredictable.

$\sigma_{\max}$	set	deviation [%]
0.4	4/4	107.4
$\rightarrow$ 0.6	4/4	88.1
0.6	4/4	−85.3
0.2	5/3	−57.4

n	exercise	type	maturity	strike	λ_n
1	European	put	100	92	−3
2	European	put	68	97	2
3	European	call	66	93	−1
4	European	put	61	98	1
5	American	put	97	113	−3
6	American	put	93	114	2
7	American	call	68	102	1
8	American	put	57	93	−2

Figure 7.25: Four cases in experiment 1, series 2 (time step $dt = 1/(2 \times 365)$), in which the relative deviation from the benchmark result exceeds 50%, and the composition of one of the outlier portfolios (marked with "$\rightarrow$")

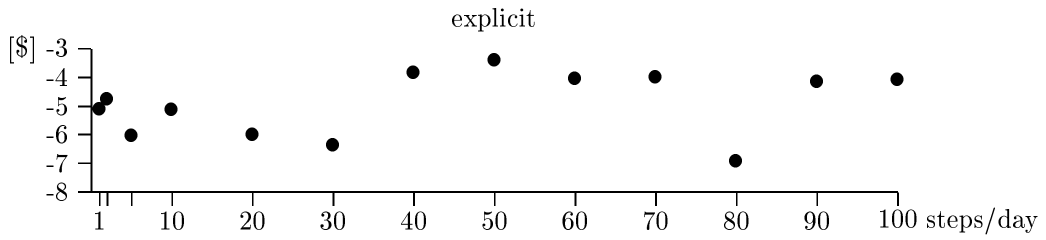
Conclusion

The benefits of collapsing the corridor of uncertainty seem worth the loss of faithfulness if the volatility band is narrow, for then the benchmark results are reproduced to a sufficiently high degree. For $0.1 \leq \sigma \leq 0.2$, for instance, the mean error is zero and the standard deviation of the error is 0.32%, for 4 American options in the portfolio (Fig. 7.23). This is equivalent to 2 matching digits.

The situation becomes less clear as $\sigma_{\max}$ increases. Whether the gain in speed of about 40% is worth the increased chance of missing the best worst-case price by a large amount must be decided case by case. As shown in Fig. 7.24, the 1% or 2 leading digit-threshold is still reached about 90% of the time.

It should be noted that the volatility bands used in the mass test are extremely wide and remain valid over the entire lifetime of the portfolio. In a more realistic setting, the range of uncertainty would be narrower or restricted in time. The next chapter explores volatility scenarios in this direction.

Numerical accuracy at timesteps in the tested range is satisfactory for narrow bands (Fig. 7.20). For wider bands, smaller time steps than those tested should be used in production mode. The use of the more accurate mixed explicit/implicit finite difference



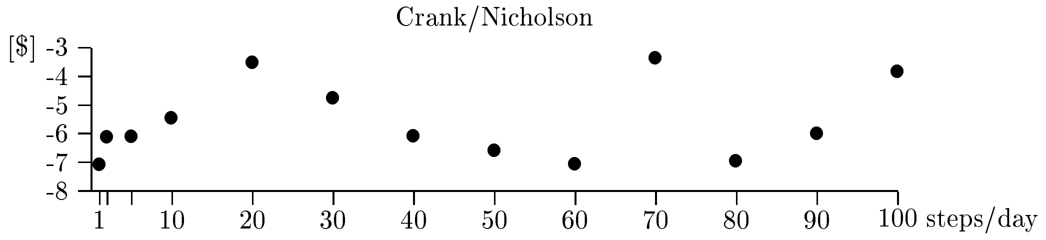


Figure 7.26: High oscillation of the best worst-case value of the portfolio marked in Fig. 7.25 even if the corridors of uncertainty are maintained. There is no qualitative difference between the explicit and Crank-Nicholson scheme. The number of steps per day d is plotted on the x-axis; the corresponding time step would be $dt = 1/(d \times 365)$

scheme would very likely improve the convergence behavior further.

7.4 American Options and Calibration

It is in principle possible to apply the ideas of Sect. 4.2.3 on calibration to portfolios of American options. The calibrated volatility $\hat{\sigma}$ would be path-dependent and not easily convertible into a two-dimensional surface. However, the volatility surface, being the goal of calibration in the first place, should have a format in which subsequent pricing is straightforward. Calibration to American options seems therefore not a viable task.

Optimizing a position in order to find the optimal hedge portfolio under worst-case assumptions, on the other hand, would still be feasible (see Sect. 4.2.2).

8 Exotic Volatility Scenarios

In Chapters 6 and 7, algorithms have been discussed that compute (best) worst-case prices under uncertain volatility scenarios in which $\sigma(S_t, t)$ and $\sigma(S_u, u)$ are independent for $t \neq u$. In this chapter we extend the notion of uncertain volatility scenarios to include evolutions of the spot volatility that depend on its past history.

The non-Markovian character of σ is expressed in by-conditions in the candidate set $\mathcal{C}$. σ no longer depends merely on S_t and t, but on the path ω in the probability space. Replicating the terminology for instruments, we call such volatility scenarios exotic volatility scenarios, as opposed to "conventional" volatility scenarios. In particular, we examine scenarios where the spot volatility can undergo one or several volatility shocks of limited duration.

8.1 Volatility Shocks for Portfolios of Vanilla Options

Volatility shock scenarios are based on the assumption that the spot volatility does not deviate from an estimated prior volatility except possibly when expected or unexpected economic events upset the market for a limited period of time. Such events may be announcements, mergers, court rulings, natural disasters, devaluations, or others. These events have the properties that

- they are difficult to quantize; and, more importantly,
- they cannot be forecasted to happen on a specific day in the future.

We use the worst-case approach for the quantization problem, and multi-lattice dynamic programming for the forecasting problem.

Definition 8.1 (Prior and shock volatility). *Assume we are given volatility values* $0 < \sigma_{\min} \leq \sigma_0 \leq \sigma_{\max}$. *Then* σ_0 *is called the* prior volatility *and expresses the subjective belief of the agent about the true model volatility.* $\sigma_{\min}$ *and* $\sigma_{\max}$ *are lower and upper bounds which the spot volatility can attain during periods of upheaval. They are called the* shock volatility *bounds.*

For simplicity, Def. 8.1 introduces constant volatility parameters. The concepts in this chapter can easily be extended to cover time and/or space dependent prior and shock volatilities. (Recall that this does not mean that the worst-case volatility is also constant!)

Definition 8.2 (Volatility shock scenario). *Assume prior and shock volatility* $0 < \sigma_{\min} \leq \sigma_0 \leq \sigma_{\max}$ *are given. A* volatility shock scenario *is characterized by its* duration $d \geq 1$*, its* periodicity $p \geq 1$ *and its* frequency $f \geq 1$*. The units of* d *and* p *are days;* f *is a dimensionless number. All values are integers.*

The interpretation is as follows: on any realized path ω *the spot volatility will be* σ_0*, except for* f *non-overlapping periods of length* d *days each, during which the spot volatility may fluctuate freely within* $\sigma_{\min}$ *and* $\sigma_{\max}$*. Here, "non-overlapping" refers to the interior of each period; they may touch at their endpoints. In addition, each of these* f *shock periods must start on a day whose day count number is a multiple of* p*, where days are counted from 0.*

The class of volatilities that fulfill this description is denoted by $\mathcal{D}$*.*

The function of p is to reduce the computational overhead and the size of the lattice. We will see below that the compute time is proportional to d/p. p may also be used to time shock periods, but to support this aspect fully a more powerful notion of periodicity may be nessesary. Although in most cases $p \leq d$, we explicitely allow the case $p > d$. The f shock periods are located between time 0 and time N. In the following, we assume $N \geq d + (f-1)\max(p,d)$ for convenience. In other words, the portfolios under investigation last long enough to fall under the influence of at least f shock periods.

Examples of volatility shock scenarios are:

- The prior volatility is $\sigma_0 = 0.15$. However, there will be a 7-day period during which the volatility may oscillate between 0.15 and 1.0. This period, caused by a merger announcement expected in the near future, can start on any day. Thus, $\sigma_0 = \sigma_{\min} = 0.15$, $\sigma_{\max} = 1.0$, $d = 7$, $p = 1$, $f = 1$.

- The central bank of country XYZ meets once a week. It is expected that an important economic decision will be made in one of its future meetings, though it is not known in which one. Heavy trading on the day following the announcement is anticipated. In this case, $\sigma_0 = \sigma_{\min} = 0.15$, $\sigma_{\max} = 1.0$, $d = 1$, $p = 7$, $f = 1$ may be a realistic volatility shock scenario.

The crucial property of volatility shock scenarios is that they leave open when the shock periods occur. If the timing of events is known, a time-dependent conventional uncertain volatility scenario works adequately. It is the additional dimension of uncertainty of timing which opens the door to worst-case considerations.

The other quantitative difference between conventional and volatility shock scenarios is the width of the volatility band: while conventional scenarios may allocate a 0.1–0.2 volatility band, for instance, volatility shock scenarios provide for volatility spikes of much larger amplitude. Wide bands in the conventional scenario suffer from two flaws: a) they lead to wide price bands, and b) they do not reflect the isolated nature of events which influence market bahavior. Volatility shock scenarios alleviate both drawbacks.

8.1.1 Worst-case Volatility Shocks

Under the worst-case paradigm volatility shock periods are located such that the resulting worst-case price is maximized. The market is regarded as adversary that triggers events perturbing the prior volatility at the most adverse moment.

The objective of worst-case pricing under a conventional volatility scenario has been formulated in Sect. 4.2:

> Given a portfolio $\mathbf{X}$ and a position $\lambda \in \mathbb{R}^k$ in $\mathbf{X}$, which $\sigma \in \mathcal{C}$ maximizes today's value of $(\mathbf{X}, \lambda)$?

The extension to volatility shock scenarios is straightforward and goes as follows.

> Given a portfolio $\mathbf{X}$ and a position $\lambda \in \mathbb{R}^k$ in $\mathbf{X}$. Given furthermore prior and shock volatilities $\sigma_{\min} \leq \sigma_0 \leq \sigma_{\max}$ and shock scenario attributes d, p and f. Which $\sigma \in \mathcal{D}$ maximizes today's value of $(\mathbf{X}, \lambda)$?

$\mathcal{D}$ has been defined in Def. 8.2 as the class of volatilities σ that satisfy

$$\sigma_{\min} \leq \sigma(\omega, t) \leq \sigma_{\max} \tag{8.8.1}$$

during shock periods and

$$\sigma(\omega, t) = \sigma_0 \tag{8.8.2}$$

during silent periods. We assume that $\mathbf{X}$ contains only vanilla options, all maturing at time t_N.

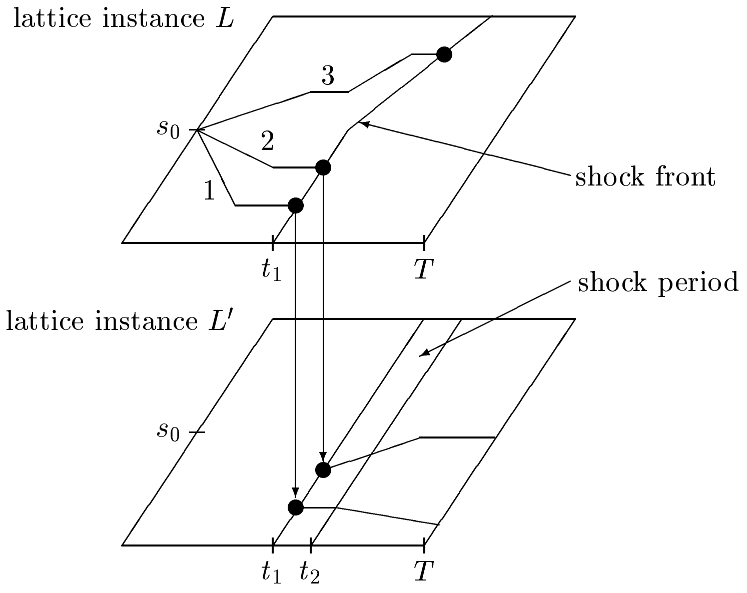


Figure 8.1: Paths 1 and 2 hit the shock front at time t_1 and switch to lattice instance L', which solves a conventional worst-case pricing problem with time-dependent $\sigma_{\min}$ and $\sigma_{\max}$ (i.e., $\sigma_{\min} = \sigma_{\max} = \sigma_0$ for $t < t_1$ and $t > t_2$). Path 3 hits the shock front at a later time and continues on a lattice instance with a different conventional worst-case volatility scenario

Multi-lattice Dynamic Programming Revisited

The worst-case volatility-shock pricing problem can be solved with multi-lattice dynamic programming. The number of lattice instances depends on the volatility shock scenario and can be known beforehand. Each lattice instance carries $(\mathbf{X}, \lambda)$, but solves PDE (4.4.8) with a different, non-path-dependent (!) volatility coefficient. Transferring data between lattice instances works much like in the American case: local decisions are made with regard to the “shock front“, i.e. the optimal (that is, worst) time of entering a shock period. The shock front is the analogon of the early exercise boundary.

Figure 8.1 gives an example. Lattice instance L is the top-level lattice instance yielding the final result $\hat{V}(0, 0; L)$. Paths 1, 2 and 3 originating at s_0 and hitting the shock front at time t_1 (paths 1 and 2) respectively at some later, unspecified time (path 3) are traced.

After hitting the shock front, paths 1 and 2 continue on lattice instance L'. L' differs from L in that it prices with the conventional uncertain volatility scenario

$$\begin{aligned} \sigma(S_t, t) &= \sigma_0 && (t < t_1 \text{ or } t > t_2) \\ \sigma_{\min} \leq \sigma(S_t, t) &\leq \sigma_{\max} && (t_1 \leq t \leq t_2) \end{aligned} \tag{8.8.3}$$

with a fixed period of volatility oscillation between times t_1 and t_2. L, on the other hand, prices with σ_0 between $t = 0$ and the shock front, whose location is determined with the dynamic programming method.

Path 3 does not hit the shock front at time t_1 and therefore does not continue on L', but on another lattice instance whose shock period is located suitably. Notice that while the shock front in L is uneven, the shock period in L' itself starts uniformly at time t_1 and ends uniformly at time t_2.

The example seems to suggest that there must be a lattice instance for every possible location of the shock period. This is not so; lattice instances can be reset and reused in the rollback scheme as soon as a shock period is finished. A combination of high-level handling of lattice instances and conventional worst-case pricing is powerful enough to solve the worst-case pricing problem under volatility shock scenarios.

Definition 8.3 (Extended lattice signature). *Given $(\mathbf{X}, \lambda)$, duration d, periodicity p and frequency f. The* extended signature *of a lattice instance L for the so-specified volatility shock scenario is a quintuple $(\mathbf{X}, \lambda, \tau, \xi, \delta)$, where $\tau \in \{\text{conventional}, \text{consolidate}\}$ is the* type*, $0 \leq \xi \leq f$ is the* level*, and $0 \leq \delta \leq \lceil d/p \rceil$ is the* offset *of the lattice instance. The offset is undefined if $\tau = $ consolidate.*

If $\mathbf{X}$ contains only vanilla options, all lattice instances carry the same portfolio $(\mathbf{X}, \lambda)$. In that case, we ommit $\mathbf{X}$ and λ and write (τ, ξ, δ).

Consolidating lattice instances use subordinate conventional lattice instances to locate the shock front. If the duration exceeds the periodicity, potential shock periods may overlap, and up to $\lceil d/p \rceil$ conventional lattice instances need to be maintained to feed a single consolidating lattice instance. Consolidating and associated conventional lattice instances are grouped in levels. Levels are ordered, for conventional lattice instances, in turn, fetch their boundary data from lower level consolidating lattice instances. Thus, L in Fig. 8.1 is consolidating while L' is conventional.

Level 0 is unique in that it does not contain any conventional lattice instances. The consolidating lattice instance of level 0 prices $(\mathbf{X}, \lambda)$ by definition with σ_0. On level 0, pricing becomes linear.

Figure 8.2 explains these concepts for $d = 4$, $p = 2$ and $f = 1$. Shock periods are possible between days 0–4, 2–6, 4–8, 6–10 and 8–10 (the last one being cut off at day 10). The main lattice instance L_1 imports worst-case prices on days 0, 2, 4, 6 and 8 from conventional lattice instances L_1^0 and L_1^1, depending on the offset. After maximizing locally just like it is done for American options, the resulting value is rolled back 2 days with linear volatility $\sigma = \sigma_0$. Then data is imported from L_1^0 or L_1^1 and compared again. The shock front is implicitly given by the outcome of the local maximization operations and continuously readjusted.

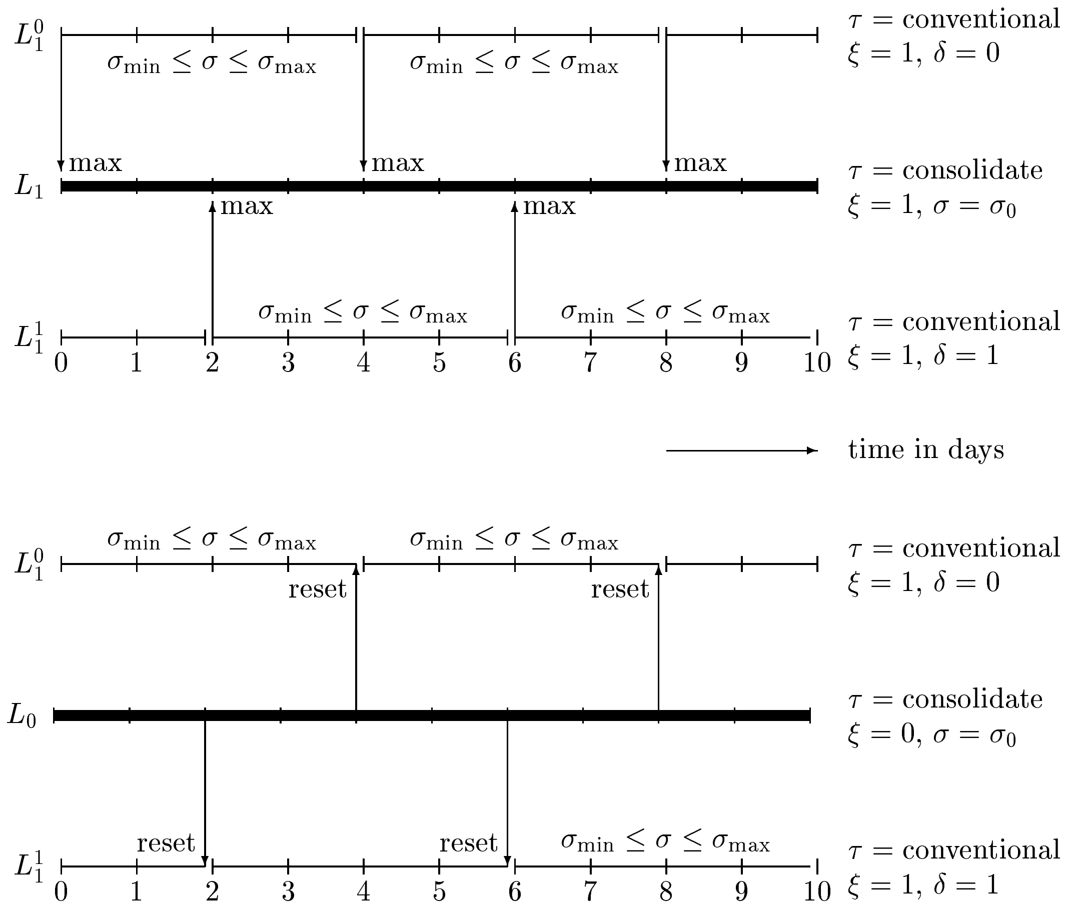
The conventional lattice instances L_1^0 and L_1^1 are reused several times. After worst-case prices have been transferred to L_1 on days 0, 2, 4, 6 and 8, the lattice instances are reset with current linear prices, copied from L_0. Here and in the subsequent paragraphs, "current" refers to the loop variable i which iterates through time slices $N, \ldots, 0$ (i is part of the input in the algorithm in Fig. 5.5). The function of L_1, L_1^0, L_1^1 and L_0 can be summarized, bottom-up, as follows:

- L_0 is the lattice instance at the lowest level and is used to price $(\mathbf{X}, \lambda)$ at the prior volatility $\sigma = \sigma_0$.

- L_1^0 is used to price $(\mathbf{X}, \lambda)$ under the conventional worst-case volatility scenario with a volatility band $\sigma_{\min} \le \sigma \le \sigma_{\max}$ during the current shock period $[2lp, 2lp + 4]$, $l \ge 0$ chosen suitably, and $\sigma = \sigma_0$ during the tail period $[2lp + 4, 10]$. The offset of L_1^1 is $\delta = 0$. As the tail period becomes longer and a volatility shock date is crossed, L_1^0 is reset with data from L_0.

- L_1^1 is used to price $(\mathbf{X}, \lambda)$ under the conventional worst-case volatility scenario with a volatility band $\sigma_{\min} \le \sigma \le \sigma_{\max}$ during the current shock period $[(2l+1)p, (2l+1)p+4]$, $l \ge 0$ chosen suitably, and $\sigma = \sigma_0$ during the tail period $[(2l+1)p+4, 10]$. The offset of L_1^0 is $\delta = 1$, corresponding to a shift of $\delta p = 2$ days of shock periods. As the tail period becomes longer and a volatility shock date is crossed, L_1^1 is reset with data from L_0.

- L_1 holds prices for $(\mathbf{X}, \lambda)$ which, during rollback, represent the expected payoff



L_1^0
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
τ = conventional
$\xi = 1$, $\delta = 0$
max
max
max
L_1
max
max
τ = consolidate
$\xi = 1$, $\sigma = \sigma_0$
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
L_1^1
τ = conventional
$\xi = 1$, $\delta = 1$
0 1 2 3 4 5 6 7 8 9 10
time in days
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
L_1^0
reset
reset
τ = conventional
$\xi = 1$, $\delta = 0$
L_0
τ = consolidate
$\xi = 0$, $\sigma = \sigma_0$
reset
reset
$\sigma_{\min} \le \sigma \le \sigma_{\max}$
L_1^1
τ = conventional
$\xi = 1$, $\delta = 1$
0 1 2 3 4 5 6 7 8 9 10

Figure 8.2: Four lattice instances L_1, L_1^0, L_1^1 and L_0 are needed to solve a volatility shock scenario with shock duration $d = 4$, periodicity $p = 2$ and frequency $f = 1$. L_1^0 is responsible for the shock periods $[2lp, 2lp+4]$, and L_1^1 is responsible for the shock periods $[(2l+1)p, (2l+1)p+4]$, where $l \geq 0$. After the worst-case price for a shock period has been incorporated into the main lattice instance L_1 through local maximization (top picture), the associated conventional lattice instance is reset with the current linear price obtained with the prior volatility σ_0 (bottom picture)

under the assumption that the volatility shock period has occured sometime between the current time slice and day 10. As the rollback proceeds from day 10 to day 0, this assumption is periodically verified by checking whether the price for $(\mathbf{X}, \lambda)$ increases if the volatility shock period starts at the current time slice.

To decrease the periodicity p from 2 to 1 requires two additional conventional lattice instances for shock periods with offsets 1 and 3, respectively. The resulting cycle of periods is $[(4l + o)p, (4l + o)p + 4]$, $0 \leq o \leq 3$ and $l \geq 0$. To increase p from 2 to 4, on the other hand, makes L_1^1 superfluous, and three lattice instances overall suffice. Also note that the days on which shock periods start and end must be matched by the lattice: if d and/or p are small, the discretization becomes necessarily denser. $1/p$ is proportional to the time complexity of the pricing problem. Fine-tuning of both d and p can lead to a significant gain in response time.

If the shock frequency f is increased from 1 to 2, a new level $\xi = 2$ needs to be added. L_2 becomes the main lattice, and $\hat{V}(0, 0; L_2)$ the overall result. L_2 is interpreted as follows:

- L_2 holds prices for $(\mathbf{X}, \lambda)$ which, during rollback, represent the expected payoff under the assumption that up to *two* volatility shock periods occur sometime between the current time slice and day 10.

The new conventional lattice instances L_2^0 and L_2^1 with signatures ($\tau =$ conventional$, \xi = 2, \delta = 0$) and ($\tau =$ conventional$, \xi = 2, \delta = 1$), respectively, are reset with data from L_1 when shock dates are crossed. They are interpreted as follows:

- L_2^0 prices $(\mathbf{X}, \lambda)$ under the conventional worst-case volatility scenario with a volatility band $\sigma_{\text{min}} \leq \sigma \leq \sigma_{\text{max}}$ during the current shock period $[2lp, 2lp + 4]$, $l \geq 0$ chosen suitably, and under the assumption that an additional shock period occurs during the tail period $[2lp + 4, 10]$.

- L_2^1 prices $(\mathbf{X}, \lambda)$ under the conventional worst-case volatility scenario with a volatility band $\sigma_{\text{min}} \leq \sigma \leq \sigma_{\text{max}}$ during the current shock period $[(2l + 1)p, (2l + 1)p + 4]$, $l \geq 0$ chosen suitably, and under the assumption that an additional shock period occurs during the tail period $[(2l + 1)p + 4, 10]$.

Care has to be taken that L_2^0 and L_2^1 are reset with data from L_1 only after L_1 has been processed: the data must reflect the result of the local maximization at L_1 on the shock date.

Figure 8.3 gives a schematic overview over the hierarchy of lattice instances for general f. Each consolidating lattice instance L_n, $0 \leq n \leq f$, carries the full solution of a worst-case volatility-shock pricing problem with frequency $f' = n$.

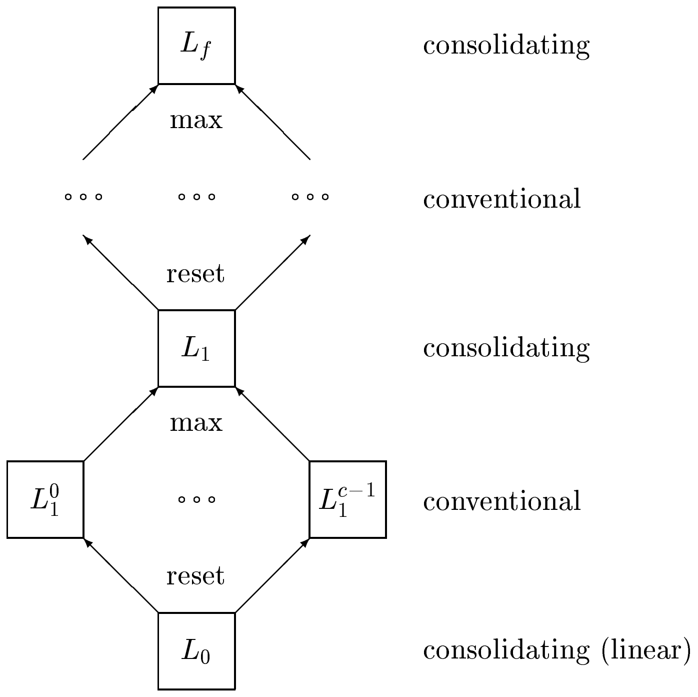


Figure 8.3: The hierarchy of lattice instances for general f. Arrows represent the dataflow. c is the number of conventional lattice instances per level. The "max" and "reset" labels correspond to the "max" and "reset" operations in Fig. 8.2

Algorithms

In the following we assume a discretization that coincides with day boundaries: $t_i = i$ for $0 \leq i \leq N$. Depending on the duration and periodicity of the shock volatility scenario, this convention may be relaxed in an actual implementation.

```
**Input**: Duration $d$, periodicity $p$, frequency $f$
**Output**: A set of lattice instances

1. Set $c = \lceil d/p \rceil$. $c$ is the number of conventional lattice instances per level
2. Create lattice instance $L_0$ with signature
       $(\tau = \text{consolidate}, \xi = 0, \delta = \text{undefined})$
3. Repeat for $n = 1, \dots, f$:
   (a) Create lattice instance $L_n$ with signature
           $(\tau = \text{consolidate}, \xi = n, \delta = \text{undefined})$
   (b) Repeat for $m = 0, \dots, c-1$:
       i. Create lattice instance $L_n^m$ with signature
              $(\tau = \text{conventional}, \xi = n, \delta = m)$
```

Figure 8.4: The algorithm to create all required lattice instances for a given volatility shock scenario

The algorithm in Fig. 8.4 computes the required number of conventional lattice instances, and creates all lattice instances. The following lemma shows that the algorithm creates the necessary number of lattice instances, and uses them optimally.

Lemma 8.4 (Lattice instance creation). *Given a volatility shock scenario with duration d, periodicity p and frequency f. For any given level n, $1 \leq n \leq f$, the algorithm in Fig. 8.4 facilitates an assignment of shock periods to conventional lattice instances such that no two overlapping shock periods are assigned to the same lattice instance. (Touching at the endpoints is allowed.)*

Proof. Any shock period can be written $[lp, lp+d]$, $l \geq 0$. The quantity $c = \lceil d/p \rceil$ defined in step 1 of the algorithm is the smallest number such that $cp \geq d$, for

$$cp = \lceil d/p \rceil p \geq (d/p)p = d \tag{8.8.4}$$

on one side, and

$$(c-1)p = (\lceil d/p \rceil - 1)\, p < ((d/p+1)-1)\, p = d \tag{8.8.5}$$

in the other direction.

Now fix a level n. Cosider the first c shock periods $[lp, lp+d]$, $0 \leq l \leq c-1$. All c shock periods overlap, for their start dates $0, p, 2p, \ldots, (c-1)p$ all lie within the first period $[0, d]$, as shown in (8.8.5). Thus, at least c lattice instances are required to fulfill the condition that shock periods assigned to the same lattice instance don't overlap. We assign each of the c shock periods to a separate lattice instance.

Let $L_n^0, \ldots, L_n^{c-1}$ be the lattice instances created. The next shock period that needs assignment is $[cp, cp+d]$. Since $cp \geq d$, assignment of this shock period to L_n^0 does not violate the no-overlap condition (although the periods may touch at their endpoints). It is easy to see how the round-robin assignment proceeds.

In summary, if day i is divisible by p, i.e. is a day on which a shock period may start, then the lattice instance to which this shock period is assigned within any given level is $m = i/p \mod c$. □

The worst-case volatility-shock pricing problem is solved in two phases. In phase 1, values are rolled back in whatever scheme has been selected (explicit or mixed explicit/implicit). In addition, local maximization is performed for consolidating lattice instances if the processed time slice falls on day on which a shock period starts. During phase 1, lower level lattice instances are processed first, and conventional lattice instances are processed before the consolidating lattice instance within the same level. This rule is an extension of the external consistency rule proposed in Sect. 5.1. Figure 8.5 shows the algorithm.

Phase 2 is dedicated to resetting the conventional lattice instances, depending on whether their offsets δ and the round-robin index $i/p \mod c$ of the shock start-date match. No particular order needs to be observed in phase 2. The data collected from consolidating lattice instances has been prepared in phase 1. The algorithm is shown in Fig. 8.6.

Input: Lattice instance L with signature (τ, ξ, δ), time i
Output: $\hat{V}(j,i;L)$ for $D \leq j \leq U$

1. If $\tau =$ conventional:
   (a) Use the algorithm in Fig. 5.5 to get worst-case values $\hat{V}(j,i;L)$ for $D \leq j \leq U$
2. If $\tau =$ consolidate:
   (a) Apply the algorithm in Fig. 5.5 with $\sigma_{\min}$ and $\sigma_{\max}$ set to σ_0 (essentially, a linearized version of the algorithm) to get initial $\hat{V}(j,i;L)$ for $D \leq j \leq U$
   (b) If $\xi > 0$ and i is divisible by the periodicity p:
      i. With $c = \lceil d/p \rceil$ and $m = i/p \mod c$, repeat for $D \leq j \leq U$:

$$\hat{V}(j,i;L) := \max \left[\hat{V}(j,i;L), \hat{V}(j,i;L_\xi^m) \right]$$

where we can be sure that conventional L_ξ^m has already been processed earlier in phase 1; adjust the gradient of $\hat{V}(j,i;L)$ accordingly

Figure 8.5: Phase-1 algorithm, applied to all lattice instances in the order L_0, L_1^0, ..., L_1^{c-1}, L_1, ..., L_f. Note that $\hat{V}(j,i;L)$ is treated as a variable which can be modified

Instead of formalizing the notion of volatility shocks any further, we use the algorithms in Figs. 8.5 and 8.6 to define the worst-case volatility-shock price of a portfolio. The consistency of the algorithms is clear by Lemma 8.4 and inspection. They are a straightforward extension of the basic concepts developed in Chapter 5.

Definition 8.5 (Worst-case volatility-shock price). *Given a volatility shock scenario with duration d, periodicity p and frequency f, together with prior volatility σ_0 and shock volatility bounds $\sigma_{\min}$ and $\sigma_{\max}$. The value obtained for a portfolio $(\mathbf{X}, \lambda)$ by running the algorithms in Figs. 8.5 and 8.6, embedded in a multi-lattice dynamic-programming framework as discussed in Chapter 5, is called the* worst-case volatility-shock price *of $(\mathbf{X}, \lambda)$.*

In particular, the subadditivity of the worst-case price asserted in Fact 4.7 (and re-

```
Input: Lattice instance $L$ with signature $(\tau, \xi, \delta)$, time $i$
Output: Adjusted $\hat{V}(j, i; L)$ for $D \le j \le U$

1. If $\tau =$ conventional:
   (a) If $i$ is divisible by the periodicity $p$:
       i. With $c = \lceil d/p \rceil$, set $m = i/p \mod c$
       ii. If $m = \delta$ repeat for $D \le j \le U$:
              $\hat{V}(j, i; L) := \hat{V}(j, i; L_{\xi-1})$
           where consolidating $L_{\xi-1}$ has been processed in phase 1; reset the gradient
           accordingly
```

Figure 8.6: Phase-2 algorithm, applied to all conventional lattice instances

peated later, for American options, in Prop. 7.12) is maintained through the application of the maximum operator in step 2(b)i in Fig. 8.5.

Numerical Issues

Volatility shock scenarions encourage short volatility spikes with large amplitude. Since these spikes can be located anywhere on the lattice, $\sigma_{\max}$ is the relevant upper volatility bound for the algorithm in Fig. 5.4. Recall that the algorithm computes the discretization in time and space for the explicit finite difference scheme. Mixed explicit/implicit schemes don't require exceptionally small time steps and may in the case of volatility shock scenarios be faster than explicit schemes. For this reason, Crank-Nicholson is used in the following experiments.

The validity of the PDE (4.4.8) is another numerical issue. Recall that the local volatility under uncertainty is given by

$$\Sigma\left(\frac{\partial^2 f}{\partial S^2}\right) = \begin{cases} \sigma_{\max} & \text{if } \frac{\partial^2 f}{\partial S^2} \ge 0 \\ \sigma_{\min} & \text{if } \frac{\partial^2 f}{\partial S^2} < 0 \end{cases} \tag{8.8.6}$$

which has the welcome property that $\frac{\partial}{\partial \lambda_n} \Sigma = 0$ almost everywhere, for $1 \leq n \leq |\lambda|$. This has the consequence that the gradient in λ is a solution of (4.4.8), too, with different boundary conditions. Volatility-shock scenarios have only a finite number of additional transitions in volatility space and therefore do not change this property.

8.1.2 Experimental Results

All tests were performed on a Pentium/166 MHz PC running Windows NT Workstation 4.0/SP 3, with 128 MB of RAM. The software is called Mtg and has been written and compiled with Microsoft Visual C++ 5.0, optimizations activated.

Experiment 1: A Butterfly Spread

Consider the butterfly spread of four call options in Fig. 8.7. The maturities of the options are 30, 50, 40 and 60 days, respectivly. The current stock price is $S_0 = 100$, and the interest rate is $r = 0.03$.

type	maturity	strike	λ_n
call	30	95	1
call	50	100	-1
call	40	110	-1
call	60	115	1

Figure 8.7: A butterfly spread consting of four call options. The spread is not perfect: the maturity dates of the calls are not aligned

The spread is priced under three volatility scenarios:

1. A linear volatility scenario with constant volatility $\sigma = 0.15$.
2. A volatility shock scenario with $\sigma_{\text{min}} = \sigma_0 = 0.15$, $\sigma_{\text{max}} = 0.5$, duration $d = 3$ days, periodicity $p = 1$ day and frequency $f = 1$.
3. A conventional worst-case volatility scenario with $\sigma_{\text{min}} = 0.15$ and $\sigma_{\text{max}} = 0.184052$, where the latter was chosen to match the average volatility over any high-volatility

path in scenario 2:

$$\sigma_{\max} = \sqrt{\frac{1}{60}\int_0^{60} \sigma^2\, dt} = \sqrt{\frac{1}{60}\left(3 \times 0.5^2 + 57 \times 0.15^2\right)} \tag{8.8.7}$$

The time step for the Crank-Nicholson finite-difference scheme is $dt = 1/(10 \times 365)$.

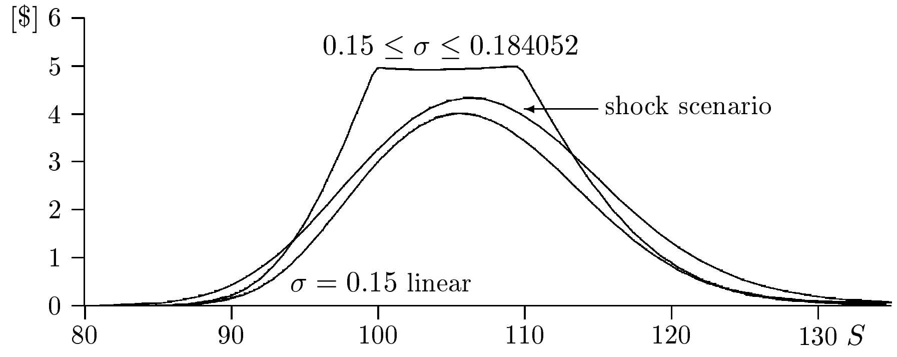


Figure 8.8: The butterfly spread of Fig. 8.7 priced under three volatility scenarios. Shown is the worst-case value plotted against today's value of the underlying. The parameters for the shock scenarios are $\sigma_{\min} = \sigma_0 = 0.15$, $\sigma_{\max} = 0.5$, $d = 3$, $p = 1$ and $f = 1$

Figure 8.8 plots the resulting worst-case values against the time-zero value of the underlying. The linear scenario obviously yields the smallest value throughout. The relation between the two non-linear scenarios is less apparent. The volatility shock scenario is smoother and comes closer to the linear scenario. It may be more appealing to practitioners.

Figure 8.9 contains an image of the top-level consolidating lattice instance. Black regions indicate where the maximum operator in step 2(b)i in Fig. 8.5 locates the potential start of a shock period. Conversly, any path starting at time 0 enters its shock period when it hits one of the black regions for the first time. Shock periods are predominantly entered near maturity dates.

Experiment 2: Increasing the Frequency

In experiment 2 the portfolio in Fig. 8.7 is priced again, under the same volatility shock scenario with $\sigma_{\min} = \sigma_0 = 0.15$, $\sigma_{\max} = 0.5$, duration $d = 3$ and periodicity $p = 1$. The

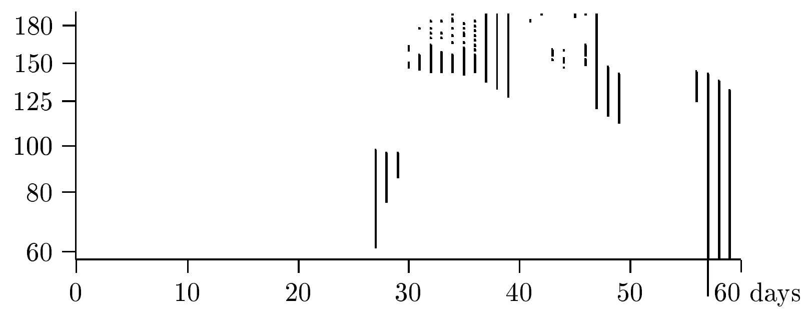


Figure 8.9: The shock front unveiled. Black regions indicate where three-day shock periods start. The four clusters correspond to the four maturities 30, 40, 50 and 60 days. The spatial axis is in log-scale; the labels are normalized

frequency f varies between 1 and 20. Figure 8.10 lists the running time, the number of lattice instances created, and the worst-case value as a function of f. Figure 8.11 shows the worst-case value graphically.

The number of lattice instances created by the dynamic creation scheme is $f \times (\lceil d/p \rceil + 1) + 1 = 4f + 1$. Here, $\lceil d/p \rceil$ is the number of conventional lattice instances per level, $\lceil d/p \rceil + 1$ is the number of overall lattice instances per level, and the additional lattice instance L_0 is used for the level-zero linear pricing. The running time mirrors the linear growth of the number of lattice instances, discounting some noise for higher values of f.

For $f = 20$ the worst-case volatility-shock value and the conventional worst-case value obtained under an uncertain volatility scenario $0.15 \leq \sigma \leq 0.5$ coincide, for 20 volatility shocks of 3 days length each cover the entire 60-day lifetime of the portfolio. A coverage of $10 \times 3 = 30$ days is, according to the data in Fig. 8.10, already sufficient to reproduce the conventional value to within 1.3%.

Figure 8.12 shows the shape of the shock front on the top-level lattice for $f = 2, 3, 4$, respectively. In this context, the top-level lattice for $f = a$ is a lattice instance b levels away from the top for $f = a + b$. If a path hits one of the black regions in the top picture, a three-day volatility-shock period is initiated after which the path continues on the lattice instance shown in Fig. 8.9. Similarly, with intermediate three-day transitions with high volatility oscillation, paths examined under scenarios $f = 3$ and $f = 4$ jump from the middle respectively bottom picture to the top respectively middle picture by passing through one of the black regions. The shock region shows a vertical pattern because shock periods may only start on day boundaries, but the time step is 1/10 of a day.

Experiment 3: Convergence

Experiments 1 and 2 were executed with a time step of $dt = 1/(10 \times 365)$. Worst-case prices for $d = 3$, $p = 1$ and $f = 1, 2, 3$, computed with 1, 2, 5, 10, 20, 50 and 100 steps per day and shown in Fig. 8.13, certify the stability of the results obtained. No significant improvement is achieved for $dt < 1/(10 \times 365)$.

Conclusion

The concept of refined volatility scenario makes direct economic sense. In particular, volatility shock scenarios promise to remedy some of the flaws of conventional uncertain volatility scenarios based on a perpetual volatility band. Among these are too pessimistic price bands and unrealistic mapping of market behavior.

The preceding discussion and experiments prove that the computational overhead is linear in the granularity d/p of the volatility shock scenario, and therefore bearable. No sacrifices have to be made in terms of accuracy.

Figure 8.8 shows that volatility-shock prices are less extremal than prices obtained under conventional uncertain volatility scenarios. Figure 8.11 shows that volatility shock scenarios react gradually to an increase in the extent of volatility oscillation. Volatility shock scenarios therefore permit to fine-tune the market model to a great degree. They promise to be a valuable tool in assessing volatility risk.

8.2 Volatility Shocks and Exotic Options

Exotic volatility scenarios and portfolios of exotic options can be combined. The computational overhead is multiplicative. The algorithm in Fig. 8.4 creates an initial set of lattice instances with signatures $(\mathbf{X}, \lambda, \tau, \xi, \delta)$; additional lattice instances with signatures $(\mathbf{X}', \lambda', \tau', \xi', \delta')$, $(\mathbf{X}', \lambda') \subset (\mathbf{X}, \lambda)$, may be created dynamically later (if the portfolio contains American options) or statically (if the portfolio contains barrier options).

Steps 1 and 2a in Fig. 8.5 refer to the untainted rollback scheme in Fig. 5.5. In the case of exotic options, more sophisticated operations based on dynamic programming need to be executed instead. Chapters 6 and 7 have explained how lattice instances with partial portfolios are maintained to locate the exercise boundary or to supply it with data, if barrier and/or American options are part of the problem. Luckily, this kind of data transfer between lattice instances can be confined to steps 1 and 2a: the maximum operator in the expression for $\hat{V}(j, i; L)$ in step 2(b)i is at the highest level and does not interfer. (In mixed implicit/explicit schemes, however, this may create the same problem as for American options, making iterative refinement of the initial solution of the underlying linear system of equations necessary.)

Figure 8.14 illustrates the distinction between the “horizontal” relationship of lattice instances for different partial portfolios, but with identical volatility shock parameters, and the “vertical” relationship between consolidating and conventional lattice instances with differing volatility shock parameters. The relationship between τ and τ', ξ and ξ', δ and δ' is predetermined and has been discussed above. The relationship between $(\mathbf{X}, \lambda)$ and $(\mathbf{X}', \lambda')$ depends on the makeup of the portfolio. Certain is only that $(\mathbf{X}', \lambda') \subset (\mathbf{X}, \lambda)$.

Figure 8.15 generalizes the microscopic example of Fig. 8.14 and shows a data flow diagram for a volatility shock scenario with frequency $f = 2$. Each stack of boxes represents a component scenario for a fixed partial portfolio. The component scenario imports data at every level from a subordinate component scenario located to its left, depending on the requirements arising from the exotic options in the portfolio.

Mtg, our pricer, is capable of handling both exotic options and volatility shock scenarios at the same time. We do not give experimental results in this work.

frequency f	time [s]	# lattices	value
1	11.5	5	3.264
2	20.9	9	3.462
3	33.2	13	3.648
4	43.3	17	3.816
5	53.0	21	3.946
6	70.8	25	4.051
7	78.8	29	4.136
8	89.1	33	4.197
9	92.5	37	4.240
10	106.6	41	4.271
11	128.1	45	4.291
12	138.8	49	4.306
13	151.1	53	4.316
14	172.9	57	4.322
15	150.3	61	4.325
16	178.3	65	4.326
17	220.3	69	4.327
18	233.4	73	4.327
19	239.2	77	4.327
20	271.5	81	4.327

Figure 8.10: Running times, number of lattice instances and worst-case volatility-shock values for the portfolio in Fig. 8.7, as a function of the shock frequency f

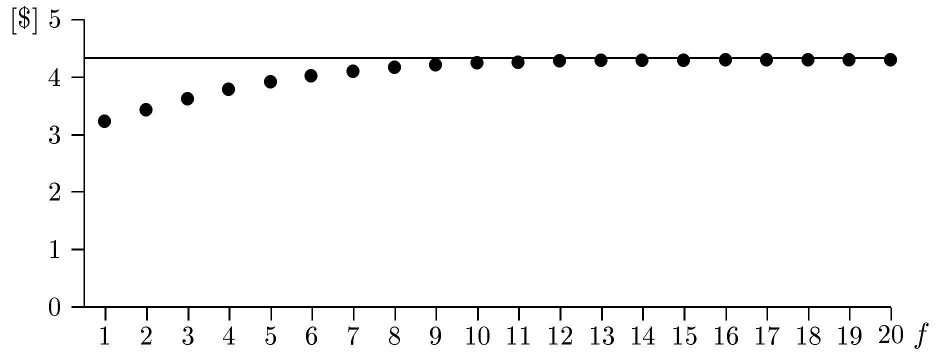


Figure 8.11: Worst-case volatility-shock values at $S = 100$ for the butterfly spread in Fig. 8.7, as a function of the shock frequency f. The horizontal line represents the worst-case value under the conventional volatility scenario $0.15 \leq \sigma \leq 0.5$

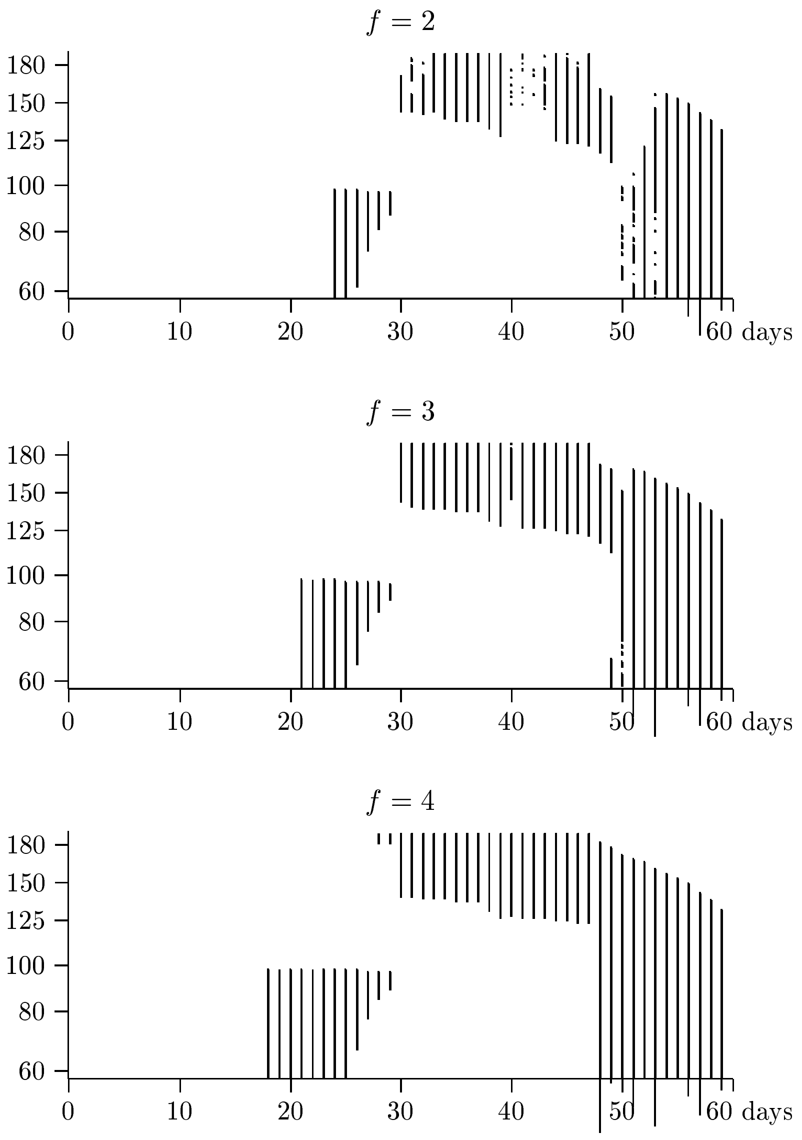


Figure 8.12: The top-level shock front unveiled for $f = 2, 3, 4$. Black regions indicate where three-day shock periods start. Notice that the shock front expands to the left as the frequency increases. (also compare with Fig. 8.9 for $f = 1$)

time step	price		
	$f = 1$	$f = 2$	$f = 3$
$1/365$	3.3859	3.5830	3.7672
$1/(2 \times 365)$	3.2922	3.4996	3.6887
$1/(5 \times 365)$	3.2717	3.4732	3.6603
$\rightarrow 1/(10 \times 365)$	3.2637	3.4619	3.6488
$1/(20 \times 365)$	3.2587	3.4571	3.6458
$1/(50 \times 365)$	3.2569	3.4549	3.6437
$1/(100 \times 365)$	3.2560	3.4541	3.6429

Figure 8.13: Worst-case prices for the call spread in Fig. 8.7 under the shock-volatility scenarios $d = 3$, $p = 1$ and $f = 1, 2, 3$. Results in experiments 1 and 2 were obtained with a time step $dt = 1/(10 \times 365)$. The data shows that there are no convergence issues; doubling the number of steps per day from 10 to 20 changes the result by 0.15, 0.13 and 0.08%, respectively

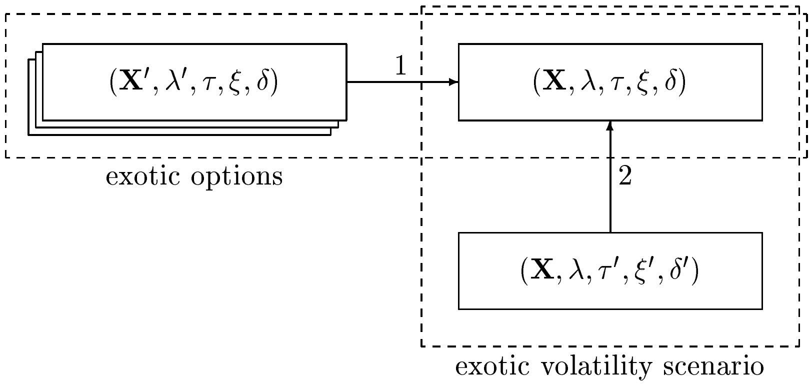

$(\mathbf{X}', \lambda', \tau, \xi, \delta)$
1
$(\mathbf{X}, \lambda, \tau, \xi, \delta)$
exotic options
2
$(\mathbf{X}, \lambda, \tau', \xi', \delta')$
exotic volatility scenario

Figure 8.14: The lattice instance with signature $(\mathbf{X}, \lambda, \tau, \xi, \delta)$ imports data from lattice instances with signatures $(\mathbf{X}', \lambda', \tau, \xi, \delta)$ and $(\mathbf{X}, \lambda, \tau', \xi', \delta')$. The numbers indicate the order in which data is imported

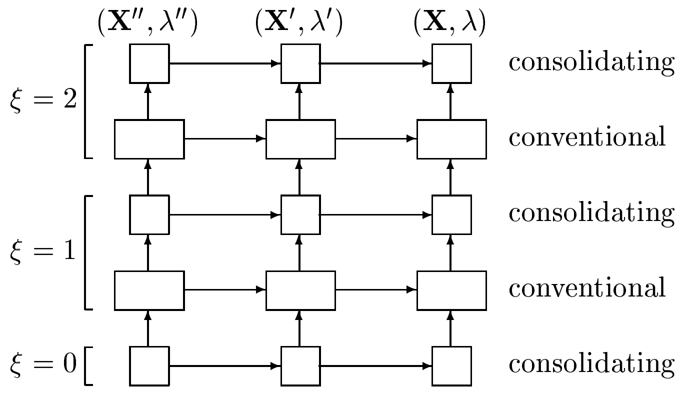


Figure 8.15: A volatility shock scenario with $f = 2$ for a portfolio $\mathbf{X}$ containing some exotic options. Each box represents one or more lattice instances; arrows represent the data flow. The size of each box is proportional to the number of lattice instances in the group it represents (in this case, we conjecture $p < d$)

Part III

Object-oriented Implementation

9 The Architecture of MtgLib

The algorithms of Chapters 6, 7 and 8 are part of a programming system for nonlinear models in computationl finance. The name of this system, Mtg, has already appeared where experimental results were presented.

Mtg consists of components written in C++ and Java. Figure 9.1 arranges the components of Mtg in a top application layer, and a bottom support layer. The support layer also contains in dashed boxes the third party software required to run the component immediately on top.

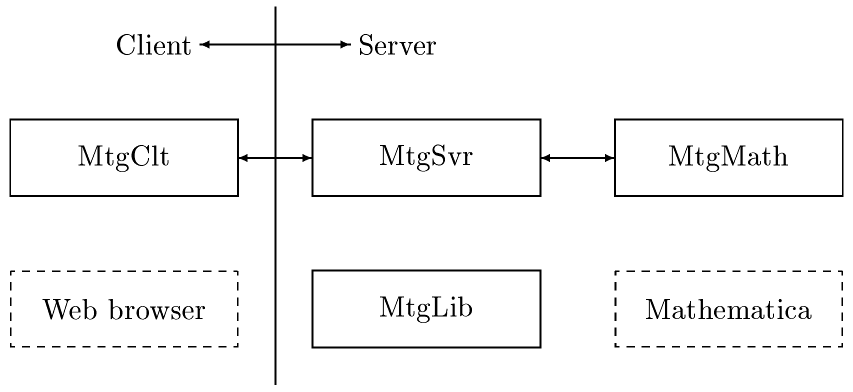


Figure 9.1: Components MtgSvr and MtgLib are written in C++. MtgClt is written in Java and runs in a Web-browser environment. MtgMath is part C++, part Mathematica script

The main components of Mtg are:

MtgLib The core C++ library. MtgLib contains the majority of the code written for this thesis, or about 81500 lines of code. MtgLib is platform-independent.

MtgSvr A background server process. MtgSvr receives and answers requests via TCP. The text protocol used by MtgSvr serves mainly to transmit descriptions of object instances of classes in MtgLib. MtgSvr is a tiny wrapper around MtgLib. Under Unix, MtgSvr is a deamon; on Windows NT, MtgSvr is implemented as a service. See Buff (1999b).

MtgClt A Java front-end that knows how to communicate with MtgSvr. MtgClt can act both as stand-alone application and as applet run by a Web browser. It is powerful enough to let the user create pricing problems with barrier and American options, under worst-case and volatility shock scenarios. It is, however, restricted to the lattice approach for Black-Scholes. MtgClt consists of approximately 11500 lines of Java code (about half of which is general-purpose).

MtgMath A front-end that uses the symbolic and plotting capabilities of the software system for technical computation, Mathematica. MtgMath was mainly used to do the experiments and prepare the graphs for this thesis.

There are other components of Mtg: MtgCal by Buff (1999c) is a model-independent online calibrator for fixed income instruments, based on Monte Carlo simulation and Entropy minimization. MtgGrab is a background process that collects current prices for US treasury paper on the Internet and calibrates a Vasicek short rate model. Daily results are published in Buff (1999d).

The philosophy of MtgGrab and MtgCal is briefly sketched in 10.2, to make the reader familiar with our current work and give an idea of future research directions. Since the thesis focuses on the complexity arising from exotic option portfolio and volatility scenarios, however, the architecture of MtgLib itself is at the center of our attention.

Before we proceed, some informal remarks about lingo concerning lattice-based evaluation. *Rollback* is the term used to describe the outer loop that iterates over the time slices $t_N, t_{N-1}, \ldots, t_0$ in the finite difference scheme. The inner loop processing that occurs for each time slice, i.e. the propagation of the solution at time slice t_{i+1} to the earlier time slice t_i, is called (rollback) *round.* Instead of time slice we sometimes say *hyperplane* to emphasize the data aspect. Under a one-factor model, the hyperplane is actually a two-dimensional plane with rows indexed $s_D, \ldots, s_0, \ldots, s_U$ (see Sect. 5.1), and columns for the total value and each gradient element. The number of columns used is called the *width* of the hyperplane. The *current* round, time slice, hyperplane, or node refers to the current iteration of the rollback loop (forgive the cyclic definition, it should be clear). We use the terms *Monte Carlo* and *simulation* interchangingly, and sometimes together.

9.1 The Class Hierarchy—External

The classes in MtgLib which correspond directly to input parameters and have some intuitive “meaning” to the user are called external. Instances of these classes may be defined in the scripting language in which MtgSvr communicates. The following categories of external classes exist:

Instruments Maturity, payoff policy, knock-out policy and early-exercise policy are the dominant orthogonal features of instruments. American/European options with or without knock-out boundaries and with linear of digital payoff are standardized in MtgLib (and in MtgClt, for that matter). A compact language allows to specify other types of instruments.

Portfolios Portfolios are collections of instruments and generalize some of their properties (the longest maturity, for instance).

Models Models consist of specifications of factors and model coefficients, possibly uncertain. With the exception of Sect. 10.2, a one-factor Black-Scholes model is used throughout this thesis.

Model coefficients Model coefficients may have their own classes to allow term structure. At this time, piecewise constant volatility and drift coefficients are supported. The volatility coefficient may be uncertain.

Scenarios Models and their (uncertain) coefficients are interpreted according to a prescribed scenario. We have discussed worst-case volatility and volatility scenarios. Their needs to be some consistency between the model and the scenario: if the model incorporates uncertain model coefficients, the scenario must be able to select concrete adaptions. Apart from that, scenarios are expressed without reference to the model.

Numerical methods Possible numerical methods are closed-form solutions (not considered here), explicit or mixed implicit/explicit finite difference schemes, or simulation methods (Monte Carlo). The requirements diverge: while finite difference schemes are based on a collection of lattice instances, Monte Carlo methods require path instances which are treated differently. In MtgLib, lattices and path spaces are

```
claim a {
    type american_put, maturity 30, strike 100 }
claim b {
    type european_put, maturity 25, strike 100 }
claim c {
    type european_call, maturity 20, strike 100, up-and-out 110 }
claim d {
    type european_put, maturity 15, strike 100, down-and-out 90 }

portfolio p { a long 200, b short 10, c long 2, d short 1 }

factor s {}
vol v { implied 30 10%..20% }
drift r { implied 30 2.5% }
model m { type back_scholes, vol v, discount r, s 100 }
scenario s { type worst_case, seller }

lattice l { model m, portfolio p, tree 3.5, time_step 0.5 }

evaluate { model m, lattice l, scenario s, portfolio p }
```

Figure 9.2: An example script understood by MtgSvr. Scripts like this can be transmitted to MtgSvr manually via telnet, or indirectly through the GUI of MtgClt

indeed seperate objects, with a third entity hierarchy of *compute engines* providing unified access.

Evaluaters MtgSvr collects objects in a repository without initiating concrete pricing operations itself. This is done by specifying an evaluator object which lives only while the particular portfolio/model/scenario combination is evaluated. Evaluators format the result and send it back through the TCP or MathLink pipe.

Figure 9.2 shows an example script that, when submitted to the MtgSvr deamon via TCP, initiates the computation of the worst-case price of a portfolio of three puts and one call, under a volatility scenarion $0.1 \leq \sigma \leq 0.2$. The script describes instances of all the classes listed above.

The following sections discuss each category in more detail. Although code fragments are included, this overview is not a tutorial on how to use MtgLib. Instead, design ideas are emphasized.

9.1.1 Instruments

The class hierarchy into which instruments are organized is shown in Fig. 9.3. The parent class `tClaim` is abstract and needs to be instantiated in subclasses.

Class name	Purpose
`tClaim`	Parent class (abstract)
`tStdClaim`	Standard calls and puts
`tCustomClaim`	Customizable in a mini-language
`tCashflow`	Supporting class (abstract)

Figure 9.3: The hierarchy of instrument classes. Indentation indicates inheritance. Standard instruments are calls and puts, American or European, with linear or digital payoff, with or without barriers

`tClaim` provides a unified interface to relevant instrument properties. Its definition is shown in Fig. 9.4. The scalar properties listed in the private section are initialized from script declarations common for all instrument types. The virtual functions in the public section must be overridden in subclasses to create the unique outlook of the particular instrument type. The middle section contains two functions that are used during the construction of the finite difference lattice: `getEvents()` must deliver the location of all relevant events (maturity, cashflow, barrier, early exercise or otherwise) on the time axis. The lattice is then guaranteed to match these events. `getBarriers()` may (but is not forced to) return the location in space of eventual knock-out barriers. Designing the lattice to match those increases numerical accuracy, but is not absolutely mandatory.

The semantic of the member variables and functions is summarized in the following paragraphs.

`m_nMaturity` indicates the number of days to maturity. No real calendar dates are supported yet by MtgLib for lattice-based evaluation.

```
class tClaim : public tObject {

    int m_nMaturity;

    double m_gMultiplier;

    bool m_bHasUpBarrier;
    double m_gUpBarrier;

    bool m_bHasDownBarrier;
    double m_gDownBarrier;

    bool m_bMonitor;

    tCashflow* m_Cashflow[...];

protected:

    virtual void getEvents( ... ) const;
    virtual void getBarriers( ... ) const;

public:

    virtual double payoff( tEngine& Engine );
    virtual double knockoutPayoff( tEngine& Engine );
    virtual double exercisePayoff( tEngine& Engine );

    virtual bool upBarrier( tEngine& Engine, double& gBarrier );
    virtual bool downBarrier( tEngine& Engine, double& gBarrier );

    virtual tExPolicy monitor( tEngine& Engine, double gUnitValue );
};
```

Figure 9.4: A crude sketch of the class definition of `tClaim`. Possible values of the enumeration type `tExPolicy` are `DontExercise`, `ForceExercise` and `MayExercise`

`m_gMultiplier` represents the position in the instrument and corresponds to λ.

`m_bMonitor` is a boolean flag that indicates whether the virtual member function `monitor()` should be used or not. This flag is set for American options.

`m_Cashflow` is a list of objects derived from the abstract class `tCashflow`, whose definition is given in Fig. 9.5. Each cashflow object implements additional, possibly space-dependent cashflow on a fixed date.

`payoff()` computes the payoff at maturity. The `tEngine` object whose reference is passed to `payoff()` and all other functions in `tClaim` provides information about the current state. For lattice instances, the `Engine` contains the current node instance $(j, i; L)$ on which `payoff()` must base its calculation. The values of s_j and t_i can be queried with `Engine.day()` and `Engine.factor()`, respectively. (Here we assume a one-factor model. Multi-factor models are also supported.) Engines are discussed below, in Sect. 9.2.1.

`knockoutPayoff()` computes the premium at knock-out. Unless overridden, this function always returns 0.

`exercisePayoff()` computes the payoff received at early exercise. By default, this function calls and returns the result of `payoff()`.

`upBarrier()` returns `true` if there is an up-and-out barrier for the current time slice, as determined by `Engine`. If also returns the barrier itself. Unless overridden, `upBarrier()` is defined as

```
bool tClaim::upBarrier( tEngine& Engine, double& gBarrier ) {
    if( m_bHasBarrier ) {
        gBarrier = m_gUpBarrier;
        return true;
    }
    return false;
}
```

`downBarrier()` works in an analoguous way for down-and-out barriers.

`monitor()` returns a safe estimate (!) of the local early-exercise policy. Possible return values are

- `DontExercise` if the instrument must not be exercised under the current state.
- `ForceExercise` if the instruments must be exercised at once.
- `MayExercise` if the instrument may or may not be exercised. Any further decision depends on the entire outlook and cannot be determined by the instrument alone.

Note the analogy with the concepts of continuation, exercise and corridor of uncertainty developed in Chapter 7. It is *not*, however, the task of `monitor()` to implement any of the speed-up techniques of Sect. 7.2. This is done by the compute engine in cooperation with the scenario object (see Sect. 9.1.5). The proper implementation for standard American options is thus simply

```
tExPolicy monitor( tEngine& Engine, double gUnitValue ) {
    return MayExercise;
}
```

`monitor()` can also be used to implement irregular barriers, bypassing the `upBarrier()` and `downBarrier()` member functions. In this case, continuation and knock-out regions are deterministic. They are implicitly located through the `DontExercise` and `ForceExercise` return values.

The definition of the supporting class `tCashflow` is given in Fig. 9.5. The member array `m_Cashflow` is examined during rollback just like the functions `upBarrier()` and `downBarrier()` are called for each time slice. The simplest instantiation of `tCashflow` would override the `generate()` member function with

```
double generate( tEngine& Engine ) {
   return c;
}
```

where `c` is some fixed coupon payment.

`tStdClaim` instantiates `tClaim` and supports instruments with the following orthogonal features:

- Call or put option?
- Linear or digital payoff?

```
class tCashflow {

    int m_nDay;

protected:

    virtual double generate( tEngine& Engine );
};
```

Figure 9.5: The definition of abstract class `tCashflow`. Cashflows are generated on day boundaries

- American or European?
- Up-and-out and/or down-and-out barrier?

Strike and maturity are the remaining properties. Its implementation is straightforward. `tCustomClaim` also instantiates `tClaim`, but does so in a customizable manner by parsing flexible script expressions for

- The payoff at maturity,
- the payoff at knock-out (if relevant),
- the payoff at early exercise (if relevant),
- the location of the knock-out barrier (time-dependent),
- a policy for determining early exercise,
- optional cashflows at fixed dates.

Figure 9.6 shows how these expressions are embedded into the parent class `tClaim`. The classes `tNumericalExpr` and `tExPolicyExpression` are not shown here; we merely note that both classes provide a member function `apply()` which is used to evaluate the expression. `payoff()`, for instance, is defined as follows:

```
double tCustomClaim::payoff( tEngine& Engine ) {
    if( m_pPayoff != 0 )
```

```
        return m_pPayoff->apply( Engine );
    return 0;
}
```

Expressions have access to the state information contained in `Engine`, through keywords such as `time`. The following script fragment, for instance, defines an up-and-out barrier call with strike 110 and barrier 120, where the barrier is only active for the first 50 days after settlement.

```
claim x {
    type custom, maturity 100,
    payoff { max( s - 110, 0 },
    up_and_out { if time < 50 then 125 endif }
}
```

9.1.2 Portfolios

Portfolios are collections of instruments. As such, they provide a generalized interface to some of the properties of instruments. The class `tPortfolio` is final; there are no subclasses.

A definition is given in Fig. 9.7. The meaning of the individual class members is as follows:

`m_Claim` References to all claims are collected here.

`m_Factor` In a multi-factor setting, different instruments may refer to different factors, or to the same factors in a different order. To establish a unique order of factors, the factors referenced in any of the instruments are collected in the array `m_Factor`.

`maturity()` The longest maturity of any of the instruments in the portfolio.

`claim()` To access individual instruments, this function must be used. The argument refers to the position of the instrument in `m_Claim`, which is sorted by maturity.

`getEvents()` This function in turn calls `tClaim::getEvents()` for each instrument and amalgamates the result, which is used in another place to calculate the discretization of the time axis for the lattice.

```
class tCustomClaim : public tClaim {

    class tCustomCashflow : public tCashflow {
    public:
        tNumericalExpr* m_pExpr;
        double generate( tEngine& Engine );
    };

    tNumericalExpr *m_pPayoff;
    tNumericalExpr *m_pKnockoutPayoff;
    tNumericalExpr *m_pExercisePayoff;

    tNumericalExpr *m_pUpBarrier;
    tNumericalExpr *m_pDownBarrier;

    tExPolicyExpr *m_pMonitor;

    double payoff( tEngine& Engine );
    double knockoutPayoff( tEngine& Engine );
    double exercisePayoff( tEngine& Engine );

    bool upBarrier( tEngine& Engine, double& gBarrier );
    bool downBarrier( tEngine& Engine, double& gBarrier );

    tExPolicy monitor( tEngine& Engine, double gUnitValue );
};
```

Figure 9.6: The definition of `tCustomClaim`, an instantiation of `tClaim`. The corresponding extension of `tCashflow` is defined locally

```
class tPortfolio {

    tClaim* m_Claim[...];
    tFactor* m_Factor[...];

public:

    int maturity() const;
    tClaim& claim( int nPos ) const;

    void getEvents( ... ) const;

    void getBarriers( const tFactor* pFactor,
        double Barrier[...] ) const;

    tRetCode matchFactors( const tModel& Model ) const;
};
```

Figure 9.7: A very condensed definition of `tPortfolio`

`getBarriers()` Calls `tClaim::getBarriers` for each instrument and combines the result, which is used by the algorithm in Fig. 5.4 to place the spatial levels of the lattice.

`matchFactors()` The factors in `m_Factor` are collected without knowledge of the particular model under which the portfolio is to be evaluated. If the elements of `m_Factor` are the factors $S_1, \ldots, S_n$, and the model makes use of factors $S'_1, \ldots, S'_m$, then $n = m$ must be asserted and the correct mapping found. This task is done by `matchFactors()`.

The functionality of portfolio objects is mostly used in the preparatory stage of evaluation. During actual rollback or simulation, instruments are directly accessed through the `claim()` member function.

9.1.3 Models

Just like instruments, models are supported through an abstract parent class, `tModel`, and child classes which provide the model-specific body. Figure 9.8 shows the dependencies.

The child class `tBSModel` is only used for lattice-based evaluation under the Black-Scholes model. The child classes `tHJMGaussianModel` and `tVasicekModel` (two levels removed from the parent class) support only Monte Carlo methods for fixed-income instruments. They are mentioned here only for the sake of completeness.

Class name	Purpose
`tModel`	Parent class (abstract)
`tBSModel`	One-factor Black-Scholes model
(`tHJMGaussianModel`)	For fixed-income
(`tShortRateModel`)	For fixed-income
(`tVasicekModel`)	For fixed-income

Figure 9.8: The model hierarchy. Fixed-income models are not discussed here and therefore parenthesized. They are, however, implemented for the calibrator whose architecture is briefly surveyed in Sect. 10.2

The definition of the class `tModel` is shown in Fig. 9.9. The semantics of the member components of `tModel` are as follows:

`m_Factor` The number of factors in the model is not predetermined. Factors are registered at creation by the child class. However, the number of factors must be known at the level of the parent class `tModel` (functions to query the number of factors and other trivial information are not included in the figure). For this reason, references to factors are stored in `m_Factor`. (Initial values of factors, however, are stored in the child class.)

`m_pCalendar` The calendar object is optional and at this time only supplies the scaling factor for the conversion between day and year-based quantities: the time-unit used in lattice calculations is one day, while model coefficients are usually quoted in their annualized form. If no calendar object is specified, a year of 365 days is assumed.

Before we proceed to describe public member functions, a remark on compute engines. The knowledge about the factor dynamics is encapsulated in the model. In particular, information about PDE's (for lattice-based methods) and SDE's (for simulation methods)

```
class tModel {

    tFactor* m_Factor[...];
    tCalendar* m_pCalendar;

public:

        // Functions for lattice-base methods:

    virtual tRetCode createEngine( const tScenario* pScenario,
        tFDEngine*& pEngine, tAccuracy m_nAccuracy );

    virtual tRetCode createSpaceAxis( tFDMethod nMethod,
        double gMaxDt, tSpaceAxis* Space[...],
        const tPortfolio* pPf = 0 );

        // Functions for simulation methods:

    virtual tRetCode createEngine( tMCEngine*& pEngine );

    virtual tRetCode createEvolution( const tPathSpace& PathSpace,
        tMCEngine::tEvolutionStub*& pEvolution ) const;
};
```

Figure 9.9: The fundamental members of the class `tModel`. A model that supports lattice-based methods must implement the first two virtual functions (`FD` = finite differences). A model that supports simulation methods must implement the last two virtual functions (`MC` = Monte Carlo)

can only be found in the model specification. The model provides this information by creating model-specific compute engines, which are based on the parent class `tEngine` and maintain and make accessible all the necessary runtime information during rollback or simulation. In some sense, the `run()` member function of `tFDEngine` or `tMCEngine` corresponds to the `main()` function in C++ programs, and its member variables contain the current global state of the computation.

Engines are discussed in more detail below, in Sect. 9.2.1. At this point we merely observe that different types of engines are created for lattice-based and simulation-based computation: `tFDEngine` and `tMCEngine` are the respective child classes.

`createEngine()`, first version Creates a compute engine for lattice-based evaluation. To create the proper engine, the model must know the scenario. Volatility shock scenarios, for instance, require a more complicated regimen for lattice instance creation than worst-case volatility scenarios (see Chapter 8). It is the engine which creates and maintains lattice instances.

The model must also know the selected speed-up technique for American options. This parameter, `nAccuracy`, is forwarded to the created engine. The name `nAccuracy` reflects the generality of the parameter; the engine chooses the speed-up technique that matches the parameter. Possible values are `Exact` (corresponding to the maintainance of corridors of uncertainty) and `Low` (corresponding to the collapsing of corridors of uncertainty).

`createSpaceAxis()` Creates the spatial discretization for lattice-based models, based on the algorithms in Figs. 5.4 and 5.5, and returns it in `Space`. `Space` is an array with one entry per factor. The parameter `nMethod` can take the values `Explicit` and `Implicit` and controls to what extent stability is a concern. `gMaxDt` corresponds to the input parameter $dt_{\max}$ in Fig. 5.4. The optional parameter `pPf` references a portfolio object. If present, the portfolio barriers are retrieved with `pPf->getBarriers()` and passed to the algorithm in Fig. 5.4.

`createEngine()`, second version This version creates a compute engine for simulation methods. At this point, simulation methods are not scenario based and therefore no additional arguments are required.

`createEvolution()` Simulation methods work by shooting one random path at a time in the so called "path space." The random path is then converted into the corresponding factor paths by calling `createEvolution()`. Evolutions are organized in their own separate class hierarchies, not shown here.

A historical note concerning the class `tCalendar`: lattice-base evaluation was implemented before actual calendar dates could be processed. In a future version, `tCalendar` will yield to features in MtgLib that already support true dates for Monte Carlo methods.

```
class tBSModel : public tModel {

    tDrift* m_pDiscount;
    tDrift* m_pCarry;
    tDrift* m_pMu;

    tVol* m_pVol;

    double m_gRoot;

public:

    tRetCode createSpaceAxis( tFDMethod nMethod, double gMaxDt,
        tSpaceAxis Space[...], const tPortfolio* pPf = 0 );

    tRetCode createEngine( const tScenario* pScenario,
        tFDEngine*& pEngine, tAccuracy nAccuracy );
};
```

Figure 9.10: The class `tBSModel`, the model body for one-factor Black-Scholes. Only lattice-based evaluation is supported

One instantiation of `tModel` is shown in Fig. 9.10. `tBSModel` only supports lattice-base evaluation for one-factor Black-Scholes with time-varying coefficients. The member components have the following interpretation:

`m_pDiscount` References an interest-rate term structure for the parameter r of (4.4.8). The class `tDrift` is explained in Sect. 9.1.4.

`m_pCarry` References a term structure for the dividend rate or foreign interest rate, depending on whether the underlying asset is a stock or an exchange rate.

`m_pMu` The no-arbitrage drift parameter. References a term structure for the difference between `m_pDiscount` and `m_pCarry`.

`m_pVol` References a volatility term structure that may exhibit uncertainty. The object `*m_pVol` contains upper and lower bounds for the local volatility for each time slice.

`m_gRoot` The initial value S_0 of the underlying asset. The lattice is constructed such that $s_0 = S_0$.

`createSpaceAxis()` Finds the stable spatial discretization that matches the barriers of the (optional) portfolio parameter and returns exactly one object instance of the class `tGeoSpaceAxis`. The prefix "`Geo`" means geometric Brownian motion. The member function `prepare()` of this class, called in `createSpaceAxis()`, uses the algorithm in Fig. 5.4. Figure 9.11 contains a skeleton of `createSpaceAxis()`.

`createEngine()` Creates the lattice-based compute engine appropriate for the scenario parameter. Two types of compute engines for one-factor Black-Scholes are currently implemented: `tGeoEngine` for worst-case volatility scenarios (class `tWorstCase`) and `tShockEngine` for volatility shock scenarios (class `tShockScenario`). Run-time type information (RTTI) is used to distinguish these cases. The function is outlined in Fig. 9.12.

(Remark: the way `createEngine()` is coded leads to an extensibility problem. Prolongued sequences of conditional statements guarded by dynamic down-casts should be avoided. There are a handful of spots in MtgLib where this problem occurs.)

9.1.4 Model coefficients

Constant model coefficients such as the initial value of the underlying asset are handled by the model class itself: `tBSModel::m_gRoot` is an example. Other model coefficients have more structure and deserve their own classes. As coefficients depend on the actual

```
tRetCode tBSModel::createSpaceAxis( tFDMethod nMethod,
    double gMaxDt, tSpaceAxis Space[...], const tPortfolio* pPf )

{
    double gMinVol, gMaxVol, gMinMu, gMaxMu;

    m_pVol->getFwdRange( gMinVol, gMaxVol );
    m_pMu->getFwdRange( gMinMu, gMaxMu );

    tGeoSpaceAxis* p = new tGeoSpaceAxis;

    if( pPf != 0 ) {
        double Barrier[...];

        pPf->getBarriers( m_Factor[0], Barrier );
        p->prepare( nMethod, m_gRoot, gMinVol, gMaxVol,
                    gMinMu, gMaxMu, Barrier );
    }
    else {
        p->prepare( nMethod, m_gRoot, gMinVol, gMaxVol,
                    gMinMu, gMaxMu );
    }

    Space.append( p );
    return OK;
}
```

Figure 9.11: A sketch of the member function `tBSModel::createSpace()`. The class `tGeoSpaceAxis` is derived from `tSpaceAxis` and supports geometric Brownian motion models. It calls `tGeoSpaceAxis::prepare()`, which implements the algorithm in Fig. 5.4

```
tRetCode tBSModel::createEngine( const tScenario* pScenario,
    tFDEngine*& pEngine, tAccuracy nAccuracy )

{
    if( dynamic_cast<const tShockScenario*>( pScenario ) != 0 ) {
        pEngine = new tShockEngine;
    }
    else
    if( dynamic_cast<const tWorstCase*>( pScenario ) != 0 ) {
        pEngine = new tGeoEngine;
    }
    else {
        return NOT_AVAILABLE;
    }

    pEngine->setAccuracy( nAccuracy );
    return OK;
}
```

Figure 9.12: The member function `tBSModel::createEngine()`. `tShockEngine` and `tGeoEngine` are both derived from `tFDEngine`. Both classes support one-factor geometric Brownian motion models (the "`Geo`" prefix)

model, corresponding classes may be quite diverse, and form a collection rather than a strictly hierarchical class tree.

Figure 9.13 shows the inheritance relations for the model coefficient classes currently supported in MtgLib. The `tTermStruct` hierarchy was developed first and does not support real calendar dates; real calendar dates are only handled on the level of `tDrift` and `tVol` and below. The classes `tHJMTermStruct` and `tShortRateTermStruct`, on the other hand, were developed with support for real calendar dates already in mind. They are used as model coefficient classes for interest-rate models and not discussed further in this thesis.

Class name	Purpose
`tTermStruct`	Parent term structure class (abstract)
`tLinTermStruct`	Term structure for linear coefficient
`tSqTermStruct`	Term structure for quadratic coeffiient
`tDrift`	General term structure class for drift (abstract)
`tStepDrift`	Piece-wise constant drift
`tVol`	General term structure class for volatility (abstract)
`tStepVol`	Piece-wise constant volatility
(`tHJMTermStruct`)	For fixed-income
(`tShortRateTermStruct`)	For fixed-income

Figure 9.13: Classes for model coefficients. These classes are model-dependent; they do not share a common base class

The Base Class `tTermStruct`

`tTermStruct` is the core class for piece-wise constant time-varying model coefficients. If α is a model coefficient for an n-factor model with factors $X_1, \ldots, X_n$, then

$$\alpha(X_1(t), \ldots, X_n(t), t) = \alpha(t) = c_k \tag{9.9.1}$$

with k such that $t \in [t_{k-1}, t_k]$ for some times slices t_{k-1} and t_k.

The basic functionality of `tTermStruct` is to compute the values

$$\alpha(t) \tag{9.9.2}$$

and

$$\frac{1}{b-a}\int_a^b \alpha(t)\,dt \tag{9.9.3}$$

fast. The granularity of t, a and b is assumed to be one day.

`tTermStruct` can be used for both linear and quadratic parameters, such as drift and volatility. This generality is achieved by introducing a scaling function ϕ and replacing (9.9.3) with

$$\phi^{-1}\left(\frac{1}{b-a}\int_a^b \phi(\alpha(t))\,dt\right) \tag{9.9.4}$$

For linear term structures, ϕ is the identity. For quadratic term structures such as volatility, $\phi(x) = x^2$.

Assume jumps occur at jump points $u_0, u_1, \ldots, u_M$, where each u matches some time sline t_i (most likely $M \ll N$). To compute (9.9.2) and (9.9.4) fast, the following quantities are maintained for each jump point u:

$$\begin{aligned} \texttt{m\_gFwd} &= \alpha_u \\ \texttt{m\_gFwd2} &= \phi(\alpha_u) \\ \texttt{m\_gImp} &= \int_0^u \phi(\alpha(t))\, dt \end{aligned} \tag{9.9.5}$$

(9.9.4) can then be computed for all intermediate time slices t with $O(1)$ overhead, by using `m_gImp` of the previous sample point u as a base and adding `m_gFwd2`, multiplied by the number of days between u and t. Subsequent normalization is straightforward.

The remaining problem is to locate the previous jump point u for a given intermediate time slice t, if t is not also a jump point. A shallow forest of bounded depth does the trick for `tTermStruct`. At the highest level of the forest, each node represents 100 consecutive days. Only if such a period contains a jump point is refinement necessary: the corresponding node branches into 10 child nodes, each covering a period of 10 days, and so on. The memory requirements for this data structure are still linear in the number of days covered by the term structure, but nevertheless reduced 100-fold compared to day-by-day storage if the number of jumps is small.

Figure 9.14 summarizes the important components of class `tTermStruct`. Their interpretation is as follows:

`m_Spec` The nested type `tSpec` describes one jump point. The member variable `m_nUnit` locates the jump point in time. `m_gFwd`, `m_gFwd2` and `m_gImp` are defined in (9.9.5). `tMap` is a template for the efficient implementation of the shallow forest data structure mentioned above. `m_Spec` is built as jump points are added with `addForward()` and `addImplied()`.

`scaleUp()` The scaling function ϕ.

`scaleDown()` The inverse of the scaling function ϕ.

`addForward()` **and** `addImplied()` The term structure object is constructed by calling `addForward()` or `addImplied()` for each jump point. When all jump points have

```
class tTermStruct {

    struct tSpec {
        int m_nUnit;
        double m_gFwd;
        double m_gFwd2;
        double m_gImp2;
    };

    tMap<tSpec> m_Spec;

protected:

    virtual double scaleUp( double gFwd ) const;
    virtual double scaleDown( double gFwd ) const;

public:

    tRetCode addForward( int nMaturity, double gFwd );
    tRetCode addImplied( int nMaturity, double gImp );

    void getFwdRange( double& gMin, double& gMax ) const;

    double forward( int nUnit ) const;
    double forward( int nFromUnit, int nToUnit ) const;

    int constantUntil() const;
    int certainUntil( const tTermStruct& TS ) const;
};
```

Figure 9.14: The skeleton of class `tTermStruct`. Although the basic time unit is one day in most cases, the code itself is independent of the concrete time unit. `m_nUnit`, `nUnit`, `nFromUnit` and `nToUnit` are therefore used instead of `m_nDay`, `nDay`, `nFromDay` and `nToDay`

been specified, the constant term structure rate between each pair of jump points is determined as follows:

- If the jump point $u = t_k$ has been added with `addForward()`, then the rate between the previous jump point and u is simply set to the actual value of the `gFwd` parameter.
- If the jump point $u = t_k$ has been added with `addImplied()`, then the rate between the previous jump point and u is set to the rate which makes the integrated rate equal to the actual value of `gImp`, i.e.

$$\phi^{-1}\left(\frac{1}{u}\int_0^u \phi(\alpha(t))\,dt\right) = \texttt{gImp}$$

  Note that this calculation can fail if ϕ is not the identity!

The parameters `nUnit`, `nFromUnit` and `nToUnit` indicate the endpoint of the respective time unit, where time units are counted from zero. This makes the extension to fractional parameters consistent.

`getFwdRange()` Once all jump points have been added and the term structure has been finalized, the oscillation of the term structure can be determined by calling `getFwdRange()`. This information is important for the construction of a stable lattice under the explicit finite difference scheme.

`forward()` The one- and two-parameter versions correspond to (9.9.2) and (9.9.4), respectively.

`constantUntil()` It may be useful to know the length of the initial constant segment of the term structure. `constantUntil()` returns this information.

`certainUntil()` It is also useful to know wether a certain configuration of model coefficients exhibits uncertainty at all. `certainUntil()` tests this by comparing the current term structure with the argument `TS`, and returning the length of the initial segment on which they agree.

The term structure extrapolates beyond the first and last jump points by propagating the rates of the first and last constant segment to $-\infty$ and ∞.

Classes Derived from `tTermStruct`

The child class `tLinTermStruct` is a straightforward instantiation of `tTermStruct`, with ϕ being the identity. The child class `tSqTermStruct` is mildly more complicated; Figure 9.15 shows its definition, together with the implementation of the scaling function and its inverse.

```
class tSqTermStruct : public tTermStruct {

    double scaleUp( double gFwd ) const {
        return gFwd * gFwd;
    }

    double scaleDown( double gFwd ) const {
        return sqrt( gFwd );
    }
};
```

Figure 9.15: The child class `tSqTermStruct`. Shown are both declaration and definition of the scaling function and its inverse

Classes with `tTermStruct` Components

The classes `tDrift` and `tVol` are more interesting. They are independent of any actual implementation of the drift or volatility term structure and offer a standard query interface for forward rates or volatilities. Piece-wise constant realizations of drift or volatility term structure are obtained by combining the `tDrift` or `tVol` shell with `tTermStruct` as "meat." The formula is

$$\texttt{tDrift} + \texttt{tLinTermStruct} = \texttt{tStepDrift}$$

and

$$\texttt{tVol} + 2 \times \texttt{tSqTermStruct} = \texttt{tStepVol}$$

`tLinTermStruct` and `tSqTermStruct` contribute as member components. The class `tStepDrift` contains a `tLinTermStruct` object and forwars queries to it; `tStepVol` contains two `tSqTermStruct` objects to allow for uncertainty, and forwards queries to them.

Composition makes more sense than multiple inheritance in this case, because the number of term structure objects is variable.

```
class tDrift {

public:

    tDrift();

    virtual void getFwdRange( double& gMin, double& gMax ) const;

    virtual double forward( int nUnit ) const;
    virtual double forward( int nFromUnit, int nToUnit ) const;

    virtual double implied( int nMaturity ) const;

    virtual int constantUntil() const;
};
```

Figure 9.16: Class `tDrift` is an abstract interface for drift coefficients. Piece-wise constant drift term-structures are one possible instantiation of `tDrift`

Figure 9.16 shows the abstract `tDrift` interface. All virtual functions are pure. A call to `implied()` is equivalent to a call to `forward()` with the first parameter set to zero.

Figure 9.17 shows the `tVol` interface. Again, all virtual functions are pure. However, any concrete instantiation of `tVol` is expected to initialize the following member variables correctly:

`m_nConstantUntil` The length of the initial period during which the volatility is constant (there is no uncertainty during that period!).

`m_nCertainUntil` The length of the initial period during which the volatility is certain. Necessarily, `m_nConstantUntil` $\leq$ `m_nCertainUntil`.

Both `m_nConstantUntil` and `m_nCertainUntil` can be retrieved from the object with trivial functions not shown in the figure.

```
class tVol {

    int m_nConstantUntil;
    int m_nCertainUntil;

public:

    virtual void getFwdRange( double& gMin, double& gMax ) const;

        // Return a single value:
    virtual double forward( int nUnit ) const;
    virtual double forward( int nFromUnit, int nToUnit ) const;

    virtual double implied( int nMaturity ) const;

        // Return a range of value:
    virtual void forward( int nUnit, double& gMin,
        double& gMax ) const;
    virtual void forward( int nFromUnit, int nToUnit, double& gMin,
        double& gMax ) const;

    virtual void implied( int nMaturity, double& gMin,
        double& gMax ) const;
};
```

Figure 9.17: Class `tVol` is an abstract interface for volatility coefficients. Piece-wise constant volatility term-structures are one possible instantiation of `tVol`

tVol has two sets of volatility retrieval functions: the first set returns a single value, the second a range of values in reference parameters gMin and gMax. If there is no uncertainty, both versions are equivalent. If there is uncertainty, however, and the volatility bounds differ, then only the second set of retrieval functions is required to return the original volatility bounds defined during construction of the object faithfully. The first set of retrieval functions may return any value, the arithmetic average between the minimum and maximum being one example and some additional prior volatility another.

It was mentioned earlier that tDrift and tVol are capable of handling real calendar dates. These features are ommitted in Figs. 9.16 and 9.17.

Piece-wise linear drift and volatility coefficients are finally realized in the classes tStepDrift and tStepVol ("Step" for step function). Figure 9.18 shows the definition and implementation of tStepDrift, which basically acts as a proxy for its tLinTermStruct member object. Figure 9.19 shows the definition of tStepVol, whose implementation is only slightly less trivial. The two-parameter forward() function, for instance, is implemented as

```
void tStepVol::forward( int nFromUnit, int nToUnit, double& gMin,
    double& gMax ) const

{
    gMin = m_MinTermStruct.forward( nFromUnit, nToUnit );
    gMax = m_MaxTermStruct.forward( nFromUnit, nToUnit );
}
```

The objects v and r in the script shown in Fig. 9.2 are automatically mplemented as tStepVol and tStepDrift instances, respectively. tStepVol and tStepDrift instances are also used to model the volatility, interest rate, and dividend rate or foreign exchange rate in class tBSModel (see Fig. 9.10).

9.1.5 Scenarios

Scenario objects perform an "advisory" function for lattice-based evaluation. They are used by compute engines derived from tFDEngine to determine locally how to select the uncertain model coefficients. They also control the assignment of lattice nodes to continuation and exercise regions, and the corridor of uncertainty. Figure 9.20 shows the class hierarchy.

```
class tStepDrift : public tDrift {

    tLinTermStruct m_TermStruct;

public:

    void getFwdRange( double& gMin, double& gMax ) const {
        m_TermStruct.getFwdRange( gMin, gMax );
    }

    double forward( int nUnit ) const {
        return m_TermStruct.forward( nUnit );
    }

    double forward( int nFromUnit, int nToUnit ) const {
        return m_TermStruct.forward( nFromUnit, nToUnit );
    }

    double implied( int nMaturity ) const {
        return m_TermStruct.implied( nMaturity );
    }

    int constantUntil() const {
        return m_TermStruct.constantUntil();
    }
};
```

Figure 9.18: Piece-wise linear drift coefficients are of type `tStepDrift`, essentially based on the functionality of `tLinTermStruct`

```
class tStepVol : public tVol {

    tSqTermStruct m_MinTermStruct;
    tSqTermStruct m_MaxTermStruct;

public:

    void getFwdRange( double& gMin, double& gMax ) const;

    double forward( int nUnit ) const;
    double forward( int nFromUnit, int nToUnit ) const;
    double implied( int nMaturity ) const;

    void forward( int nUnit, double& gMin, double& gMax ) const;
    void forward( int nFromUnit, int nToUnit,
        double& gMin, double& gMax ) const;

    void implied( int nMaturity, double& gMin, double& gMax ) const;
};
```

Figure 9.19: Piece-wise linear, possibly uncertain volatility coefficients are of type `tStepVol`. The implementation of the member functions is almost as trivial is those of `tStepDrift`

The Base Class

Figure 9.21 shows how these two tasks are reflected in the class definition. Before indiviual components are discussed, however, we need to clarify the usage of *tags*.

In Defs. 5.1 and 8.3, the notion of (extended) lattice signatures has been introduced to uniquely identify individual lattice instances in the collection of lattice instances maintained for the particular problem. To be able to handle a wide set of scenarios, a concrete scenario must provide a way to translate a scenario-dependent lattice signature $(\mathbf{X}, \lambda, \mathrm{optseq})$ into a regularized signature which is easier to process.

Definition 9.1 (Regularized lattice signature). *Let* $(\mathbf{X}, \lambda, \mathrm{optseq})$ *be the pattern of signatures for lattice instances for a concrete scenario, with* optseq *denoting an optional sequence of coefficients. Then triples of the form* $(\mathbf{X}, \lambda, 2m)$*,* $m \geq 0$*, are called* regularized

Class name	Purpose
`tScenario`	Parent class (abstract)
`tWorstCase`	Worst-case volatility scenario
`tShockScenario`	Volatility shock scenario

Figure 9.20: Scenario classes. Each class extends the functionality of its parent

lattice signatures *for the scenario if there is an unambiguous mapping between the two patterns. Furthermore, for the root lattice instance from which the final result is retrieved,* $2m = 0$ *must hold.* $2m$ *is called the signature* tag.

Compute engines use this regularized form to manage the storage of lattice instances. For worst-case volatility scenarios, optseq is empty, and the tag is always zero. For volatility shock scenarios, optseq = (τ, ξ, δ). It is straightforward to translate triples (τ, ξ, δ) into integer tags $2m$. Arranging lattice instances as shown in Fig. 8.3 and counting them from top to bottom, and from left to right, yields a valid sequence.

Tags are even-numbered. Odd-numbered tags are also used internally, reversing the evaluation view-point from sell-side to buy-side, or vice versa. Thus, if $\hat{F}(L)$ is computed on a lattice instance with regularized lattice signature $(\mathbf{X}, \lambda, 2m)$, then $-\hat{F}(L')$ is computed on lattice instance L' with signature $(\mathbf{X}, -\lambda, 2m+1)$. This reversal is necessary to compute the boundaries of corridors of uncertainty, namely $\hat{F}(L_n^U)$ and $-\hat{F}(L_D^n)$. (Recall that the signatures of L_n^U and L_n^D are $(X_n, 1)$ and $(X_n, -1)$, respectively. See Sect.7.2.1.)

With this information, the member elements of `tScenario` are as follows:

`m_nPosition` The nested type `tPosition` allows to flip the evaluation view-point *globally.* So far in this thesis, worst-case scenarios have always beed regarded from the seller's point of view, and prices have been maximized to cover the worst-case liability. To take the buyer's position, on the other hand, means to seek the smallest price to pay, in order to avoid loosing when the market behaves adversely. This changes the maximization to a minimization procedure: the "$\sup_{\sigma \in \mathcal{C}}$" turns into an "$\inf_{\sigma \in \mathcal{C}}$" in (4.4.10) and all the similar equations that follow. Simlarly, the "$\min_{A \subseteq A(L)} \max_{B \subseteq B(L)}$" first introduced in Def. 7.8 and occuring throughout Chapter 7 changes to "$\max_{A \subseteq A(L)} \min_{B \subseteq B(L)}$." This is because the interpretation

```
class tScenario {

public:

    enum tPosition {
        Buyer,
        Seller
    };

private:

    tPosition m_nPosition;

public:

    virtual bool underControl( double gMultiplier );

    virtual void refineExPolicy( tFDEngine& Engine, int nBaseTag,
        int nIndex, double gDontExValue, double gExValue,
        double gMultiplier, tExPolicy& nExPolicy );

    virtual double selectVol( int nTag, double gGamma,
        double gMin, double gMax );

    virtual bool endureOver( int nTag, double gNewTotal,
        double gOldTotal );

    virtual bool chooseOver( int nTag, double gNewTotal,
        double gOldTotal );
};
```

Figure 9.21: The definition of the abstract class `tScenario`. In general, worst-case scenarios are asymmetric for the buy- and sell-side. Which particular viewpoint is to be adopted is indicated by the value of `m_nPosition`

of λ changes: positive λ now indicates a long position, whereas negative λ indicates a short position.

Any child class of `tScenario` is expected to initialize `m_nPosition`. For consistency with the earlier text, we assume here and below that `m_nPosition` = `Seller`.

`underControl()` If, for a given instrument X_n in the portfolio under consideration, $\lambda_n > 0$, then X_n is held short and not under the control of the agent. If $\lambda_n < 0$ the instrument is held long; potential early exercise is under control of the agent. `underControl()` interpretes its parameter `gMultiplier` as λ_n and returns whether the corresponding position enables the agent to exercise control.

Since the situation is reversed if the global view-point changes from the sell-side to the buy-side, a separate function is justified.

`refineExPolicy()` The class `tClaim` relies on the member function `monitor()` to produce an initial assessment of the local early exercise options for an individual instrument (see Sect. 9.1.1). If `monitor()` returns `DontExercise` or `ForceExercise`, the current lattice-instance node is assigned to the continuation respectively exercise region of the instrument for good. If `monitor()` returns `MayExercise`, as it does for standard American options, then the `tScenario` object is asked in turn to try to make a definite statement. Only when the `tScenario` object returns `MayExercise` as well is the current node assigned to the corridor of uncertainty of the instrument.

The safest policy is thus to return `MayExercise` throughout. However, as the `tScenario` object has access to other lattice instances through the `Engine` argument, a more advanced strategy such as described in Sects. 7.2.1 and 7.2.2 may be employed. This must be done in `tScenario`'s child classes by overriding `refine-ExPolicy()`.

The argument `nBaseTag` corresponds to the tag $2m$ of the regularized signature of the current lattice L. `nIndex` is the index of the claim in the portfolio, to be used as argument for `tPortfolio::claim()`. `gDontExValue` is the unit value of the instrument obtained through rollback. `gExValue` is the unit value of the instrument returned by `tClaim::exercisePayoff()` for the current node. `gMultiplier` is the number of contracts, and `nExPolicy` is an in-out parameter, initially set to `MayExercise`.

`selectVol()` The local volatility is selected between `gMin` and `gMax`. The actual implementation of the prototype in Fig. 9.21 may base its selection on `gGamma`, the finite difference approximation of $\frac{\partial^2}{\partial S^2}\hat{f}$, where $\hat{f}(S_t, t; L) = \hat{F}_t(L \mid S_t)$. The regularized tag of L is `nTag`. Two restrictions are immediately obvious:

- `selectVol()` works only for a one-factor model;
- selection schemes that require information beyond gamma cannot be realized.

The uncertain volatility models of Sects. 4.2.1, 4.2.2 and, with the introduction of a prior volatility parameter, 4.2.3, are all feasible, though.

`endureOver()` **and** `chooseOver()` This pair of functions is substituted for the max and min operators in the expression "$\min_{A \subseteq A(L)} \max_{B \subseteq B(L)}$" that occurs throughout Chapter 7 and in Figs. 7.4 and 7.6. The functions are folded over a sequence of values; `gOldTotal` represents the value selected thus far, and `gNewTotal` represents the new candidate. If the new value is to be selected over the old value, the function returns `true`.

Formula (7.7.95) of Sect 7.2.3 is used as a recipe for folding `endureOver()` over the arguments of the max operator and `chooseOver()` over the arguments of the min operator. The function names reflect the absence respectively presence of control by the agent.

Derived Classes

`tWorstCase` is immediately derived from `tScenario`. Its definition is shown in Fig. 9.22. Figure 9.23 shows a listing of the member function `selectVol()`, and Fig. 9.24 shows the implementation of `endureOver()`. `chooseOver()` is implemented in an analogue fashion.

Figure 9.25, finally, contains an outline of the function `refineExPolicy()`. The function consists of two branches, the first being executed if the pricing problem is linear or the corridors of uncertainty ought to be collapsed (in which case `gDontExValue` is the partial derivative of the worst-case value with respect to $\lambda_{\texttt{nIndex}}$). The second branch maintains the corridor of uncertainty by looking up the singleton portfolios $(X_{\texttt{nIndex}}, 1)$ and $(X_{\texttt{nIndex}}, -1)$. The function `getClaim()` of class `tFDEngine` does just that. This branch is an implementation of the algorithm in Fig. 7.7. Note that in addition to

```
 1 class tWorstCase : public tScenario {
 2
 3 public:
 4
 5     bool underControl( double gMultiplier );
 6
 7     void refineExPolicy( tFDEngine& Engine, int nBaseTag,
 8         int nIndex, double gDontExValue, double gExValue,
 9         double gMultiplier, tExPolicy& nExPolicy );
10
11     double selectVol( int nTag, double gGamma, double gPrior,
12         double gMin, double gMax );
13
14     bool endureOver( int nTag, double gNewTotal, double gOldTotal );
15     bool chooseOver( int nTag, double gNewTotal, double gOldTotal );
16 };
```

Figure 9.22: `tWorstCase` instantiates the abstract member functions of `tScenario`

implementing the algorithm, `refineExPolicy()` must reverse all selection criteria if the global view-point is changed to the buy-side. This is done by setting the corrective constant `nTag` in line 16.

Only the case where `gMultiplier` is non-negative is shown in Fig. 9.25. THe other case is handled symmetrically.

The class `tShockScenario` is a true extension of `tWorstCase`. No function of `tWorst-Case` is overridden, as volatility-shock scenarios essentially only broaden the candidate set of volatilities. Figure 9.26 shows the definition of `tShockScenario`.

The interpretation of the member variables of `tShockScenario` follows Def. 8.2:

`m_nDuration` The duration parameter $d \geq 1$.

`m_nPeriodicity` The periodicity parameter $p \geq 1$.

`m_nFrequency` The frequency parameter $f \geq 1$

These variables are retrieved during rollback by a specialized compute engine of class `tShockEngine`. Since engines are passed only base references to objects of class `tScenario`

```
double tWorstCase::selectVol( int nTag, double gGamma,
    double gMin, double gMax )

{
    if( nTag % 2 == 0 ) {
        switch( position() ) {
            case Buyer :
                return ( gGamma <= 0 ) ? gMax : gMin;
            case Seller :
                return ( gGamma >= 0 ) ? gMax : gMin;
        }
    }
    switch( position() ) {
         case Buyer :
            return ( gGamma >= 0 ) ? gMax : gMin;
         case Seller :
            return ( gGamma <= 0 ) ? gMax : gMin;
    }
}
```

Figure 9.23: The body of the function `tWorstCase::selectVol()`. Depending on tag and global view-point, the function bases its decision on convexity respectively concavity. Recall that odd `nTag` indicates that $-\hat{F}(L')$ is being computed, where the signature of L' is $(\mathbf{X}, -\lambda, \texttt{nTag})$ and there exists a lattice instance L with signature $(\mathbf{X}, \lambda, \texttt{nTag} - 1)$. Since the negative signs are not actually applied, all comparisons need to be inverted

during initialization, a down-cast must be performed by `tShockEngine` to access these values. This is done safely with RTTI support.

Remark: the volatility shock scenario introduces additional events which should be matched by the lattice. For that purpose, `tScenario` provides an (initially empty) method `getEvGenerator()` that is overridden by `tShockScenario`. This function is not shown in the figures.

9.1.6 Numerical methods

MtgLib provides two ways to evaluate portfolios numerically: based on lattices, and with Monte Carlo simulation. Lattice-based evaluation is better supported at the time of this

```
bool tWorstCase::endureOver( int nTag, double gNewTotal,
    double gOldTotal )

{
    if( nTag % 2 == 0 ) {
        switch( position() ) {
            case Buyer :   // minimize
                return gNewTotal < gOldTotal;
            case Seller :  // maximize
                return gNewTotal > gOldTotal;
        }
    }
    switch( position() ) {
        case Buyer :   // maximize
            return gNewTotal > gOldTotal;
        case Seller :  // minimize
            return gNewTotal < gOldTotal;
    }
}
```

Figure 9.24: The body of the function `tWorstCase::endureOver()`. Depending on tag and global view-point, the function mimicks a max or min operator

writing and the exclusive topic in the earlier parts of this thesis. For this reason, we focus exclusively on the lattice-based facilities of MtgLib in the following paragraphs.

The classes that support lattice-based numerical evaluation fall into two categories: those that define the lattice template and manage lattice instances, and those that actually perform the finite difference rollback. Figure 9.27 shows both categories. The first group is capable of handling multi-factor models; the second group is not.

Some of the classes in Fig. 9.27 might as well be labeled "internal", since they are not directly visible through the scripting interface. They are listed here because of their proximity to the hierarchy of lattice-related classes, which is visible through the scripting interface.

```
 1 void tWorstCase::refineExPolicy( tFDEngine& Engine, int nBaseTag,
 2     int nIndex, double gDontExValue, double gExValue,
 3     double gMultiplier, tExPolicy& nExPolicy )
 4
 5 {
 6     double gValue;
 7
 8     if( Engine.isLinear() || Engine.accuracy() == Low ) {
 9         if( gExValue > gDontExValue )
10             nExPolicy = xForceExercise;
11         else
12             nExPolicy = xDontExercise;
13     }
14     else {
15         if( gMultiplier >= 0 ) {
16             int nTag = ( position() == Buyer ) ? 0 : 1;
17
18             if( gExValue > gDontExValue ) {
19                 Engine.getClaim( nIndex,
20                     nBaseTag + 1 - nTag, gValue );
21                 if( gExValue > gValue )
22                     nExPolicy = xForceExercise;
23             }
24             else {
25                 Engine.getClaim( nIndex, nBaseTag + nTag, gValue );
26                 if( gExValue <= gValue )
27                     nExPolicy = xDontExercise;
28             }
29         }
30         else {
31             // the other case is symmetric
32         }
33     }
34 }
```

Figure 9.25: An outline of the function `refineExPolicy()` of class `tWorstCase`. The constant `Low` in line 8 corresponds to the strategy to collapse corridors of uncertainty; the `else` branch maintains corridors of uncertainty

```
class tShockScenario : public tWorstCase {

    int m_nDuration;
    int m_nPeriodicity;
    int m_nFrequency;
};
```

Figure 9.26: The class `tShockScenario` merely adds the parameters of a volatility shock scenario as defined in Def. 8.2

Lattice Templates and Instances

The class `tLattice` describes the layout of the lattice. Number of factors, time discretization, space discretization, shape (tree or box) and space trimming determine the layout. Figure 9.28 shows how `tLattice` is defined.

The interpretation of the individual member components of `tLattice` is as follows:

`m_pModel` The lattice template needs to know about the model in order to create the entries for `m_Space`. It uses the function `tModel::createSpaceAxis()` for that purpose.

`m_bIsBox` The lattice can have the shape of a rectangular grid, or that of a tree, with the root labeled with S_0. This flag determines whether the rectangular grid shape is used.

`m_bIsTrimmed` **and** `m_gTrimDev` In order to reduce the running time, the lattice may be trimmed symmetrically at the outer regions. `m_bIsTrimmed` determines whether this is done. `m_gTrimDev` indicates the number of standard deviations after which the trimming should occur. The default values are `true` and `3.5`. See Parás (1995) and the comment at the beginning of Sect. 5.1 for more details.

`m_nMethod` Can be either `Explicit` or `Implicit` and is used, among other things, as argument in calls to `tModel::createSpaceAxis()`, where it is used to ensure stability.

`m_Bounds` Determines the dimensions of the lattice layout when viewed as $(n+2)$-dimensional hypercube, where n is the number of factors, the $(n+1)$-st dimen-

Class name	Purpose
`tLattice`	Lattice template
`tTimeAxis`	Discretization of time
`tSpaceAxis`	Discretization of space for one factor (abstract)
`tGeoSpaceAxis`	Discretization of space based on geometric Brownian motion
`tLatticeInstance`	Lattice instance, what else?
`tOFSolver`	One-factor finite difference solver (abstract)
`tOFExplicit`	Explicit finite difference solver
`tGeoExplicit`	Explicit solver for models based on geometric Brownian motion
`tOFImplicit`	Crank-Nicholson finite difference solver
`tGeoImplicit`	Crank-Nicholson solver for models based on geometric Brownian motion
`tGeoSolver`	Additional base class for `tGeoExplicit` and `tGeoImplicit` (abstract)

Figure 9.27: The collection of classes that work together to support lattice-based evaluation. The prefix "`OF`" stands for one-factor. `tGeoExplicit` and `tGeoImplicit` have two parent classes and are thus a case of multiple inheritance

sion is time and the last dimension is the combined gradient + total value-vector. `m_Bounds` is basically a sequence of pairs of upper and lower index bounds; the class `tArrayBounds` is not shown.

`m_Space` An array with one entry per factor. `tSpaceAxis` is an abstract class; concrete instantiations work with particular models. Currently implemented is only `tGeoSpaceAxis`, which complements the model class `tBSModel`. Note that the dimension of each space axis must be consistent with the corresponding information in `m_Bounds`.

`m_pTime` The discretization of the time axis, which may be non-uniform. The time axis is only finalized after all the space axes have been created, for the required cap on the

```
class tLattice {

    tModel* m_pModel;

    bool m_bIsBox;
    bool m_bIsTrimmed;
    double m_gTrimDev;

    tFDMethod m_nMethod;

    tArrayBounds m_Bounds;

    tSpaceAxis* m_Space[...];
    tTimeAxis* m_pTime;

public:

    tOFSolver* createOFSolver();

    tRetCode createInstance( tPortfolio* pPf,
        const tSignature* pSig, int nTag,
        tLatticeInstance*& pInstance ) const;
};
```

Figure 9.28: The class `tLattice` defines the layout of the lattice (the lattice template), from which lattice instances are created by calling `createInstance()`

size of the largest time step can only then be known. (See output *dt* of the algorithm in Fig. 5.4.) The class `tTimeAxis` is final. `tTimeAxis` is purely mathematical and does not support real calendar dates.

`createOFSolver()` Finite difference solvers are discussed in the next section. This function creates a solver for one-factor models. Its implementation is simple: since the finite difference approximation to partial derivatives depend on the geometry of the discretization as well as the underlying stochastic process, the request is forwarded to the space axis, which knows about these properties:

```
tOFSolver* tLattice::createOFSolver()
```

```

{
    return m_Space[0]->createOFSolver();
}
```

Multi-factor solvers are not implemented.

`createInstance()` The lattice template is also used to create lattice instances of it. The signature of the new lattice instance is implied by the arguments `pPf`, `pSig` and `nTag`. `tSignature` is implemented as a bitfield; its precise definition is not shown.

The lattice instances created by `createInstance()` in `tLattice` belong to the class `tLatticeInstance`, a very condensed definition of which is shown in Fig. 9.29.

Lattice instances do not allocate memory for the entire grid, but only for two adjacent hyperplanes, cut perpendicular to the time axis. This is standard procedure for memory-aware implementations of one-level finite-difference schemes and tree methods. One hyperplane contains the values for the previously processed time slice t_{i+1}, the other receives the result of the current rollback round for time slice t_i. (In Sect. 7.2.3 we have seen that this can lead to considerable slowdown due to restart.)

Additional temporary space may be necessary. In Sect. 7.2.3, a scheme to save intermediate results of the minmax calculation has been proposed to increase the efficiency slightly. `m_Prep` is used for this purpose. Also, some finite difference solvers may need their own scratch space; Crank-Nicholson, for instance, requires extra storage for the decomposed coefficient matrix and the right-side vector of the linear system which it has to solve. `m_Temp1` and `m_Temp2` can be activated for that purpose.

In summary, the components of `tLatticeInstance` shown in Fig. 9.29 have the following meaning:

`m_Slot` The portfolio/signature argument pair supplied to `createInstance()` of class `tLattice` is converted in a compact array `m_Slot` of references to instruments.

The definition of `m_Slot` as array of references to instruments is incomplete, however. Instruments have different maturity dates and thus enter into the computation at different times during the rollback, therefore widening the lattice instance dynamically (of course, all memory is allocated before-hand, and the widening is

```
class tLatticeInstance {

    tClaim* m_Slot[...];

    int m_nCurrent;

    tMultiArray<double> m_Buffer[2];

    tMultiArray<double> m_Prep;
    tMultiArray<double> m_Temp1;
    tMultiArray<double> m_Temp2;

public:

    void beforeRollback( int nDay );
    void afterRollback( int nDay );

    void rotate();

    tMultiArray<double>& current() {
        return m_Buffer[m_nCurrent];
    }

    tMultiArray<double>& last() {
        return m_Buffer[m_nLast];
    }
};
```

Figure 9.29: A *very* condensed summary of the essential elements of `tLatticeInstance`. The template `tMultiArray` allows arrays whose dimensions are determined by objects of class `tArrayBounds`. `m_Prep`, `m_Temp1` and `m_Temp2` can, but must not be used during rollback

only logical). `m_Slot` has additional features to allow this process to occur efficiently. Their are also some additionl supporting members, for instance for index translation between `m_Slot` and `tPortfolio::m_Claim`. All this is not shown for simplicity.

`m_nCurrent` The index in `m_Buffer` of the multi-array representing the current hyperplane. "Current" refers to the time slice, say t_i, that is being computed in the current rollback round. The "last" hyperplane refers to the hyperplane of time slice t_{i+1}.

`m_Buffer` This buffer holds two hyperplanes of the total space of the lattice instance. `m_Buffer[m_nCurrent]` contains the current hyperplane; `m_Buffer[m_nCurrent−1]` contains the last hyperplane. We stress again that the innermost coordinate of each hyperplane loops through the gradient $\hat{v}_k$ plus the total worst-cast value $\hat{V}$.

`m_Prep` Temporary space used for intermediate results (see Sect. 7.2.3).

`m_Temp1` and `m_Temp2` Temporary space, mainly used by mixed explicit/implicit schemes such as Crank-Nicholson.

`beforeRollback()` and `afterRollback()` These functions are called before and after rollback rounds when the current time slice t_i falls on a day boundary. These functions take care of maturing instruments and adjust the (logical) width of the lattice instance.

`rotate()` Replaces `m_nCurrent` with `m_nCurrent` $-$ 1 and thus rotates the current and last hyperplanes.

`current()` Returns a reference to the current hyperplane.

`last()` Returns a reference to the last hyperplane.

`current()` and `last()` are not the only functions to access the elements of the lattice instance. Their are additional functions to read and write `m_Prep`, `m_Temp1` and `m_Temp2`. There are also functions to access not entire multi-arrays, but single innermost rows or entries. The list of actual member functions exceeds the list of functions shown in Fig. 9.29 several times.

The classes `tTimeAxis` and `tSpaceAxis` are less interesting. We only note that `tSpaceAxis` contains the virtual member `createOFSolver()` mentioned above, and that the derived class `tGeoSpaceAxis` implements this function as follows:

```
tOFSolver* tGeoSpaceAxis::createOFSolver()

{
    if( isImplicit() )
        return new tGeoImplicit( this );
    return new tGeoExplicit( this );
}
```

`tGeoSpaceAxis` also implements the algorithm in Fig. 5.4 to find a stable discretization.

Finite Difference Solvers

Figure 9.27 shows the hierarchy of finite difference solvers, but does not emphasize the multiple-inheritance relationship very strongly. This is done in Fig. 9.30, which shows the dependency graph. Doubly framed classes are abstract base classes which are ultimately used as interfaces to access the functionality of the concrete solver.

The purpose of each class is as follows:

`tOFSolver` This base class is general in the sense that its member functions rely on the assembly of the tridiagonal coefficient matrix and the right-side vector for one rollback round at some other place. (Both explicit and mixed explicit/implicit methods can be expressed in this manner.) Once the linear system of equations has been set up, its solution can be computed independently from the concrete financial model or spatial lattice geometry. The most visible feature of `tOFSolver` is the pure virtual member function `solve()`.

`tOFExplicit` Provides a body for the prototype `solve()` in `tOFSolver`. Uses an explicit forward Euler one-level scheme.

`tOFImplicit` Provides a body for the prototype `solve()` in `tOFSolver`. Uses a mixed explicit/implicit Crank-Nicholson scheme. In addition, allows incremental refinement, which is necessary for American options.

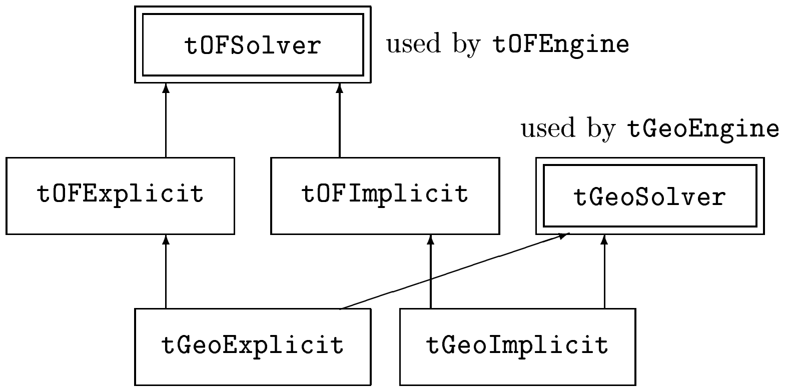


Figure 9.30: The class hierarchy for finite difference solvers. The abstract base class `tOFSolver` is used by member functions of `tOFEngine` to access the functionality of a particular solver. Similarly, the abstract base class `tGeoSolver` is used by `tGeoEngine`

`tGeoSolver` This abstract class contains a reference to the `tGeoSpaceAxis` object that has created the solver. It also has access to the model drift and volatility coefficients from which to build the tridiagonal transition matrix. `tGeoSolver` acts as pheripheral source of information.

`tGeoExplicit` This class is the bridge between `tGeoSolver` and `tOFExplicit`. If retrieves model coefficients form the former and instantiates the transition weights for the latter. The simplified nature of the linear system of equations in the explicit case is taken into account.

`tGeoImplicit` This class is the bridge between `tGeoSolver` and `tOFImplicit`. If retrieves model coefficients form the former and instantiates the transition matrix and right-side vector for the latter.

`tOFSolver` and Child Classes Some of the essential features of `tOFSolver` are shown in Fig. 9.31. A description follows:

`tProcessParamsStub` This empty type provides a handle to whatever process parameters need to be transferred in order to compute the transition matrix. This type is public.

```
class tOFSolver {

public:

    struct tProcessParamsStub {
    };

    class tIncrement {
        public:
            virtual void beginIncrement( ... );
            virtual void doIncrement( ... );
            virtual void endIncrement( ... );
    };

protected:

    virtual void calcWeights( int nFromLevel, int nToLevel,
        const tProcessParamsStub& Params );

public:

    virtual void solve();
    virtual tRetCode refine( tIncrement& Incr );
};
```

Figure 9.31: Some of the features of `tOFSolver`. The virtual functions *and* the local types must be expanded in child or otherwise related classes

In our case, it is expanded by `tGeoSolver` to provide the local drift and volatility coefficients.

`tIncrement` If American options are involved and a mixed explicit/implicit scheme is used, the solution of the linear system of equations needs to be refined. Iterative refinement proceeds in alternatingly reevaluating early-exercise decisions and performing a subsequent over-relaxation step. This loop may require additional memory space or knowledge, which is encapsulated in a class derived from `tIncrement`. The loop is then executed by calling the virtual member functions of `tIncrement`.

`calcWeights()` This function must compute the transition weights. The arguments `nFromLevel` and `nToLevel` indicate the location of the interval of nodes in the lattice for which the rollback is being performed (the interval may change due to knock-out or a tree-shaped lattice). The function also gets to see the process parameters in the argument `Params`, after a proper down-cast.

`solve()` The top-level function that initiates the current rollback round. This function contains the numeric part of the code.

`refine()` If `solve()` has successfully finished and the portfolio contains American options, `refine()` must be called repeatedly to adjust the result. This function makes use of the `tIncrement` interface.

`tOFExplicit` is a straightforward instantiation of `tOFSolver`. `calcWeights()` remains still unresolved, since `tOFExplicit` is a general explicit solver that does not know about the concrete model coefficients. `solve()` is implemented, but the function `refine()` is ignored.

The class `tOFImplicit` is more complex. It implements both `solve()` and `refine()`. `solve()` performs a LU decomposition based on the LAPACK modules DGTTRF and DGTTRFS, which were translated and adapted from Fortran. The code allows for partial pivoting. See LAPACK (1999) for more details.

The function `refine()` reuses the decomposition of `solve()` to modify the current result. It does so in an over-relaxation step where the relaxation parameter ω lies between 1 and 2 and is dynamically adapted, based on the previous iteration count.

`tGeoSolver` and Child Classes The auxiliary class `tGeoSolver` is defined in Fig. 9.32. It mainly expands the empty type `tProcessParamsStub` defined in class `tOFSolver`. The elements `m_gVol`, `m_gDrift` and `m_gDiscount` are the local values of the model coefficients in class `tBSModel`. The process parameters are retrieved from the model by the compute engine, which is an instance of type `tGeoEngine` (more on compute engines below).

```
class tGeoSolver {

public:

    struct tProcessParams :
            public tOFSolver::tProcessParamsStub {
        double m_gVol;
        double m_gDrift;
        double m_gDiscount;
    };
};
```

Figure 9.32: The class `tGeoSolver` expands the stub class `tProcessParamsStub`, defined in `tOFSolver`. Objects of class `tProcessParams` will be supplied by the compute engine

The classes `tGeoExplicit` and `tGeoImplicit` combine the model specific information captured in `tGeoSolver` and the numerical functionality of `tOFExplicit` and `tOFImplicit`. Solvers actually created by the program belong to either class, whose definition is shown in Fig. 9.33. The implementation of `calcWeights()` in either case is without surprises and therefore ommitted.

9.1.7 Evaluaters

Evaluaters collect portfolio, model, scenario and numerical method, make sure they all fit together and initiate the evaluation process. Evaluaters are short-lived, contrary to objects of other external classes which can be reused (note that the evaluater object in Fig. 9.2 does not need a name). They are used once and then thrown away. Figure 9.34 shows the definition.

The individual components of `tEvaluate` have the following semantics:

`m_pPortfolio` A reference to the portfolio to be evaluated.

```
class tGeoExplicit : public tOFExplicit, public tGeoSolver {

    void calcWeights( int nFromLevel, int nToLevel,
        const tProcessParamsStub& Params );
};

class tGeoImplicit : public tOFImplicit, public tGeoSolver {

    void calcWeights( int nFromLevel, int nToLevel,
        const tProcessParamsStub& Params );
};
```

Figure 9.33: Actual solvers belong either to class `tGeoExplicit` or `tGeoImplicit`. They compute the transition matrix from information provided in `Params`

`m_pModel` A reference to the mode under which the portfolio is to be evaluated.

`m_pScenario` A reference to the scenario for the evaluation. This parameter is ignored if Monte-Carlo is the numerical method of choice.

`m_pOptimizer` At the time of this writing, calibration through optimization is only possible for Monte-Carlo methods. The optimizer object adds an outer loop to the evaluation process, which the compute engine must know about. Ignored if lattice-based evaluation is selected.

`m_pLattice` If this pointer is set, the evaluation is lattice-based.

`m_pPathSpace` If this pointer is set, the evaluation is done with Monte-Carlo simulation. Only one of `m_pLattice` and `m_pPathSpace` can be set. The class `tPathSpace` is not explained here.

`m_pFDEngine` The compute engine created with the call

```
m_pModel->createEngine( m_pScenario, m_pFDEngine, m_nAccuracy );
```

if `m_pLattice` is set. See also Fig. 9.9.

`m_pMCEngine` The compute engine created with the call

```
class tEvaluate {

    tPortfolio* m_pPortfolio;
    tModel* m_pModel;
    tScenario* m_pScenario;
    tOptimizer* m_pOptimizer;
    tLattice* m_pLattice;
    tPathSpace* m_pPathSpace;

    tFDEngine* m_pFDEngine;
    tMCEngine* m_pMCEngine;

    tCurveContainer m_CurveContainer;
    tImageContainer m_ImageContainer;

    tAccuracy m_nAccuracy;

public:

    tRetCode run();
};
```

Figure 9.34: Evaluaters know about all the objects that make up a particular pricing problem, cross-reference them and oversee the evaluation process

```
    m_pModel->createEngine( m_pMCEngine );
```

if `m_pPathSpace` is set.

`m_CurveContainer` Besides pricing and optimization (under Monte-Carlo), MtgLib also offers curve-generating functionality from calibrated path spaces. Curves can be written to files, to be used in subsequent pricing rounds.

`m_ImageContainer` Calibrated curves can also be converted into images. Currently supported is the GIF format popular on the World Wide Web. Curves and images appear in Sect. 10.2. No implementation details are provided.

`m_nAccuracy` This parameter applies to lattice-based evaluation only and controls which of the speed-up techniques of Sect. 7.2 should be used. Possible values are `Low` (collapsing the corridors of uncertainty for American options) and `Exact` (maintaining the corridors of uncertainty). See also Sect. 9.1.3.

`run()` This functions initiates the evaluation process. It transfers control to the `run()` member function of either `*m_pFDEngine` or `*m_pMCEngine`.

9.2 The Class Hierarchy—Internal

The vast majority of the classes in MtgLib are internal—not visible through the scripting language, of which an example is given in Fig. 9.2. The solver classes in Fig. 9.27, although listed together with the hierarchy of lattice classes, may be considered internal, for instance. In this section, we restrict ourselves to the discussion of the most important category of internal classes: compute engines.

9.2.1 Compute Engines

Figure 9.35 shows the current hierarchy of compute engines. We discuss only the branch that forks of `tFDEngine`, since all algorithmic aspects dealt with in the earlier parts of this thesis occur in classes derived from `tFDEngine`. `tMCEngine` and its subclasses are based on Monte-Carlo simulation.

`tEngine`, `tFDEngine` and `tOFEngine` are all abstract; they do not function by themselves. Instances are created from `tGeoEngine` and `tShockEngine`, depending on the scenario as shown in Fig. 9.12. The classes are used as follows:

`tEngine` This class has several purposes:

- It contains references to objects used by all types of compute engines, like the model or portfolio.
- It performs some initialization that can be done on that high a level; for instance, it matches factors referenced in the portfolio with factors defined in the model.
- It defines an interface to retrieve singleton portfolios. This is important for corridors of uncertainty.

Class name	Purpose
`tEngine`	Base class (abstract)
`tFDEngine`	Extended base class for lattice-based evaluation (abstract)
`tOFEngine`	Extended base class for one-factor lattice-based evaluation (abstract)
`tGeoEngine`	Worst-case evaluation for geometric Brownian motion models
`tShockEngine`	Evaluation under volatility shock scenarios
`tMCEngine`	Extended base class for Monte-Carlo simulation (abstract)
`tHJMEngine`	For fixed income
`tShortRateEngine`	For fixed income

Figure 9.35: The hierarchy of compute engines. `tShockEngine` extends `tGeoEngine` by overriding some functions that handle administrative tasks between rollback rounds

Many functions that need to access the current state (the payoff functions in Fig. 9.4, for instance) do so either through the `tEngine` or through the `tFDEngine` interface, after a proper down-cast.

`tFDEngine` This class extends `tEngine` in several regards:

- It contains more references to objects defined for the current problem, for instance the lattice template and the scenario.
- It implements the main `run()` function of any compute engine derived from it. `tFDEngine` contains local data structures for the dynamic lookup mechanism of lattice instances through (regularized) signatures. In some sense, `tFDEngine` implements the "outer loop" over the time axis of the finite difference scheme.
- It contains more information about the current state than `tEngine`. It knows which time slice t_i is currently being processed (since it drives the loop!), and provides variables, to be set by subclasses, that locate individual nodes

in the corresponding hyperplane. It provides functions to access this state information.

`tFDEngine` is equipped to handle multi-factor models. This class implements the core of the multi-lattice dynamic programming paradigm introduced in Sect. 5.1.

`tOFEngine` While `tFDEngine` works for multi-factor models, `tOFEngine` does not. `tOFEngine` adds inner-loop functionality to `tFDEngine`. (For the inner loop for fixed t_i, the number of factors must be known.)

`tGeoEngine` This class instantiates `tOFEngine` to support one-factor geometric Brownian motion models under worst-case volatility scenarios.

`tShockEngine` This class extends `tGeoEngine` to support volatility shock scenarios.

The following paragraphs go into some implementation details.

The Abstract Class `tEngine`

A shortened definition of `tEngine` is shown in Fig. 9.36. The meaning of the individual member components is as follows:

`m_pPortfolio` A reference to the portfolio object under investigation.

`m_pModel` A reference to the model used for the evaluation.

`m_FactorXlat` The class `tPortfolio` has a member function `matchFactors()` that unifies the factor tables in the portfolio and model objects (see Fig. 9.7). The resulting index permutation is stored in `m_FactorXlat`.

`beforeRun()` This function takes care of initialization issues that can be handled with limited information. Factor matching is an example. If this function is overridden in child class `tXYZEngine`, the original function must always be called first:

```
tRetCode tXYZEngine::beforeRun() {
    if( tEngine::beforeRun() != OK ) ...
    ...
    return OK;
}
```

```
class tEngine {

    tPortfolio* m_pPortfolio;
    tModel* m_pModel;

    int m_FactorXlat[...];

protected:

    virtual tRetCode beforeRun();
    virtual tRetCode afterRun();

public:

    virtual void getClaim( int nIndex, int nTag,
        double& gUnitValue );
};
```

Figure 9.36: The abstract base class `tEngine` performs some preparation and cleanup tasks before and after evaluation. It also defines the interface for the retrieval of singleton portfolios, which is important to find the corridors of uncertainty for American options

As the class hierarchy builds up, each class contributes to initialization by overriding `beforeRun()`, but still calling the function in the parent class.

`afterRun()` Performs cleanup jobs, mostly related to memory management, after evaluation has been completed. (Curve and image generation are also possible aspects.) Again, overriding functions must make sure to call the original eventually:

```
tRetCode tXYZEngine::afterRun() {
    ...
    return tEngine::afterRun();
}
```

`getClaim()` This function is a pure virtual interface to retrieve the current value of the singleton portfolio with signature $(X_{\texttt{nIndex}}, \alpha)$, where $\alpha = \pm 1$, depending on `nTag` and scenario settings such as sell-side or buy-side point-of-view. The function is called, for instance, in `tWorstCase::refineExPolicy()`, shown in Fig. 9.25

The Abstract Class `tFDEngine`

`tFDEngine` implements the loop over the time axis and handles the repository of lattice instances. It works independently of the number of factors. Figure 9.37 shows the relevant fragment of its definition. Individual members are used as follows:

`m_nDay` and `m_gFractionOfDay` The value of t_i in days, where t_i is the current time slice. More precisely,

$$t_i = \texttt{m\_nDay} + \texttt{m\_gFractionOfDay}$$

The distinction between days and fractions of days is convenient, because the granularity for events connected with instruments or scenarios is at the level of days.

`m_pLattice` A reference to the lattice template.

`m_pScenario` A reference to the scenario object.

`m_Pos` This variable is not maintained by `tFDEngine`, only provided in order to be accessible through it. Any subclass that loops over the nodes of the current time slice t_i must update `m_Pos` during the preparatory phase of each loop iteration. The preparatory phase ends once the finite difference solver takes over.

To keep `m_Pos` consistent is important, since functions like `tClaim::payoff()` must know exactly which node is currently being processed.

`doRound()` Executes exactly one rollback round, for all currently known lattice instances. This function is final; it calls the function `doTask()`, which must be instantiated in any subclass, to process each lattice instance. `doRound()` observes the rule proposed in 5.1.1 for external consistency, augmented by provisions that guarantee that lattice instances are processed in the correct order under volatility shock scenarios (in fact, any scenario that uses a consistent pattern for regularizing signatures).

`doTask()` The pure virtual prototype that is called by `doRound()`.

`getLatticeInstance()` Accesses the lattice instance whose regularized signature is determined from the pair `Sig/nTag`. If the lattice instance does not exist, `getLatticeInstance()` creates it and interrupts the current iteration of `doRound()` through the C++ exception mechanism.

```
class tFDEngine : public tEngine {

    int m_nDay;
    double m_gFractionOfDay;

    tLattice* m_pLattice;
    tScenario* m_pScenario;

    int m_Pos[...];

    tRetCode doRound();

protected:

    virtual void doTask( tLatticeInstance& Instance );

    void getLatticeInstance( const tSignature& Sig, int nTag,
        tLatticeInstance*& pInstance );

    tRetCode beforeRun();
    tRetCode afterRun();

public:

    tRetCode run();

    void getClaim( int nIndex, int nTag, double& gUnitValue );
};
```

Figure 9.37: A small part of the definition of `tFDEngine`. Shown are the state information, the interface to access lattice instances, and the main `run()` function

`beforeRun()` After successful completion of the parent function, `beforeRun()` initializes the repository of lattice instances and creates the top-level instance.

`afterRun()` Merely calls the parent functions, and does not do any additional processing.

`run()` This function contains the central control loop over the time domain, as shown in Fig. 9.38.

`getClaim()` Instantiates the virtual function `getClaim()` whose prototype is defined in Fig. 9.36.

```
tRetCode tFDEngine::run()

{
    if( ( nRet = beforeRun() ) != OK )
        return nRet;

    int nNumOfRounds = m_pLattice->numOfSlices();

    for( int k = 0; k < nNumOfRounds; ++k ) {
        if( ( nRet = doRound() ) != OK ) {
            cleanup();
            return nRet;
        }
    }

    return afterRun();
}
```

Figure 9.38: A schematized listing of the central control loop in the function `run()` of class `tFDEngine`. `numOfSlices()` is not included in Fig. 9.28; it returns the number of discretization points in the time domain. `doRound()` belongs to `tFDEngine` and executes one rollback round

The Abstract Class `tOFEngine`

`tOFEngine` is the last abstract class in the chain of ever more specialized classes for compute engines. Subclasses of `tOFEngine` can be used to created concrete engines.

```
class tOFEngine : public tFDEngine {

    class tIncrement : public tOFSolver::tIncrement {
        tOFEngine& m_Engine;

        tIncrement( tOFEngine& Engine ) : m_Engine( Engine ) {}

        void beginIncrement( int nAdjDown, int nAdjUp )  {
            m_Engine.beginIncrement( nAdjDown, nAdjUp ); }
        void doIncrement( const int Pos[...] ) {
            m_Engine.doIncrement( Pos );       }
        void endIncrement( int nAdjDown, int nAdjUp )  {
            m_Engine.endIncrement( nAdjDown, nAdjUp ); }
    };

    tOFSolver* m_pSolver;

    void doBarriers( int& nAdjDown, int& nAdjUp );
    void doBoundary( int nAdjDown, int nAdjUp );
    void doRollback( int nAdjDown, int nAdjUp );
    void doMonitor( int nAdjDown, int nAdjUp );
    void doPayoff();

    void beginIncrement( int nAdjDown, int nAdjUp );
    void doIncrement( const int Pos[...] );
    void endIncrement( int nAdjDown, int nAdjUp );

protected:

    virtual const tOFSolver::tProcessParamsStub& getProcessParams();
    virtual tRetCode createSolver( tOFSolver*& pSolver );

    void doTask( tLatticeInstance& Instance );

    tRetCode beforeRun();
    tRetCode afterRun();
};
```

Figure 9.39: The class `tOFEngine`: a compute engine for one-factor models

`tOFEngine` provides the inner-loop functionality for one-factor models. The member function `doTask()` performs one rollback-round for the lattice instance passed as argument. What is missing to make `tOFEngine` a full-fledged compute engine is the calculation of the local model coefficients for the solver.

Figure 9.39 shows the definition of `tOFEngine`. The members of `tOFEngine` have the following semantics:

`tIncrement` The empty interface `tIncrement` has been defined in Fig. 9.31 for class `tOFSolver` to support incremental refinement for mixed explicit/implicit schemes. Here, the interface is instantiated as a proxy that forwards all requests to the parent compute engine. `nAdjUp` and `nAdjDown` are the adjusted number of nodes above and below the centered root node of the lattice. Adjustments occur when instruments knock out and therefore set up a new boundary.

`m_pSolver` The finite difference solver created with a call to `createSolver()`, see below.

`doBarriers()` The rollback round is executed in stages. Each stage is dedicated to a sub-task. `doBarriers()` locates all the barriers and initializes temporary data structures that guide the subsequent sub-tasks. It also returns the location of the adjusted boundary in `nAdjUp` and `nAdjDown`. These values are used in all subsequent sub-tasks.

`doBoundary()` Performs the second sub-task. If barriers have been found, lattice instances for subordinate partial portfolios must be accessed to set the boundary data. This is done by calling `getLatticeInstance()`.

`doRollback()` Once the boundary has been taken care of, the "numerical" part of the rollback (i.e., what is commonly associated with the term) is done for the continuation region. After some preparation, this function essentially calls `m_pSolver->-solve()`.

`doMonitor()` If American options are present, early exercise policies are gathered for each node by calling `tClaim::monitor()` for all relevant instruments and refining estimates with `m_pScenario->refineExPolicy()`. Then, early exercise combinations are evaluated where alternatives exist. This is the minmax calculation, with exploitation of intermediate results as outlined in Sect. 7.2.3.

`beginIncrement()`, `doIncrement()`, `endIncrement()` These functions repeat the monitoring of American options, given the current result. They contribute to the incremental refinement in the over-relaxation method used under mixed explict/implicit schemes. Refinement is started with `m_pSolver->refine()` before `doPayoff()` is called.

`doPayoff()` In the final sub-task, payoffs of maturing instruments and fixed cashflows are added. `doPayoff()` is only called *after* incremental refinement through the `tIncrement` proxy has been completed.

`getProcessParams()` This pure virtual function must be instantiated by a subclass. It supplies the missing information on which evaluation relies. To defer the instantiation of `getProcessParams()` makes `tOFEngine` general.

`createSolver()` This function creates the finite difference solver: it calls `m_pLattice->-createOFSolver()`. (The lattice template, in turn, relays the request to the space axis, as described in Sect. 9.1.6.)

`doTask()` The main function of the class. It calls `doBarriers()`, `doBoundary()`, `do-Rollback()`, `doMonitor()`, refines, and calls `doPayoff()`, in that order.

`beforeRun()` After calling the parent version, this function creates the solver by calling `createSolver()`.

`afterRun()` Deletes the solver and jumps to the parent version.

`tFDEngine` and `tOFEngine` together are the logistic heart of lattice-based evaluation (the solver and lattice class hierarchies are the numerical one), comprising combined about 2800 lines of code.

The Class `tGeoEngine`

Not much remains to do to complete `tOFEngine` to a working compute engine for one-factor geometric Brownian motion models. Figure 9.40 shows the rather short definition.

The task of `tGeoEngine` is to ensure that the solver receives the correct model coefficients for the current time slice. The function `getProcessParams()` reads the drift and volatility bounds from the model (to which a reference is provided in `tEngine`). It

```
class tGeoEngine : public tOFEngine {

    tGeoSolver::tProcessParams m_Params;
    const tOFSolver::tProcessParamsStub& getProcessParams();
};
```

Figure 9.40: The class `tGeoEngine` prepares the model coefficients for `tOFEngine` and `tGeoSolver`

computes the local gamma and uses the scenario object (to which a reference is kept in `tFDEngine`) to select the scenario volatility, by calling the member function `selectVol()`. Process parameters are then stored in `m_Params` and returned.

The Class `tShockEngine`

```
class tShockEngine : public tGeoEngine {

    int m_nDepth;

    int m_nCurTag;
    tLatticeInstance* m_pCurCoInstance;

    tRetCode beforeRun();
};
```

Figure 9.41: The class `tShockEngine` and some of its members

The class `tShockEngine` extends `tGeoEngine` for volatility shock scenarios, which require periodic data transfers between lattice instances. Figure 9.41 shows some of the members of `tShockEngine`. They are interpreted as follows:

`m_nDepth` The number of conventional lattice instances per consolidating lattice instance. If d is the duration of the volatility shock scenario, and p its periodicity, then `m_nDepth` $= \lceil d/p \rceil$. See Sect. 8.1.1 for motivation.

`m_nCurTag` The regularized tag of the lattice instance being processed. `m_nCurTag` and the scenario parameters d, p and f imply the extended signature variables τ, ξ and δ as defined in Def. 8.3.

`m_pCurCoInstance` If the extended-signature parameter τ of the current lattice instance is "conventional", then `m_pCurCoInstance` references the consolidating lattice instance of the same level. If the current lattice instance is consolidating, then `m_pCurCoInstance` points to the conventional lattice instance from which data might have to be imported. (Data is only compared and possibly imported on certain dates, and only with with respect to one conventional lattice instance at a time.)

`beforeRun()` Creates all the extra lattice instances needed for the volatility shock scenario. This function implements the algorithm in Fig. 8.4.

9.2.2 Other Groups of Classes

MtgLib contains about 135 classes, of which only those that form the combinatorial and mathematical kernel have been discussed in the previous sections. Other classes fill in the infrastructure to create actual applications.

These categories of classes are also part of MtgLib:

- Figure 9.2 shows an example of the scripting language in which MtgSvr communicates. In general, each object class knows how to parse itself. Each object class contains a static member function `parse()` that creates a new object from a script definition. There is a central class `tParser`, and peripheral classes `tScanner`, `tSource`, `tFileSource`, `tStringSource` and `tNetSource` for support. For customized claims, classes `tExpression`, `tNumericalExpr` and `tExPolicyExpr` provide the necessary extension to the scripting language. The parser is of the recursive-descent variety.

- MtgSvr resides as a service on Windows NT PC's. The classes `tSocket`, `tService`, `tJobServer` support this background operation.

- MtgCal, to be discussed briefly in Sect. 10.2, uses the CGI protocol, aided by classes `tCgi` and `tTclCgi`.

- Some low-level classes provid special data structures: `tHeap` and `tHeap2` for one- and two-dimensional dynamic arrays; `tMultiArray`; `tMap` for one-dimensional, highly homogeneous arrays; and `tSignature`, which is implemented as a bitfield.

Future work will deal with additional support for actually traded instruments, calibration and hedging capabilities based on Monte Carlo simulation, and improved remote accessibility.

10 Towards Web-based Applications

We consider it a worthwhile undertaking to use contemporary technology to disseminate results in a way that proves their applicability empirically and at the same time creates potentially useful tools for the community. We think the World Wide Web and its standard technologies like CGI, Java or Javascript enable us to do just that. In our experimental web site, pricers and calibration tools make it possible for everyone to apply the results of our research, in particular with regard to uncertain volatility scenarios.

The following sections briefly discuss MtgClt/MtgSvr and MtgCal, two online applications. The client-server application MtgClt/MtgSvr discussed in Sect. 10.1 prices vanilla, barrier and American options portfolios under worst-case and volatility shock scenarios. MtgCal, discussed in Sect. 10.2 is an online calibration-tool for fixed income and the focus of our current research efforts.

10.1 Example 1: a Client/Server UVM Pricer

The UVM (Uncertain Volatility Model for historical reasons) pricer consists of two components. MtgClt is a Java applet that is anchored in an HTML page in our website. MtgSvr is a C++ program that physically resides on the machine that serves the applet, but is logically separated from the web server (we use the Apache server for Windows NT).

The pair MtgClt/MtgSvr implements all algorithms discussed in this thesis, and is therefore empirical proof for their practical applicability.

MtgClt contains a GUI (graphical user interface) that lets the user enter data in three categories:

- In the *portfolio* category, up to eight vanilla, barrier, American and customized options can be entered. Preconfigured option types include options with linear and digital payoff.
- In the *scenario* category, model coefficients such as volatility, interest rate and dividend rate (respectively foreign interest rate) are specified. All coefficients can have term structure format (`tStepDrift` and `tStepVol` are used to represent the coefficients). In addition, the volatility may exhibit uncertainty.

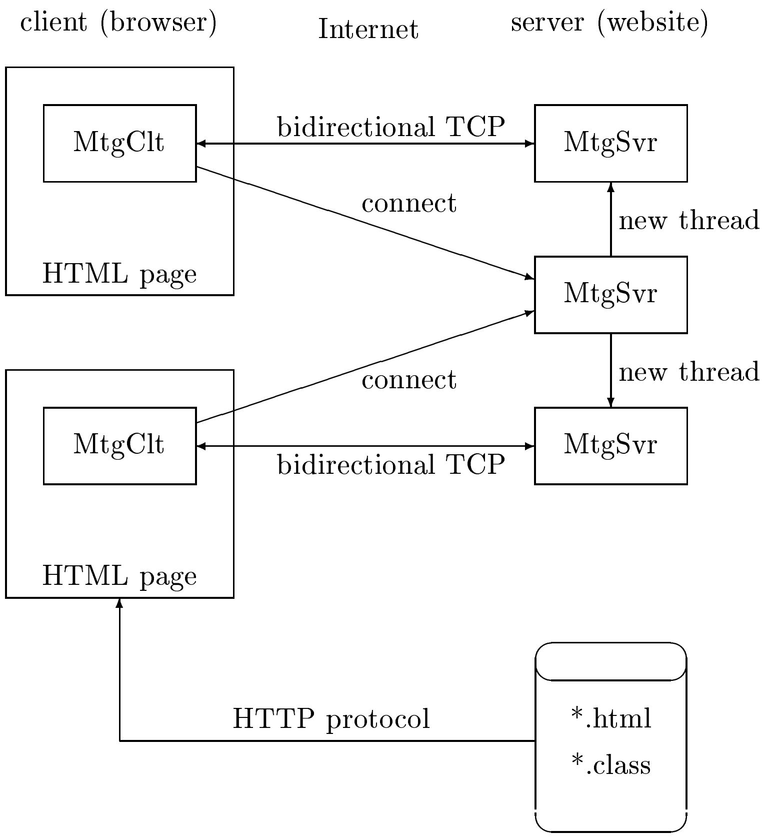


Figure 10.1: The architecture of MtgClt/MtgSvr. MtgSvr accepts incoming connections and processes requests in separate threads (Windows NT) or forked-off processes (Unix). The *.html and *.class files are served by the Apache web server

The user also selects between the worst-case volatility scenario and the volatility shock scenario. In the latter case duration, periodicity and frequency are entered. Another data field determines whether the global point of view is sell-side or buy-side oriented. The distinction has been briefly made in Sect. 9.1.5.

- In the *advanced settings* category, finite difference scheme (explicit or Crank-Nicholson), trimming parameters (see Sect. 9.1.6), speedup techniques for American options (maintaining/collapsing of corridors of uncertainty) and time steps are selected.

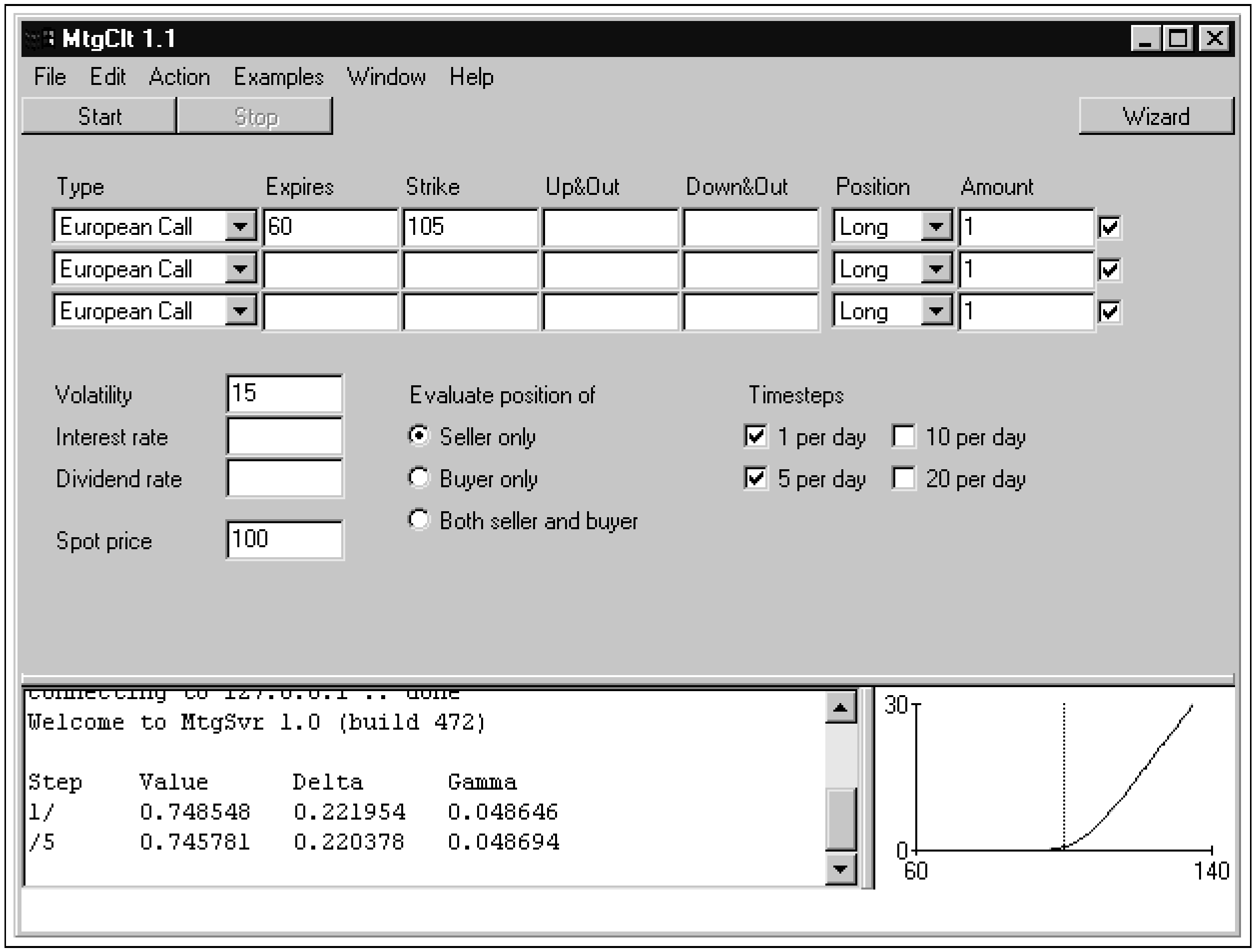
  In order to give a better idea of the convergence behavior of the program for the particular pricing problem, more than one time step can be entered. The result is then computed and listed for all time steps.

Graphically, MtgClt distinguishes between “One-Click” mode in which all categories (slightly down-sized) are combined in a single entry form, and “Wizard” mode in which each category is assigned its own form.

Once all entries have been made, the user presses the “Start” button and MtgClt connects to the MtgSvr via TCP. MtgSvr handles the incoming connection by creating a new thread (under Windows NT) or by forking off a copy of itself (under Unix). Thus, several incoming requests can be handled at the same time.

MtgClt converts the data in the entry fields into a script in the proprietary scripting language of MtgSvr (an example is shown in Fig. 9.2) and sends the script. The new thread created by MtgSvr parses the script, creates the objects it defines and executes the `evaluate` statements (of which there must be at least one). The result is sent back to MtgClt as soon as it becomes available. MtgClt/MtgSvr therefore use a simply request/response scheme to communicate.

This architecture is shown in Fig. 10.1. Figures 10.2, 10.3 and 10.4 contain some screen snapshots of MtgClt in action. Only the MtgClt entry forms are shown; the browser that runs MtgClt is hidden.



Type	Expires	Strike	Up&Out	Down&Out	Position	Amount	
European Call	60	105			Long	1	✓
European Call					Long	1	✓
European Call					Long	1	✓

Volatility 15
Interest rate
Dividend rate
Spot price 100

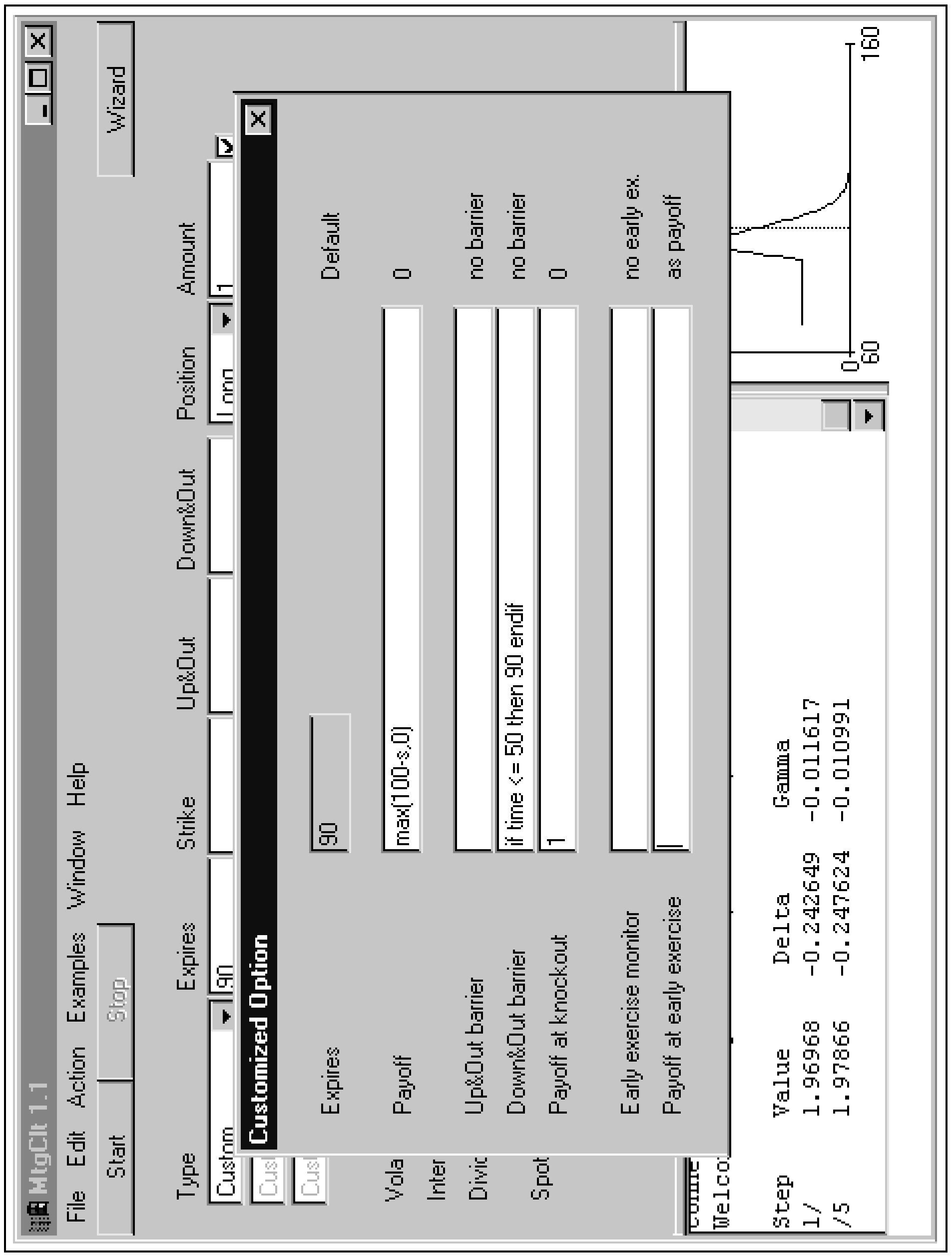
Evaluate position of
[✓ Seller only]
Buyer only
Both seller and buyer

Timesteps
[✓ 1 per day] 10 per day
[✓ 5 per day] 20 per day

```
connecting to 127.0.0.1 .. done
Welcome to MtgSvr 1.0 (build 472)

Step    Value       Delta       Gamma
1/      0.748548    0.221954    0.048646
/5      0.745781    0.220378    0.048694
```

Figure 10.2: A European call option, evaluated with $dt = 1/365$ and $dt = 1/(5 \times 365)$

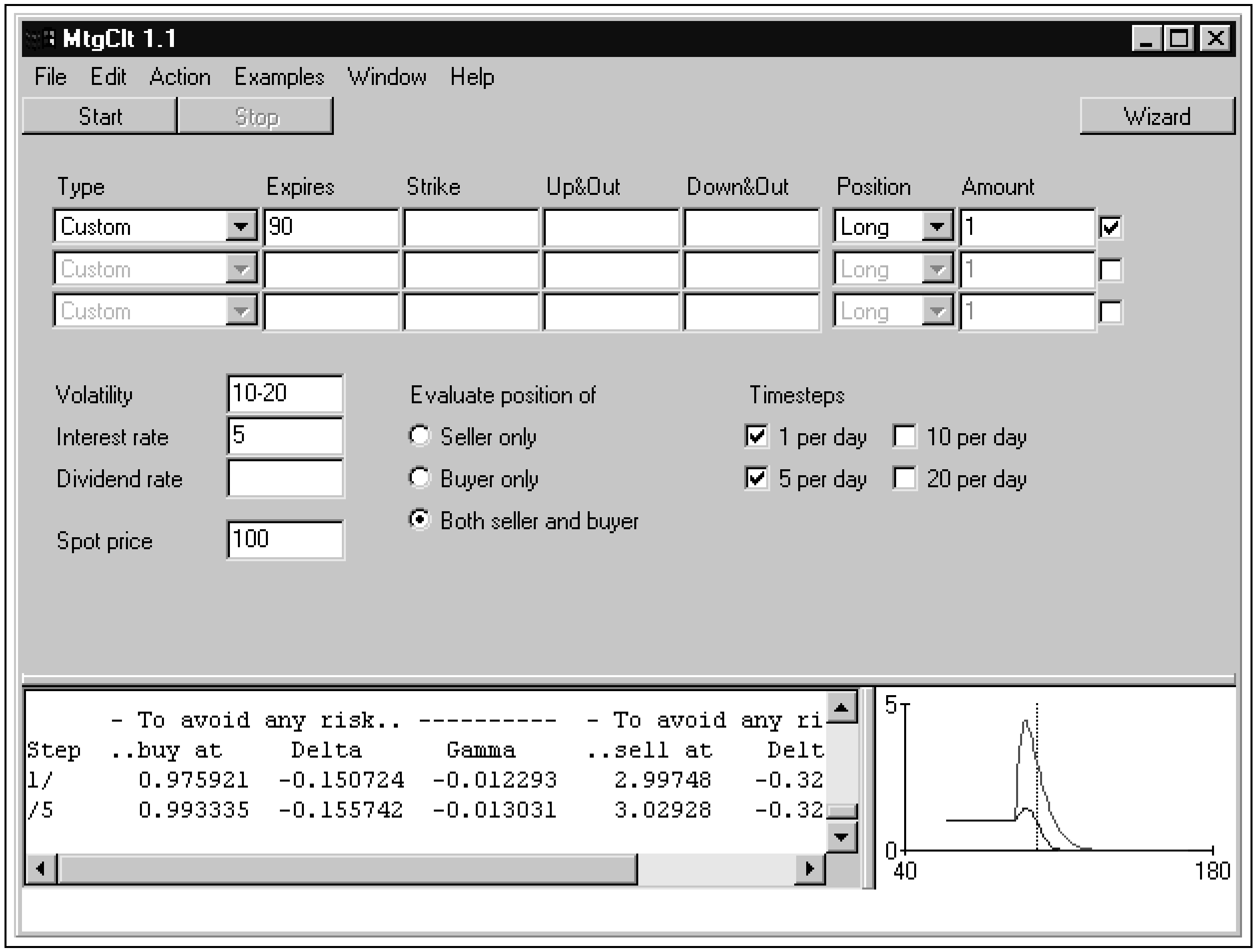


Type	Expires	Strike	Up&Out	Down&Out	Position	Amount
Custom	90				Long	1

		Default
Expires	90	
Payoff	max(100-s,0)	0
Up&Out barrier		no barrier
Down&Out barrier	if time <= 50 then 90 endif	no barrier
Payoff at knockout	1	0
Early exercise monitor		no early ex.
Payoff at early exercise		as payoff

```
Step  Value    Delta      Gamma
1/    1.96968  -0.242649  -0.011617
/5    1.97866  -0.247624  -0.010991
```

Figure 10.3: A customized 90-day down-and-out put. The barrier exists only for 50 days. The knock-out premium is $1



Type	Expires	Strike	Up&Out	Down&Out	Position	Amount	
Custom	90				Long	1	✓
Custom					Long	1	☐
Custom					Long	1	☐

Volatility	10-20
Interest rate	5
Dividend rate	
Spot price	100

Evaluate position of
- ○ Seller only
- ○ Buyer only
- ● Both seller and buyer

Timesteps
- ✓ 1 per day
- ✓ 5 per day
- ☐ 10 per day
- ☐ 20 per day

```
     - To avoid any risk.. ----------  - To avoid any ri
Step ..buy at      Delta      Gamma     ..sell at    Delt
1/     0.975921  -0.150724  -0.012293    2.99748    -0.32
/5     0.993335  -0.155742  -0.013031    3.02928    -0.32
```

Figure 10.4: The put of Fig. 10.3 evaluated under the worst case scenario in which $0.1 \leq \sigma \leq 0.2$

10.2 Example 2: Remote Calibration Sketched

Calibration has been very shortly introduced in Sect. 4.2.3, in connection with uncertain volatility models. In this section, we present an online, i.e. "remote", calibrator that allows users to choose their own model preferences.

The calibrator differs from the UVM pricer presented in the last section in two important aspects:

- It uses a Monte-Carlo simulation method;
- it calibrates fixed income models to fixed income instruments (currently to US treasury bonds).

Thus, it seems, all algorithms presented in this thesis become useless. This is indeed so for the multi-lattice dynamic programming algorithms and the algorithms for barrier and American options. The object-oriented software framework discussed in Chapter 9, however, allows to embed the new components in a preserving manner. Classes for fixed income instruments are derived from `tClaim` just the same (`tUSTBond` for the US treasuries); so are classes for models (`tHJMGaussianModel`, `tShortRateModel` and `tVasicekModel`, based on `tModel`) and compute engines (`tHJMEngine` and `tShortRateEngine`, based on `tMCEngine`). Throughout chapter 9 we have included hints in places where such extensions have been made.

The following exposition should therefore be viewed as addendum to the core topics of this thesis. It stresses the importance of sound software design that leads to extensible software. It emphasizes our vision of financial computing on the Internet (or Intranets) and throws some light on ongoing research and possible future directions.

10.2.1 Theoretical Foundations

The following paragraphs are necessarily incomplete. By no means do we give an exhaustive account of the theory. The interested reader is referred to Cover and Thomas (1991) and Avellaneda (1998).

Let $\bar{\mathbf{X}}$ and $\bar{\pi}$ be a vector of $\bar{k}$ contingent claims and a corresponding price vector, respectively. The position $\hat{\lambda} \in \mathbb{R}^{\bar{k}}$ that minimizes (4.4.19) in Fact 4.9 implies a calibrated volatility $\hat{\sigma}$ which prices $\bar{\mathbf{X}}$ correctly.

$\hat{\lambda}$ and $\hat{\sigma}$ may be found with lattice-based dynamic programming, as stated in Fact 4.10. Here we follow a different approach.

For simplicity, assume the Vasicek model

$$dr = (\theta - \alpha r)dt + \sigma dX \tag{10.10.1}$$

for the short rate process $r = \{r_t\}$ (any other model will do, too). dX is the random shock, α the speed of mean reversion, and $\frac{\theta}{\alpha}$ the level of mean reversion.

In the Monte-Carlo setting, the process r is simulated for N paths $\omega_1, \ldots, \omega_N$. The value of any instrument X can then be evaluated by approximating its discounted expected payoff:

$$F_0(X) \doteq \frac{1}{N} \sum_{i=1}^{N} \exp\left(-\int_0^T r_t(\omega_i)dt\right) X(\omega_i) \tag{10.10.2}$$

The summation in (10.10.2) amounts to assigning to each path the weight $\frac{1}{N}$. This uniform probability distribution P of paths is consistent with the *prior model* (10.10.1).

Now assume a different probability distribution Q for the paths $\omega_1, \ldots, \omega_N$, i.e. $0 < q_1, \ldots, q_N < 1$ and $\sum_{i=1}^N q_i = 1$. The Kullback-Leibler distance of the new distribution Q to the original, uniform distribution P is

$$\begin{aligned} H(Q|P) &= \sum_{i=1}^{N} Q(\omega_i) \log\left(\frac{Q(\omega_i)}{P(\omega_i)}\right) \\ &= \sum_{i=1}^{N} Q(\omega_i) \log\left(\frac{Q(\omega_i)}{1/N}\right) \\ &= \log N + \sum_{i=1}^{N} Q(\omega_i) \log Q(\omega_i) = \log N + \sum_{i=1}^{N} q_i \log q_i \end{aligned} \tag{10.10.3}$$

$0 \leq H(Q|P) \leq \log N$, and $H(Q|P) = 0$ if $Q = P$. A measure $Q \neq P$ implicitely changes the price of the instrument X:

$$F_0(X \mid Q) \doteq \sum_{i=1}^{N} q_i \exp\left(-\int_0^T r_t(\omega_i)dt\right) X(\omega_i) \tag{10.10.4}$$

If N is much greater than $\bar{k}$, it makes sense to ask for Q which correctly prices $\bar{\mathbf{X}}$, given $\bar{\pi}$. It is furthermore reasonable to assume that the model builder prefers Q that minimizes $H(Q|P)$. Given (10.10.3), this is equivalent to maximizing the entropy $H(Q) =$

$-\sum_{i=1}^{N} q_i \log q_i$. Avellaneda (1998) shows that under certain assumptions, this constrained entropy optimization problem has a unique solution, which can be found by the method of Lagrange multipliers:

$$\inf_{\bar{\lambda} \in \mathbb{R}^{\bar{k}}} \left\{ \sup_{Q} \left[H(Q) + F_0(\bar{\lambda} \cdot \bar{\mathbf{X}} \mid Q) \right] - \bar{\lambda} \cdot \bar{\pi} \right\} \tag{10.10.5}$$

This formula corresponds to (4.4.19) in Fact 4.9.

The supremal Q can be found directly for fixed λ. The optimal $\hat{\lambda}$ is found with a gradient-based optimization algorithm (L-BFGS-B in our case).

10.2.2 Extensions to MtgLib

Compute engines for Monte Carlo have already been listed as members of MtgLib in Fig. 9.35. Classes for interest rate models are listed in Fig. 9.8. Figure 10.5 shows some additional classes that contribute to the outer optimization loop (recall that `tEvaluate` in Fig. 9.34 contains a member variable `m_pOptimizer`).

Class name	Purpose
`tOptimizer`	optimizer template (abstract)
`tEntropyOpt`	minimum entropy optimizer template
`tOptInstance`	optimizer instance (abstract)
`tMCOptInstance`	optimizer instance for Monte Carlo (abstract)
`tMCEntropyOptInstance`	optimizer instance for minimum entropy optimization (also inherits from `tMinimizer`)
`tMinimizer`	wrapper for L-BFGS-B

Figure 10.5: Extensions to MtgLib. Not shown in this picture are the compute engines derived from `tMCEngine`; these are mentioned in Sect. 9.2.1

The purpose of each class in Fig. 10.5 is as follows:

`tOptimizer` The abstract base class for optimizer templates. Just as for lattices, we distinguish between optimizer templates that contain information on the type of the optimizer, and actual optimizer instances which are created by optimizer templates as requested by compute engines and used only once. Optimizer instances contain the "dirty" variables used during the actual computation.

The main feature of `tOptimizer` is the member function `createInstance()`, which is pure virtual.

`tEntropyOpt` Supports minimum entropy optimizer templates. `createInstance()` returns objects of type `tMCEntropyOptInstance`. In addition, minimum entropy optimizer templates specify the upper and lower bound for Lagrange multipliers. The following fragment would be a valid definition of a minimum entropy optimizer in the scripting language used by MtgCal:

```
optimizer xyz {
    type entropy,
    low -100, high 100
}
```

(MtgCal uses the same scripting language as MtgSvr.)

`tOptInstance` The abstract base class for optimizer instances. This class is designed with both lattice-based and simulation methods in mind, although optimization right now is supported only for simulation methods.

`tOptInstance` contains basic member variables such as `m_Price` (the price vector $\bar{\pi}$), `m_Lambda` (the output vector holding the optimal $\hat{\lambda}$'s) and `m_Gradient` (used by the gradient-based minimization routine).

`tMCOptInstance` A specialization of `tOptInstance` for Monte Carlo simulation. The member variable `m_Weight` is the vector that holds the alternative distribution Q for the Monte Carlo paths. This class also precomputes the discounted cashflows for each instrument and path, since this information needs to be computed only once. It then calls a pure virtual member function `minimize()` to do the actual optimization.

`tMCEntropyOptInstance` Inherits from both `tMCOptInstance` and `tMinimizer`; implements `minimize()` defined in `tMCOptInstance` by passing control to `tMinimizer::-minimize()`, which in turn calls back a member function `eval()` in each iteration.

`eval()` is the partition function for the minimum entropy problem (see Avellaneda (1998)).

`tMinimizer` A wrapper to the L-BFGS-B code that has been translated from Fortran to C (see Zhu *et al.* (1994)).

There are other extensions to MtgLib that deal with the generation of curves and their output as GIF images. These extensions are ommitted to keep this chapter short.

10.2.3 Architecture

The requirements on the client environment posed by MtgCal are less stringent than those necessary for MtgClt/MtgSvr. In particular, the client only needs to support Javascript instead of Java. Communication between the client and the server uses the CGI protocol. This avoids low-level TCP and thus solves the firewall problem.

Calibration does not always fit into the request-response pattern of the HTTP protocol, because it may require a long time to complete. Calibration is therefore split into a sequence of steps:

1. The user enters data into a form and submits it by clicking the "Calibrate" button.

2. The web server receives the request, reads the data and passes it to MtgCal, which is installed as CGI handler.

3. MtgCal spawns off a separate copy of itself and passes the data to it. The new process immediately starts calibration. The original instance of MtgCal creates a temporary HTML page asking the user for patience ("in progress..."), which in turn is returned to the client by the web server.

   The user therefore experiences immediate feedback, regardless of the prospective duration of calibration.

4. The temporary HTML page contains some Javascript code that periodically submits a query to MtgCal. MtgCal detects that the query comes from the temporary page and checks on the status of the current calibration, rather than initiating a new one.

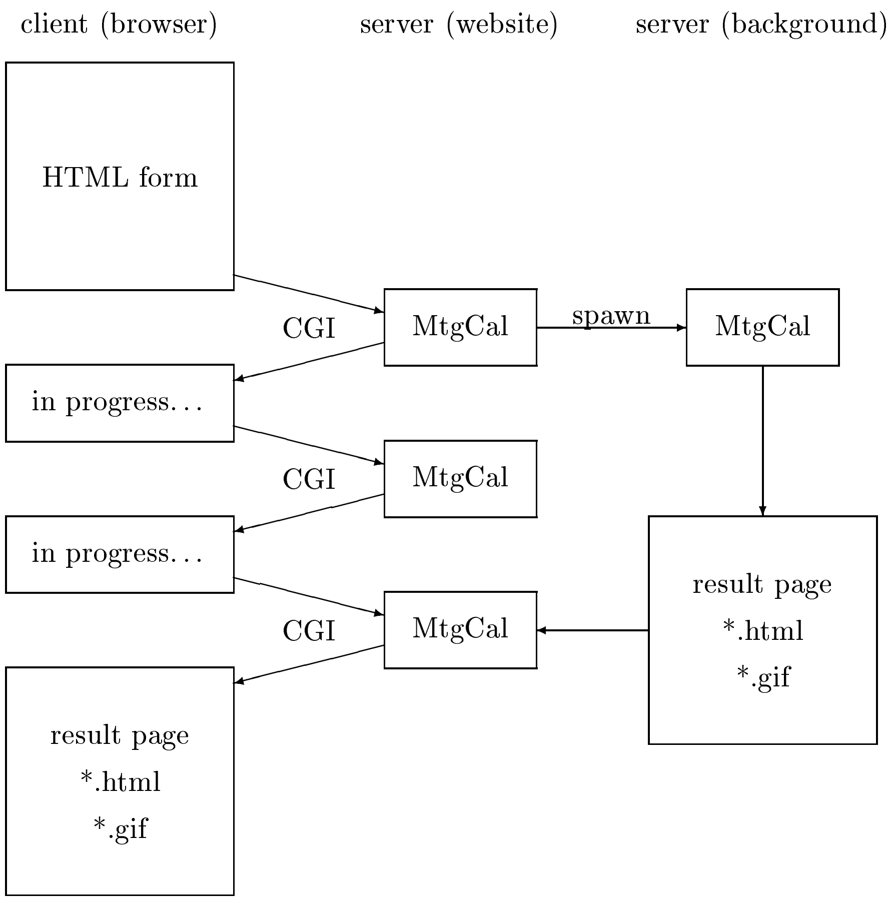


Figure 10.6: The architecture of MtgCal. HTML, Javascript and the CGI protocol are used to transfer data between the client and the server. Proxies and firewalls pose no problem for this setup. Calibration is done in the background on the server

5. As soon as the background instance of MtgCal finishes calibration, it creates a result HTML pages with links to GIF images. The result page is detected at the next status check, returned to the client, and the user finally sees the result.

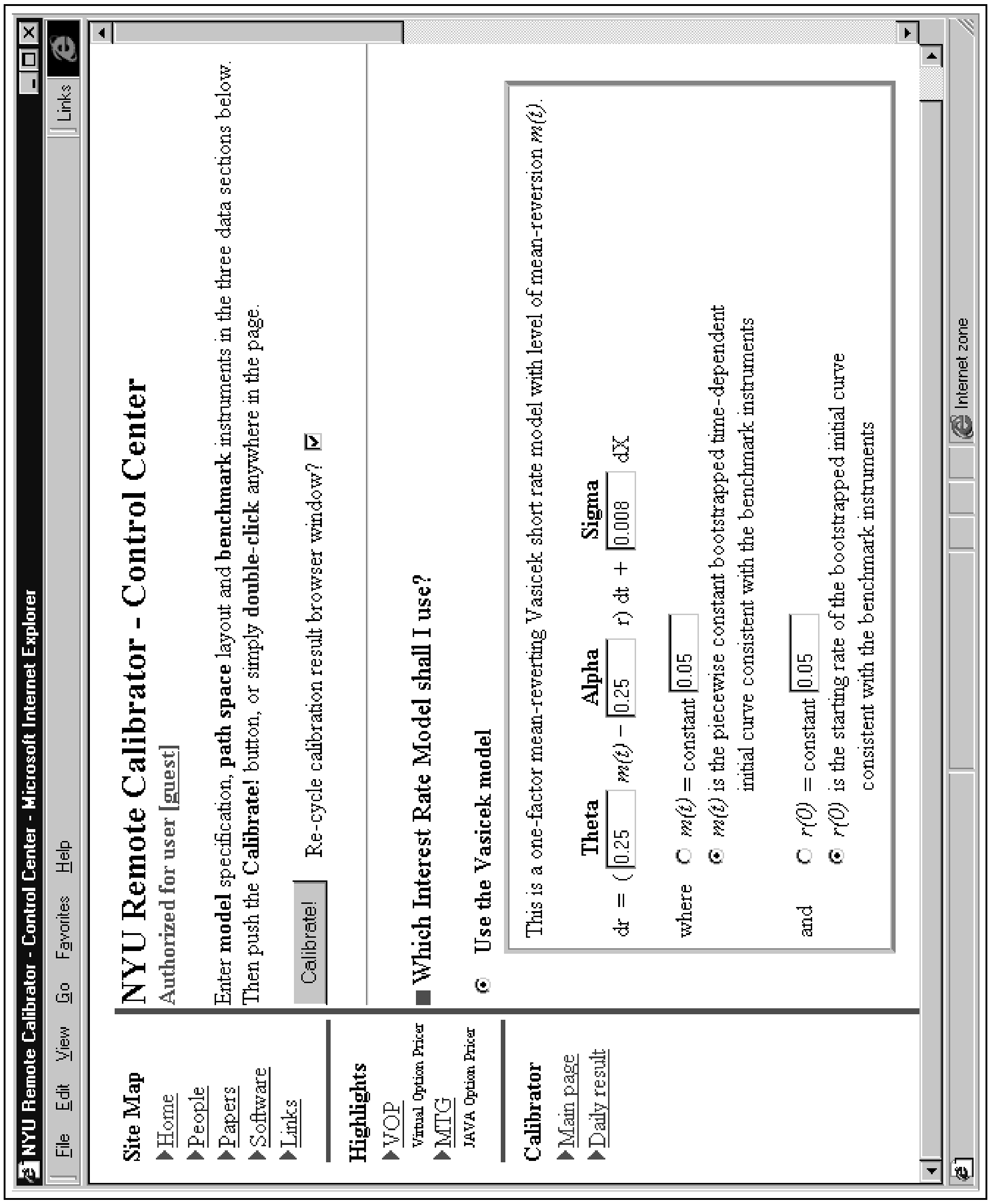
   In addition to images, pure data such as the calibrated forward rate curve is also written to disk on the server, to support subsequent rate calculations and, in the future, pricing.

The client engages in a polling action to eventually find the result. Other approaches are possible, but this one seems the most robust and is straightforward to implement. Figure 10.6 shows the architecture pictorially.

It is possible to submit another calibration request even while a background copy of MtgCal is active. Each calibration request supercedes the previous one and terminates background copies of MtgCal prematurely.

Figures 10.7, 10.8, 10.9 and 10.10 show the layout of the HTML form into which calibration requests are entered, and of the result page, respectively.

The result, once computed, is persistent. Figure 10.11 shows how the calibrated forward rate curve can be used to compute interest rates under different quoting conventions.



NYU Remote Calibrator - Control Center

Authorized for user [guest]

Enter **model** specification, **path space** layout and **benchmark** instruments in the three data sections below. Then push the **Calibrate!** button, or simply **double-click** anywhere in the page.

Re-cycle calibration result browser window? [✓]

Which Interest Rate Model shall I use?

[✓] **Use the Vasicek model**

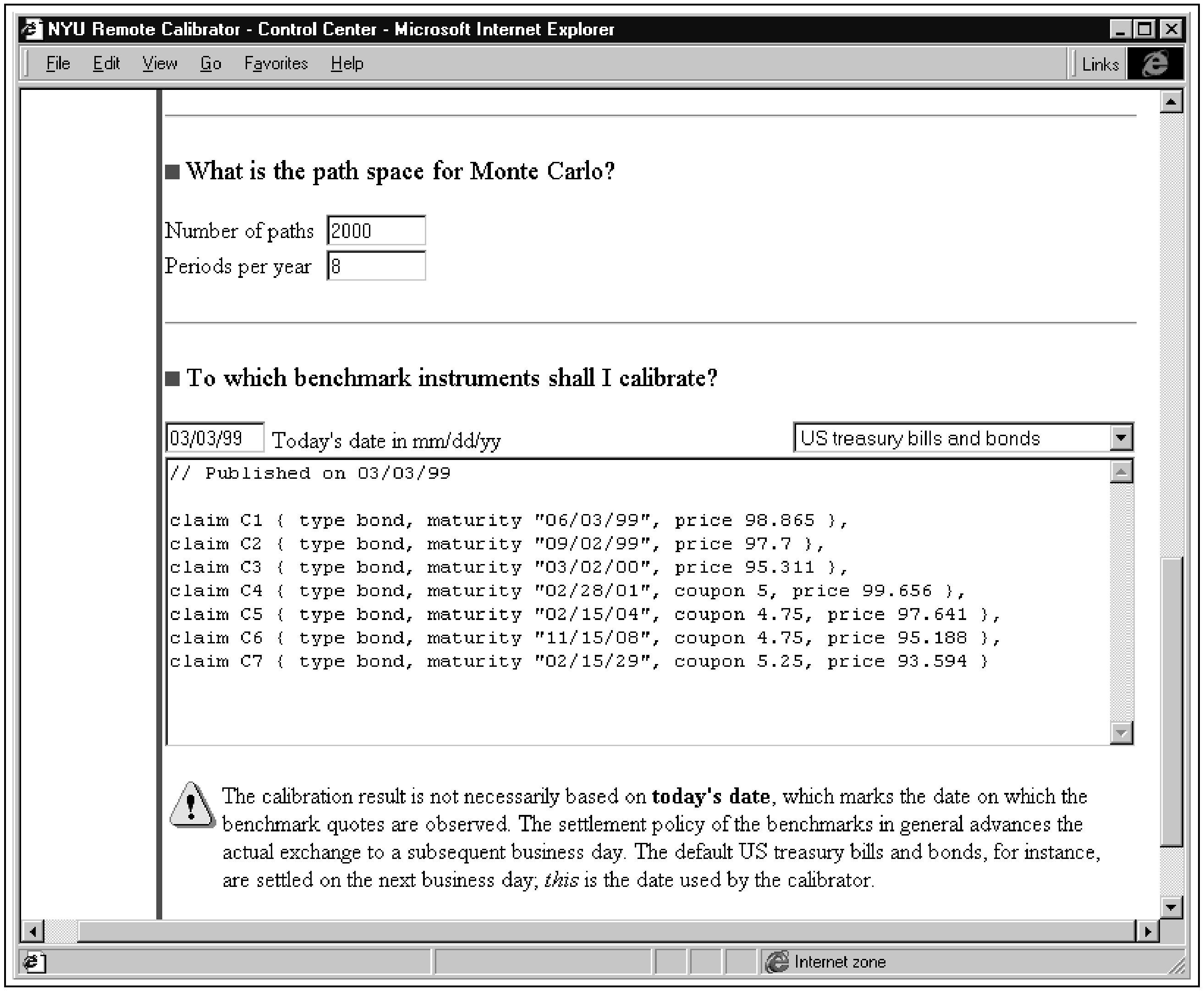
This is a one-factor mean-reverting Vasicek short rate model with level of mean-reversion $m(t)$.

dr = (**Theta** 0.25 $m(t)$ − **Alpha** 0.25 r) dt + **Sigma** 0.008 dX

where
- ○ $m(t)$ = constant 0.05
- [✓] $m(t)$ is the piecewise constant bootstrapped time-dependent initial curve consistent with the benchmark instruments

and
- ○ $r(0)$ = constant 0.05
- [✓] $r(0)$ is the starting rate of the bootstrapped initial curve consistent with the benchmark instruments

Figure 10.7: The top half of the HTML form for MtgCal


NYU Remote Calibrator - Control Center - Microsoft Internet Explorer
File Edit View Go Favorites Help
Links
Internet zone

■ What is the path space for Monte Carlo?

Number of paths: 2000
Periods per year: 8

■ To which benchmark instruments shall I calibrate?

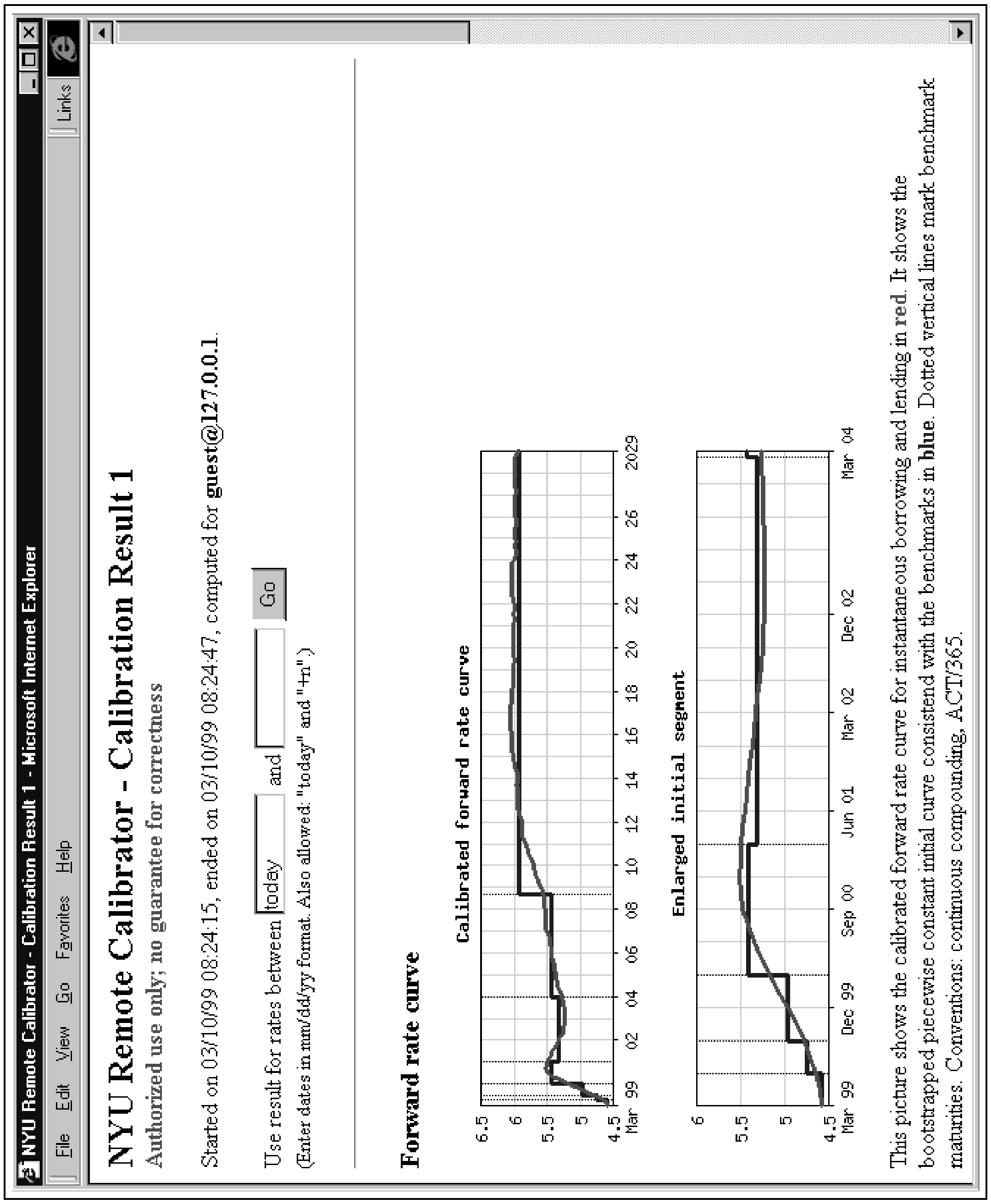
03/03/99 Today's date in mm/dd/yy — US treasury bills and bonds

```
// Published on 03/03/99

claim C1 { type bond, maturity "06/03/99", price 98.865 },
claim C2 { type bond, maturity "09/02/99", price 97.7 },
claim C3 { type bond, maturity "03/02/00", price 95.311 },
claim C4 { type bond, maturity "02/28/01", coupon 5, price 99.656 },
claim C5 { type bond, maturity "02/15/04", coupon 4.75, price 97.641 },
claim C6 { type bond, maturity "11/15/08", coupon 4.75, price 95.188 },
claim C7 { type bond, maturity "02/15/29", coupon 5.25, price 93.594 }
```

The calibration result is not necessarily based on **today's date**, which marks the date on which the benchmark quotes are observed. The settlement policy of the benchmarks in general advances the actual exchange to a subsequent business day. The default US treasury bills and bonds, for instance, are settled on the next business day; *this* is the date used by the calibrator.

Figure 10.8: The bottom half of the HTML form for MtgCal



NYU Remote Calibrator - Calibration Result 1

Authorized use only; no guarantee for correctness

Started on 03/10/99 08:24:15, ended on 03/10/99 08:24:47, computed for **guest@127.0.0.1**.

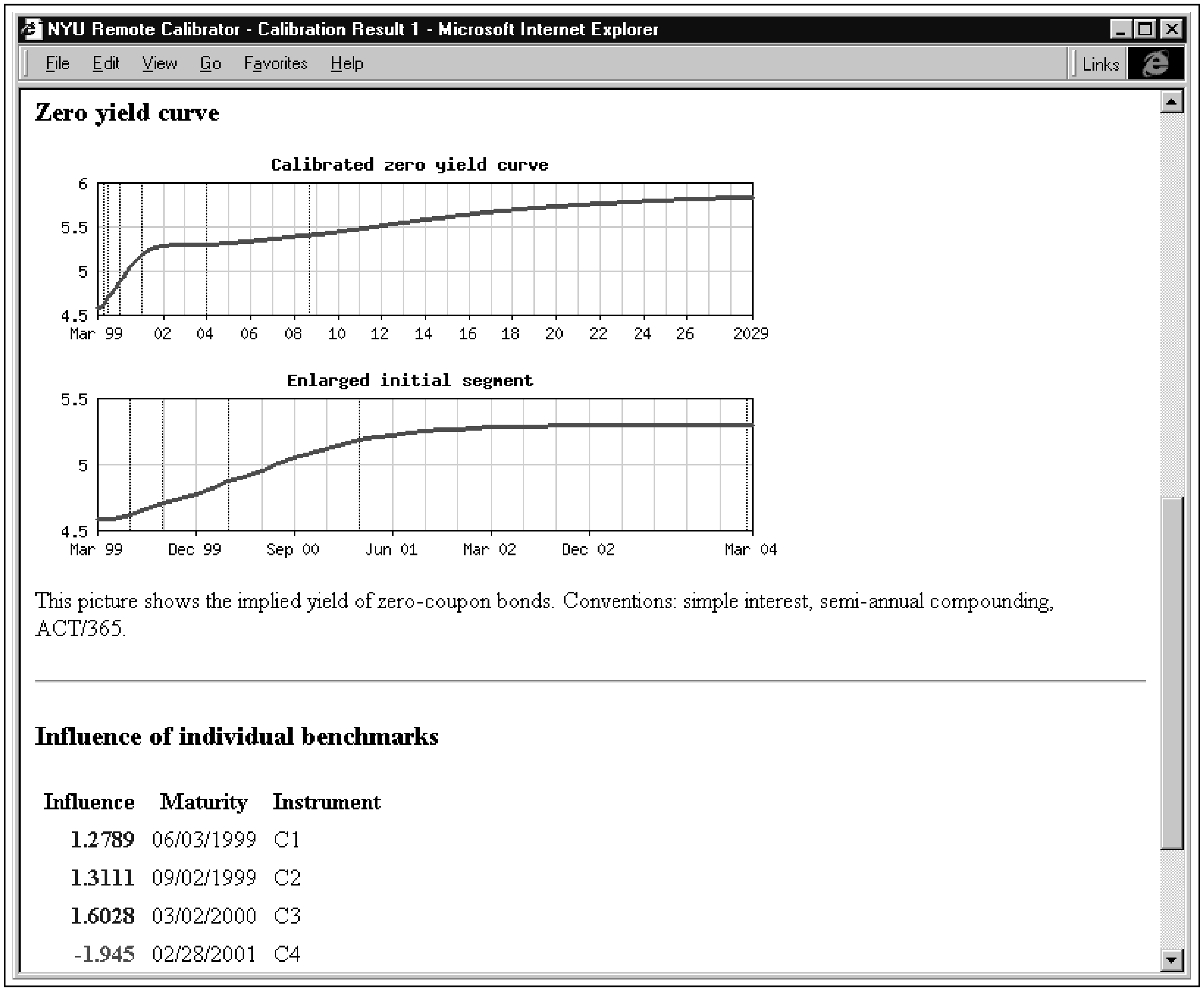
Use result for rates between today and

(Enter dates in mm/dd/yy format. Also allowed: "today" and "+n")

Forward rate curve

This picture shows the calibrated forward rate curve for instantaneous borrowing and lending in **red**. It shows the bootstrapped piecewise constant initial curve consistend with the benchmarks in **blue**. Dotted vertical lines mark benchmark maturities. Conventions: continuous compounding, ACT/365.

Figure 10.9: The top half of the HTML result page after calibration


NYU Remote Calibrator - Calibration Result 1 - Microsoft Internet Explorer
File Edit View Go Favorites Help
Links
Calibrated zero yield curve
6
5.5
5
4.5
Mar 99
02
04
06
08
10
12
14
16
18
20
22
24
26
2029
Enlarged initial segment
5.5
5
4.5
Mar 99
Dec 99
Sep 00
Jun 01
Mar 02
Dec 02
Mar 04

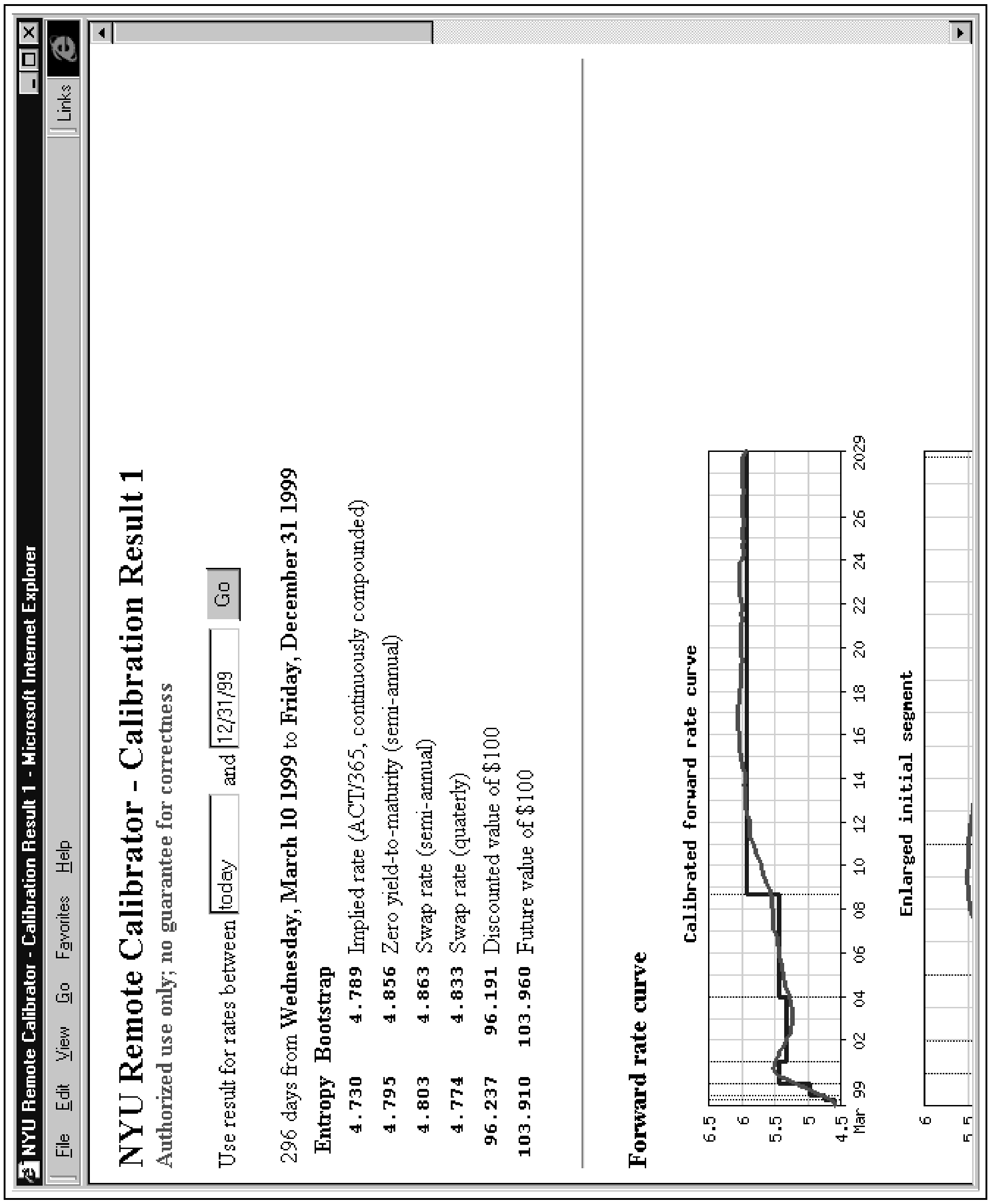
Zero yield curve

This picture shows the implied yield of zero-coupon bonds. Conventions: simple interest, semi-annual compounding, ACT/365.

Influence of individual benchmarks

Influence	Maturity	Instrument
1.2789	06/03/1999	C1
1.3111	09/02/1999	C2
1.6028	03/02/2000	C3
-1.945	02/28/2001	C4

Figure 10.10: The bottom half of the result page. Influence of benchmarks is measured by Langrange multipliers. Not shown are $\lambda_5 = 0.2105$, $\lambda_6 = 0.0478$, and $\lambda_7 = -0.0778$



NYU Remote Calibrator - Calibration Result 1

Authorized use only; no guarantee for correctness

Use result for rates between today and 12/31/99

296 days from **Wednesday, March 10 1999** to **Friday, December 31 1999**

Entropy	**Bootstrap**	
4.730	**4.789**	Implied rate (ACT/365, continuously compounded)
4.795	**4.856**	Zero yield-to-maturity (semi-annual)
4.803	**4.863**	Swap rate (semi-annual)
4.774	**4.833**	Swap rate (quaterly)
96.237	**96.191**	Discounted value of $100
103.910	**103.960**	Future value of $100

Figure 10.11: The calibration result can be used to calulate rates

11 Conclusion

We have laid the theoretical foundation of and implemented algorithms that price portfolios of vanilla, barrier and American options under uncertain volatility scenarios such as the worst-case volatility and volatility shock scenarios. Our implementation follows object-oriented principles and is modular and extensible.

In particular, our algorithmic contributions are

- a method to precompute the number of subordinate pricing problems that arise when the portfolio under consideration contains barrier options;
- a method to arrange statically as well as dynamically the hierarchy of pricing problems that arise under nonlinear scenarios in general;
- techniques to handle portfolios that contain American options in particular;
- a heuristic that allows to cut down the number of pricing problems for portfolios with American options;
- an extension of worst-case volatility scenarios to volatility shock scenarios in which the volatility may experience one or several periods of high-amplitude fluctuations at unpredictable times.

Each of these theoretical achievements has a concrete representation in the class hierarchy of the C++ class library MtgLib.

We have also demonstrated that complicated algorithms can be brought to a wider audience by using contemporary Internet technologies. MtgClt/MtgSvr and MtgCal are two online applications available in our website. We project that web computing for finance will quickly gain importance, in our own research and at other places.

Bibliography

Arnold, L. (1973): *Stochastic Differential Equations: Theory and Applications*, Wiley

Avellaneda, M., and R. Buff (1998): "Combinatorial Implications of Nonlinear Models: the Case of Barrier Options," to appear in *Appl. Math. Finance*

Avellaneda, M., and A. Parás (1995): "Pricing, and Hedging Derivative Securities in Markets with Uncertain Volatilities," *Appl. Math. Finance*, 2, 73–88

Avellaneda, M., and A. Parás (1996): "Managing the Volatility Risk of Portfolios of Derivative Securities: the Lagrangian Uncertain Volatility Model," *Appl. Math. Finance*, 3, 21–52

Avellaneda, M., C. Friedman, R. Holmes, and D. Samperi (1997): "Calibrating Volatility Surfaces via Relative-entropy Minimization," to appear in *Appl. Math. Finance*

Avellaneda, M. (1998): "Minimum-entropy Calibration of Asset-pricing Models," to appear in *International Journal of Theoretical and Applied Finance*

Barraquand, J., and T. Pudet (1996): "Pricing of American Path-dependent Contingent Claims," *Math. Finance*, 6, No. 1, 16–51

Baxter, M., and A. Rennie (1996): *Financial Calculus*, Cambridge University Press

Bensoussan, A. (1984): "On the Theory of Option Pricing," *Acta Applicandae Mathematicae*, 2, 139–158

Black, F., and P. Karasinski (1991): "Bond and Options Pricing when Short Rates are Lognormal," *Financial Analysts Journal*, July-August 1991, 52–59

Borodin, A. N., and P. Salminen (1996): *Handbook of Brownian Motion—Facts and Formulae*, Basel: Birkhäuser

Breeden, D. T., and R. H. Litzenberger (1978): "Prices of State-contingent Claims Implicit in Option Prices," *Journal of Business*, 51, No. 4, 621–651

Buff, R. (1999a): "Worst-case Scenarios for American Options," to appear in *International Journal of Theoretical and Applied Finance*

Buff, R. (1999b): “Java Pricer for Barrier and American Options,” http://www.courantfinance.cims.nyu.edu/mtg

Buff, R. (1999c): “Courant Remote Calibrator,” http://www.courantfinance..cims.nyu.edu/remotecalibrator/welcome.html

Buff, R. (1999d): “Daily Calibration Result,” http://www.courantfinance..cims.nyu.edu/remotecalibrator/auto/result.html

Cheuk, T. H. F., and T. C. F. Vorst (1996): “Complex Barrier Options,” *The Journal of Derivatives*, 4, No. 1, 8–22

Conze, A., and Viswanathan (1991): “Path Dependent Options: The Case of Lookback Options,” *J. Finance*, 46, No. 5, 1893–1907

Cormen, T. H., C. E. Leiserson, and R. L. Rivest (1990): *Introduction to Algorithms*, McGraw-Hill

Courtadon, G. (1982): “A More Accurate Finite Difference Approximation for the Valuation of Options,” *J. Financial Quant. Anal.*, 27, 697–703

Cover, T. M., and Thomas, J. A. (1991): *Elements of Information Theory*, John Wiley & Sons, New York

Cox, J. C., J. E. Ingersoll, Jr., and S. A. Ross (1985a): “A Theory of the Term Structure of Interest Rates,” *Econometrica*, 53, No. 2, 385–407

Cox, J. C., J. E. Ingersoll, Jr., and S. A. Ross (1985b): “An Intertemporal General Equilibrium Model of Asset Prices,” *Econometrica*, 53, 363–384

Cox, J. C., and M. Rubinstein (1985): *Options Markets*, Englewood Cliffs, NJ: Prentice-Hall

Duffie, D. (1996): *Dynamic Asset Pricing Theory*, 2nd ed., Princeton, NJ: Princeton University Press

Duffie, D., and R. Kan (1996): “A Yield-factor Model of Interest Rates,” *Math. Finance*, 6, No. 4, 379–406

Föllmer, H., and M. Schweizer (1991): "Hedging of Contigent Claims under Incomplete Information," *Applied Stochastic Analysis*, eds. M. H. A. Davis and R. J. Elliott. New York: Gordon and Breach, 389–414

Geman, H., and M. Yor (1996): "Pricing, and Hedging Double-barrier Options: a Probabilistic Approach," *Math. Finance*, 6, No. 4, 365–378

Geske, R., and K. Shastri (1985): "Valuation by Approximation: A Comparison of Alternative Option Valuation Techniques," *J. Financial Quant. Anal.*, 20, 45–71

Goldman, M. B., H. B. Sosin, and M. A. Gatto (1979): "Path Dependent Options: Buy at the High, Sell at the Low," *J. Finance*, 34, No. 5, 1111-1127

Heath, D., R. Jarrow, and A. Morton (1992): "Bond Pricing, and the Term Structure of Interest Rates: a New Methodology for Contigent Claims Valuation," *Econometrica*, 60, No. 1, 77-105

Ho, T. S., and S. Lee (1986): "Term Structure Movements and Pricing Interest Rate Contingent Claims," *J. Finance*, 41, 1011–1028

Hofmann, N., E. Platen, and M. Schweizer (1992): "Option Pricing under Incompleteness and Stochastic Volatility," *Math. Finance*, 2, No. 3, 153–187

Harrison, J. M., and D. M. Kreps (1979): "Martingales and Arbitrage in Multiperiod Securities Markets," *J. Econ. Theory*, 20, 381–408

Harrison, J. M., and S. R. Pliska (1981): "Martingales and Stochastic Integrals in the Theory of Continuous Trading," *Stoch. Process. Appl.*, 11, 215–260

Hull, J. C. (1993): *Options, Futures and Other Derivative Securities*, 2nd ed., Prentice Hall

Hull, J. C., and A. White (1987): "The Pricing of Options on Assets with Stochastic Volatilities," *J. Finance*, 42, 281–300

Hull, J. C., and A. White (1990): "Pricing Interest Rate Derivative Securities," *Review of Financial Studies*, 3, 4, 573–592

Hull, J. C., and A. White (1990): "Valuing Derivative Securities Using the Explicit Finite Difference Method," *J. Financial Quant. Anal.*, 25, 87–100

Hull, J. C., and A. White (1994): *Numerical Procedures for Implementing Term Structure Models*, Working paper, University of Toronto

Jackwerth, J. C., and M. Rubinstein (1996): "Recovering Probability Distributions from Option Prices," *J. Finance*, 51, No. 5, 1611–1631

Jarrow, R. A. (1996): *Modelling Fixed Income Securities and Interest Rate Options*, McGraw-Hill

Jeanblanc-Picque, M., N. El Karoui, and R. Viswanathan (1991): "Bounds for the Price of Options," *Applied Stochastic Analysis*, eds. I. Karatzas and D. Ocone, New York: Springer

Johnson, H., and D. Shanno (1987): "Option Pricing When the Variance is Changing," *J. Financial Quant. Anal.*, 22, 143–151

Karatzas, I. (1988): "On the Pricing of American Options," *Applied Mathematics and Optimization*, 17, 37–60

Karatzas, I. (1989): "Optimization Problems in the Theory of Continous Trading," SIAM J. Control Optim., 27, 1221–1259

Kloeden, P. E., and E. Platen (1991): *The Numerical Solution of Stochastic Differential Equations*, New York: Springer

Kunitomo, N., and M. Ikeda (1992): "Pricing Options with Curved Boundaries", *Math. Finance*, 2, No. 4, 275–272

Lagnado, R., and S. Osher (1997): "A Technique for Calibrating Derivative Security Pricing Models: Numerical Solution of an Inverse Problem," *J. Comp. Finance*, 1, No. 1, 13–25

Lamberton, D., and B. Lapeyre (1996): *Introduction to Stochastic Calculus Applied to Finance*, London: Chapman & Hall

"LAPACK," http://gams.nist.com

Longstaff, F. A., and E. S. Schwartz (1998): "Valuing American Options By Simulation: A Simple Least Squares Approach," Anderson Graduate School of Management working paper, University of California, Los Angeles, 1998

Lyden, S. (1996): "Reference Check: A Bibliography of Exotic Options Models," *The Journal of Derivatives*, 4, No. 1, 79–91

Parás, A. (1995): *Non-linear Partial Differential Equations in Finance: a Study of Volatility Risk and Transaction Costs*, Ph.D. thesis, New York University

Pirkner, C. D., Weigend, A. S, and Zimmermann, H. (1999): "Extracting Risk-Neutral Densities from Option Prices Using Mixture Binomial Trees," *Proceedings of the 1999 IEEE/IAFE/INFORMS Conference on Computational Intelligence for Financial Engineering (CIFEr'99)*, 135–158

Press, H. W., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling (1988): *Numerical Recipes in C*, Cambridge University Press

Roberts, C. O., and C. F. Shortland (1997): "Pricing Barrier Options with Time-dependent Coefficients," *Math. Finance*, 7, No. 1, 93–93

Rubinstein, M. (1995): "Implied Binomial Trees," *J. Finance*, 49, 771–818

Rubinstein, M., and E. Reiner (1991): "Breaking Down the Barriers," *RISK*, 4, No. 8

Schweizer, M. (1991): "Option Hedging for Semimartingales," *Stoch. Proc. Appl.* 37, 339–363

Scott, L. O. (1987): "Option Pricing when the Variance Changes Randomly: Theory, Estimation and an Application," *J. Financial Quant. Anal*, 22, 419-438

Roberts, G. O., and C. F. Shortland (1997): "Pricing Barrier Options with Time-dependent Coefficients," *Math. Finance*, 7, No. 1, 83–93

Thomas, J. W. (1995): *Numerical Partial Differential Equations: Finite Difference Methods*, Springer Verlag, New York

Turnbull, S. M., and L. M. Wakeman (1991): "A Quick Algorithm for Pricing European Average Options," *J. Financial Quant. Anal.*, 26, No. 3, 377–389

Wiggins, J. B. (1987): “Option Values under Stochastic Volatility: Theory and Empirical Estimates,” *J. Financial Econ.*, 19, 351–372

Vasicek, O. (1977): “An Equilibrium Characterization of the Term Structure,” *J. Financial Econ.*, 5, 177–188

Wilmott, P., J. Dewynne, and S. Howison (1993): *Option Pricing: Mathematical Models and Computation*, Oxford Financial Press

Zhu, Y., and M. Avellaneda (1997): “A Risk-Neutral Stochastic Volatility Model,” Working paper, New York University

Zhu, C., Boyd, R. H., Lu, P., and Nocedal, J. (1994): *L-BFGS-B: Fortran Subroutines for Large-scale Bound-constrained Optimization*, Northwestern University, Department of Electrical Engineering

Algorithms for Nonlinear Models in Computational Finance and their Object-oriented Implementation

by

Robert Buff

Advisor: Marco Avellaneda

Individual components of financial option portfolios cannot be evaluated independently under nonlinear models in mathematical finance. This entails increased algorithmic complexity if the options under consideration are path-dependent. We describe algorithms that price portfolios of vanilla, barrier and American options under worst-case assumptions in an uncertain volatility setting. We present a generalized approach to worst-case volatility scenarios in which only the duration, but not the starting dates of periods of high volatility risk are known. Our implementation follows object-oriented principles and is modular and extensible. Combinatorial and numerical algorithms are separate and orthogonal to each other. We make our tools available to a wide audience by using standard Internet technologies.

Algorithms for Nonlinear Models in Computational Finance and their Object-oriented Implementation

by

Robert Buff

Advisor: Marco Avellaneda

Individual components of financial option portfolios cannot be evaluated independently under nonlinear models in mathematical finance. This entails increased algorithmic complexity if the options under consideration are path-dependent. We describe algorithms that price portfolios of vanilla, barrier and American options under worst-case assumptions in an uncertain volatility setting. We present a generalized approach to worst-case volatility scenarios in which only the duration, but not the starting dates of periods of high volatility risk are known. Our implementation follows object-oriented principles and is modular and extensible. Combinatorial and numerical algorithms are separate and orthogonal to each other. We make our tools available to a wide audience by using standard Internet technologies.