University of Piraeus

School of Finance and Statistics
Department of Banking and Management
M.Sc. in Banking and Finance
Banking and Financial Management

A LATTICE MODEL FOR PRICING INTEREST RATE DERIVATIVES

Nikolaos Kaklidakis MXRH 2408

Supervisor: **Assistant Professor Nikolaos Egglezos**

Committee: **Professor Nikolaos Kourogenis**
Associate Professor Michail Anthropelos
Assistant Professor Nikolaos Egglezos

Piraeus, March 2026

In memory of Professor John C. Hull

“I wonder about all the roads not taken and am moved to quote Frost...but won't.”

Sylvia Plath

Abstract

This thesis investigates lattice methods for pricing interest rate derivatives when the short rate follows a one-factor arbitrage-free term structure model. Two recombining trees are considered in the analysis. The first one is a symmetric binomial lattice and the second one is a mean-reverting trinomial lattice. Both models are calibrated to the AAA euro area government bond yield curve on 31 January 2024.

The calibration is performed in order to match the observed zero-coupon term structure exactly. This is achieved by calculating recursively the sequence of shift parameters a_n at each time step of the tree. After the calibration stage, the models are used to price European call options on zero-coupon bonds. The valuation is done using backward induction through the lattice.

The numerical results show that both models reproduce the market discount factors with negligible differences. However, when the trinomial tree is constructed without restrictions, some transition probabilities become negative or greater than one. To avoid this issue, a truncated recombining trinomial tree is implemented together with boundary adjustments, while the exact fit of the initial curve is maintained.

The empirical findings show that the binomial tree produces consistently higher option prices than the truncated trinomial tree.

Keywords: Interest-Rate Derivatives, Short-Rate Models, Term Structure of Interest Rates, Binomial Tree, Trinomial Tree, Hull-White Model, Zero-Coupon Curve, Lattice Calibration, Bond Options, Backward Induction

Περίληψη

Η παρούσα διπλωματική εργασία διερευνά μεθόδους πλέγματος για την αποτίμηση παραγώγων επιτοκίων όταν το βραχυπρόθεσμο επιτόκιο ακολουθεί ένα μονοπαραγοντικό μοντέλο διάρθρωσης επιτοκίων χωρίς αρμπιτράζ. Στην ανάλυση εξετάζονται δύο ανασυνδυάζουσες δενδρικές δομές. Η πρώτη αποτελεί ένα συμμετρικό διωνυμικό πλέγμα, ενώ η δεύτερη ένα τριωνυμικό μοντέλο με μέση επιστροφή. Και τα δύο μοντέλα έχουν ρυθμιστεί στην καμπύλη αποδόσεων κρατικών ομολόγων της ευρωζώνης, με πιστοληπτική διαβάθμιση ΑΑΑ στις 31 Ιανουαρίου 2024.

Η ρύθμιση έλαβε χώρα προκειμένου να αναπαραχθεί επακριβώς η δομή της παρατηρούμενης διάρθρωσης επιτοκίων μηδενικού τοκομεριδίου. Αυτό επετεύχθη μέσω του αναδρομικού υπολογισμού της ακολουθίας των μετατοπίσεων παραμέτρου a_n σε κάθε χρονικό βήμα του δένδρου. Μετά το στάδιο της ρύθμισης, τα μοντέλα χρησιμοποιούνται για την αποτίμηση ευρωπαϊκών δικαιωμάτων προαίρεσης πάνω σε ομόλογα μηδενικού τοκομεριδίου. Η εκτίμηση γίνεται με τη χρήση της μεθόδου της οπισθογενούς επαγωγής στο πλέγμα.

Τα αριθμητικά αποτελέσματα καταδεικνύουν πως και τα δύο μοντέλα αναπαράγουν τους προεξοφλητικούς παράγοντες της αγοράς με ελάχιστες αποκλίσεις. Παρά ταύτα, όταν το τριωνυμικό δένδρο κατασκευάζεται χωρίς περιορισμούς, ορισμένες πιθανότητες μετάβασης λαμβάνουν αρνητικές τιμές ή υπερβαίνουν τη μονάδα. Για την αποφυγή του εν λόγω προβλήματος γίνεται χρήση ενός περικομμένου ανασυνδυάζοντος τριωνυμικού δένδρου με προσαρμογές στα όρια, διατηρώντας παράλληλα την ακριβή προσαρμογή στην ακριβή καμπύλη.

Τα εμπειρικά αποτελέσματα καταδεικνύουν πως το διωνυμικό δένδρο παράγει συστηματικά υψηλότερες τιμές δικαιωμάτων προαίρεσης εν συγκρίσει με το περικομμένο τριωνυμικό δένδρο.

Λέξεις-κλειδιά: Παράγωγα Επιτοκίων, Μοντέλα Βραχυπρόθεσμου Επιτοκίου, Διάρθρωση Επιτοκίων, Διωνυμικό Δένδρο, Τριωνυμικό Δένδρο, Υπόδειγμα Hull-White, Καμπύλη Μηδενικού Τοκομεριδίου, Ρύθμιση Πλέγματος, Δικαιώματα Προαίρεσης Ομολόγων, Οπισθογενής Επαγωγή

Table of Contents

1. Introduction

1.1 Research Motivation

Interest rate derivatives are central for fixed income markets, as they are used for a variety of purposes, such as hedging of future funding costs, the management of the interest rate exposure of bond portfolios and the determination of the value of claims whose cash flows depend on future interest-rate movements. However, unlike equity options, where the pricing is more straightforward as the modelling focus usually consists of one single underlying asset, they depend on the future movement of the term structure as a whole. This attribute makes the model construction of interest rate derivatives a challenging endeavor, with particular importance placed on the consistency with no-arbitrage pricing.

It is not sufficient for a term structure model to describe possible future paths of interest rates in a plausible way. The model must also be consistent with the market information available at the valuation date. Practicely the requirement for our model is to match the initial zero coupon curve in a precise manner. If we violate this condition then our subsequent results on the bond prices and the derivative values will be incosistent with the market data observed from the outset. Therefore it is deemed as a fundamental requirement for an arbitrage free pricing model for the initial term structure to fit precisely.

Lattice methods remain particularly important in this context for reasons that go beyond computational convenience. A recombining tree provides a direct representation of the evolution of the short rate over time and makes the valuation procedure transparent, since bond and derivative values can be computed at each node of the tree. Once the tree has been calibrated to the initial term structure, pricing is performed by backward induction. These features make lattice models suitable both for analytical discussion and for empirical implementation.

A recombining short-rate tree does not have a unique specification. Binomial and trinomial constructions represent the short-rate process in different ways, and they may also display different numerical properties once the initial term structure has been imposed. As a result, the structure of the lattice itself changes, which can lead to differences in derivative prices.

In this study we examine the issue within the class of Markov one-factor arbitrage - free term-structure models that are presented on recombining short-rate lattice models. Our main interest lies in the construction of an exact fit trinomial short rate tree of the Hull-White type, with attention on the exact fitting of the initial zero coupon curve, as well as the pricing of the the interest rate derivatives on the calibrated lattice models. In the case of the trinomial tree, particular importance has been placed on the numerical issue of the admissible transition probabilities.

1.2 Research Objective

The purpose of this dissertation is to present and examine the construction of recombining short rate lattices, for the pricing of interest rate derivatives within the frame of one-factor arbitrage free term structure. The analysis focuses on two specifications: a symmetric exact-fit binomial tree and a mean-reverting trinomial tree.Both lattices are calibrated to the same observed zero-coupon curve and then used to price European call options on zero-coupon bonds. The numerical behavior of the two specifications is compared, together with a small set of sensitivity checks.

1.3 Contribution of the Thesis

This dissertation studies one-factor arbitrage-free short-rate models that are represented on recombining lattice structures. The main objective is the exact fitting of the initial zero-coupon yield curve and the pricing of interest-rate derivatives using the calibrated tree.

Special attention is given to the trinomial specification, which is the main reference model of the analysis. In the numerical implementation it is found that the unrestricted trinomial tree produces probability violations for the selected parameter values. To address this problem, a truncated recombining trinomial tree with boundary adjustments is used, while keeping the exact fit to the initial term structure.

In addition, a symmetric binomial tree with exact fit is constructed under the same market curve and is used as a benchmark. This makes possible a direct numerical comparison between the two lattice models. Therefore, the dissertation contributes both to the construction of the models and to the comparison of their implications for bond-option prices.

1.4 Structure of the Thesis

The dissertation consists of six chapters.

Chapter 1 introduces the topic of the study. It explains why lattice models are useful in the pricing of interest-rate derivatives and it presents the objective of the dissertation. The chapter also briefly places the work within the literature on arbitrage-free term structure models with one factor.

Chapter 2 provides a review of the relevant literature. First the main approaches to arbitrage-free term structure modelling are discussed. Afterwards the focus moves to short-rate tree models and the calibration of these models to the initial yield curve. Particular attention is given to the paper of Hull and White (1993), since their exact-fit trinomial tree construction is an important reference for the analysis developed later.

Chapter 3 presents the mathematical background that is used in the study. The chapter introduces the basic fixed-income concepts and pricing relations that are required later in the analysis. These include zero-coupon bonds, discount factors, state prices and the backward induction procedure that is used in lattice valuation.

Chapter 4 describes the construction of the binomial and trinomial short-rate lattices. The chapter explains how the recombining trees are built and how the deterministic shift sequence is used in order to match the initial zero-coupon yield curve. It also presents the pricing relations used for bonds and interest-rate derivatives. In the case of the trinomial tree, some discussion is also given to admissibility conditions and to the boundary adjustments that appear in the truncated implementation.

Chapter 5 presents the numerical analysis. In this chapter the market data are introduced and the calibration of the trees to the observed zero-coupon curve is described. Pricing results are reported for European call options on zero-coupon bonds. The chapter also examines the sensitivity of the results with respect to the time step, the mean-reversion parameter, the volatility parameter and the strike price. A comparison between the binomial and trinomial trees is also presented.

Chapter 6 provides the conclusions of the dissertation. The main findings are summarized and some limitations of the analysis are discussed. Finally, some possible extensions for future research are briefly mentioned.

2. Literature Review

2.1 Introduction

The current thesis belongs within the literature and framework of the one-factor short-rate process represented on a recombining tree for the modelling of the term structure. Our main issue therefore lie on the construction of the lattice model and its calibration to the initial zero coupon curve, as well as the pricing of the interest rate derivatives on the calibrated tree that arises. Particular emphasis has therefore been places on the exact-fit trinomial approach that is associated with Hull and White (1993), while simultaneously a symmetric exact-fit binomial tree has been used as a benchmark for comparative purposes.

The broader research literature on arbitrage-free term-structure modelling has been advanced along different paths. The three main approaches of Hull and White (1993) have been to model discount bond prices, to model instantaneous forward rates and to model the short rate. Our present study falls within that third category, because we work with a one factor short rate process and its representation on a recombining lattice, under discrete time. Our choice has been appropriate for the purpose and the level of this thesis because model construction is directly linked with numerical implementation and derivative pricing.

Calibration has been a fundamental component within the framework that we have been called to work on, since a short rate tree is practically useful solely to the extent that it remains consistent with the observed market term structure prevailing at the valuation date. That is the reason why the literature regarding the interest-rate trees is seen departing from the earlier lattice methods that were designed to handle equity options, since in the case of the interest rate, a lattice tree is required to match the initial zero coupon curve used for the valuation, instead of just approximating the local dynamics of the process. Therefore the role of the exact fit methods is central and crucial.

In the following sections we are going to examine the literature's main parts that constitute the main framework that is relevant to our analysis, in order to prepare the ground for the modelling method that we are going to adopt. Specifically, we place particular attention on the one factor arbitrage-free short rate models, the construction of a lattice model in a manner that matches perfectly and the contributions of Hull and White (1993) that serve as the main reference for our

adoption of the trinomial tree that takes place later, while additionally we include a binomial tree as a benchmark for numerical comparison.

2.2 Alternative Approaches to Modeling the Term Structure

Historically, the literature on arbitrage-free term structure modelling developed along three main directions. The first approach models discount bond prices directly. The second approach models instantaneous forward rates. The third approach models the short rate. Hull and White (1993) review these three approaches and then focus mainly on the case where the short rate is the state variable and the term structure is driven by a one-factor Markov process.

The discount-bond approach starts from the stochastic process followed by zero-coupon bond prices across different maturities. On the other hand, the forward rate approach focuses on the evolution of instantaneous forward rates over time. Both of these approaches played an important role in the development of arbitrage-free term structure theory. At the same time, the short-rate approach is closely related to numerical pricing methods. This happens because bond prices and derivative values can be derived from the evolution of the short rate.

The topic of this dissertation belongs to the short-rate modelling literature. The central element of the analysis is a one-factor short-rate process represented with a recombining lattice. The objective is not only to describe how the short rate evolves locally in the tree. Another objective is to make sure that the model is calibrated so that it reproduces the initial zero-coupon yield curve. This characteristic differentiates the later no-arbitrage tree models from earlier lattice models that were developed for other derivative pricing problems.

Hull and White (1993) is the main reference used in this part of the literature. In their paper the authors develop a general trinomial-tree method for one-factor Markov short-rate models which fit the initial market term structure. The analysis in this dissertation follows the same line and focuses on exact fitting of the initial zero-coupon curve. A symmetric exact-fit binomial tree is also used as a benchmark.

2.3 Interest Rate Trees and Calibration to the Initial Yield Curve

Compared with the trees used in standard option pricing, interest-rate trees have to satisfy stricter requirements. In equity pricing, the tree tracks the price of an asset that is directly traded in the market. In interest-rate modelling, however, the tree tracks the short rate, or some variable connected to it. This changes the problem in a fundamental way.

What matters, therefore, is not just the path followed by the rate through time. The tree must also be constructed in such a way that it matches the zero-coupon curve at the valuation date. If it fails to do so, the model will imply bond prices that are inconsistent with market data from the beginning. Calibration, in this setting, is therefore part of the construction of the lattice and not an additional step introduced afterwards.

This became one of the main themes of the arbitrage-free term-structure literature. Ho and Lee (1986) were among the first to show that a short-rate model could be modified to fit the initial term structure exactly. A few years later, Black, Derman and Toy (1990) developed a binomial short-rate tree fitted both to the current yield curve and to a volatility term structure. Their model showed clearly that an interest-rate tree is more than a discretization technique, since it must also reflect market prices at the initial date.

After imposing this requirement, the tree is harder to construct. Several things have to be taken into account, such as the branching structure, the distance between nodes, and the time-dependent adjustments used in the calibration. These are not only technical details, because they also affect the prices produced by the model afterwards. This means that the tree has to reflect both the assumed dynamics of the short rate and the term structure observed in the market.

This is the point where Hull and White (1993) become relevant. Their paper proposes a general trinomial-tree procedure for one-factor Markov short-rate models that is consistent with the initial market data. The present study moves along the same line. In particular, it focuses on fitting the initial zero-coupon curve exactly and then using the calibrated lattice to price bond options.

2.4 Hull and White (1993)

Hull and White (1993) is the main reference for the analysis in this thesis. In their paper they study one-factor short rate models in an arbitrage-free framework. They also show how these models can be implemented using a recombining trinomial

tree. In this approach the motion of the short rate is important, but another important requirement is that the model should match the initial term structure observed in the market.

This idea is different compared to earlier models. In many earlier papers a process for the short rate is chosen first. After that the model is compared with market prices to see if the results are close. Hull and White start from the opposite direction. They take the market yield curve as given and construct the model so that the initial term structure is fitted exactly.

The trinomial tree plays an important role in this construction. Hull and White show that this type of tree can represent many one-factor Markov models and at the same time it keeps the recombining property. Because of this the trinomial tree can be used as a general numerical method and not only for one particular model.

In their paper the authors also study an extension where the model is calibrated not only to the initial yield curve but also to the volatility of discount bond yields. This part is not considered in the present analysis. Here the focus is only on fitting the initial zero-coupon curve and then pricing derivatives on the calibrated lattice. This is still consistent with the topic of the thesis, which studies exact-fit one-factor models constructed with a trinomial interest rate tree and their numerical effect on option prices.

Another point in Hull and White (1993) is the role of the tree after the calibration. Once the lattice is constructed, bond options and other non-path dependent derivatives can be valued by working backwards through the tree.

The approach followed here is similar. The trinomial tree remains the main object of the analysis. In addition, a symmetric exact-fit binomial tree is also introduced as a benchmark under the same initial term structure.

2.5 One-Factor Short-Rate Models and Their Lattice Representation

Early continuous-time short-rate models usually started from an assumption about the motion of the interest rate. For example, Vasicek-type models introduced mean reversion in a Gaussian framework. The Cox–Ingersoll–Ross model proposed a different diffusion specification, which under certain parameter restrictions keeps

the interest rate non-negative. Later research also examined other specifications, such as lognormal models and models where some parameters can change over time. These different specifications are important because they affect the behavior of bond prices, the shape of the term structure and also the way volatility appears in the model.

However, moving from a continuous-time short-rate model to a model that can be used for pricing is not immediate. When the model is applied to market data, the initial zero-coupon curve has to be matched and derivative prices must be computed numerically. At this stage lattice methods are usually used. A recombining tree provides a discrete representation of the short-rate process. Using this tree, bond prices and derivative values can be obtained with backward induction. For this reason the structure of the tree must reflect the assumptions that are made for the short-rate dynamics.

Hull and White introduce their trinomial tree procedure exactly in this context. Their method allows a wide class of one-factor Markov short-rate models to be represented on a recombining lattice, while at the same time the model fits the initial term structure. Therefore the tree is not connected only with one specific model. It can be seen as a numerical representation of a broader class of one-factor short-rate models. This is one of the reasons why the paper became very influential in the literature.

The present study follows this line. The purpose here is not to compare many different short-rate models. The main focus is on the lattice representation of a one-factor arbitrage-free short-rate model which fits the initial zero-coupon curve. The trinomial specification follows the approach of Hull and White. In addition, a symmetric exact-fit binomial tree is used as a benchmark under the same observed term structure.

2.6 Advantages and Limitations of Lattice Methods in Interest-Rate Modeling

Lattice methods continued to be important in the literature on the term structure. One reason is that they allow model specification, calibration and valuation to be treated in the same numerical framework. In short-rate models, once the tree is built and calibrated, bond prices and interest-rate derivatives can be calculated using backward induction. Hull and White (1993) describe this feature as one of the main advantages of tree models, especially when the model is one-factor and Markov.

Another reason for using lattice models is the recombining structure. If the tree is not recombining, the number of nodes becomes very large very quickly. A recombining tree avoids this problem because many paths meet again at the same node. This keeps the size of the tree under control. This is important in interest-rate models, since the tree must be detailed enough to fit the initial term structure but also small enough to allow derivative pricing. Hull and White (1993) support this idea and argue in favor of a Markov yield-curve model represented on a recombining tree.

The literature also explains that lattice models in interest-rate modelling are different from the early option-pricing trees used in equity markets. Ho and Lee (1986) proposed a no-arbitrage model which fits the initial term structure exactly. Later, Black, Derman and Toy (1990) introduced a binomial short-rate tree calibrated to the yield curve and also to a volatility structure. In these models the tree is not used only as a numerical approximation. It must also reproduce the market yield curve observed at the valuation date.

At the same time, some limitations of lattice models are also discussed in the literature. In a one-factor tree the whole term structure depends on only one source of uncertainty. This makes the model easier to use, but it also limits the possible movements of the yield curve. Hull and White (1993) note that different assumptions about the short-rate process may lead to different derivative prices. The tree is useful because it allows these differences to be examined numerically.

There are also limitations related to the structure of the tree itself. The branching structure determines how much of the local behavior of the short rate can be represented while keeping the tree recombining. For this reason the trinomial tree became more common in later studies. Boyle (1986) proposed a three-jump model as an extension of the binomial idea. Later research (Chan et al., 2008) considered the extra branch useful because it gives more flexibility in numerical approximation. Hull and White (1993) adopt the trinomial structure for similar reasons in short-rate models.

Finally, some difficulties appear when the tree is implemented numerically. Even if the model fits the initial term structure exactly, numerical stability still depends on several parameters. These include drift, volatility, node spacing and the time step. In short-rate trees this can affect whether the transition probabilities are admissible. For this reason the numerical construction of the lattice is an important part of the analysis in this dissertation.

2.7 Position of the Present Study in the Literature

The discussion in the previous section indicates that the present thesis follows the one-factor short-rate modelling framework for the arbitrage-free term structure of interest rates. An important reference for this approach is Hull and White (1993). In that paper the authors show how it is possible to construct a recombining trinomial tree which fits the initial term structure observed in the market. This idea is taken as the starting point for the modelling used in this work.

In this study the analysis focuses only on the part of the Hull–White model that deals with exact fitting of the initial zero-coupon yield curve. The later extension of the model, where the calibration also includes volatility information from the market, is not examined here. Instead, the emphasis is placed on the construction of the lattice, its calibration to the observed term structure, and the valuation of European call options on zero-coupon bonds by using backward induction.

Within this framework the trinomial tree is the main model studied in the thesis. In addition, a symmetric binomial tree which also fits the same initial term structure is implemented for comparison purposes. Therefore, the comparison is made between two lattice models that are calibrated to the same market curve, while the trinomial tree remains the main point of reference.

3. Mathematical Preliminaries

3.1 Discrete Time Setting

Let the time interval $[0, T]$ be divided into N equal subintervals of length Δt, so that

$$t_n = n\Delta t, \quad n = 0, 1, \ldots, N.$$

All prices and rates are defined on this grid. The short rate at node (n, j) is denoted by $r_{n,j}$. Since the model is one-factor, $r_{n,j}$ is the only state variable.

The market input at time 0 is the set of zero-coupon bond prices

$$P(0, t_n), \quad n = 1, \dots, N.$$

These prices are treated as given and will later be used as calibration targets. At each node (n, j), one-period discounting is written as

$$e^{-r_{n,j}\Delta t}$$

A recombining tree is assumed throughout, so that the number of nodes grows linearly with the number of time steps.

3.2 State Prices

Let $Q_{n,j}$ denote the state price of node (n, j), that is, the value at time 0 of one monetary unit received only if the process reaches node (n, j) at time t_n. At the initial node,

$$Q_{0,0} = 1$$

State prices are generated recursively by multiplying the previous state price by the corresponding transition probability and the one-period discount factor.

In a binomial tree, if the risk-neutral probability of an upward move from node (n, j) is $p_{n,j}$, then

$$Q_{n+1,j+1} \leftarrow Q_{n,j} p_{n,j} e^{-r_{n,j}\Delta t},$$
$$Q_{n+1,j} \leftarrow Q_{n,j}(1 - p_{n,j)} e^{-r_{n,j}\Delta t}$$

In a trinomial tree, if the upward, middle, and downward probabilities are p_u, p_m, p_d, then

$$Q_{n+1,j+1} \leftarrow Q_{n,j} p_u e^{-r_{n,j\Delta t}}$$

$$Q_{n+1,j} \leftarrow Q_{n,j} p_m e^{-r_{n,j\Delta t}}$$

$$Q_{n+1,j-1} \leftarrow Q_{n,j} p_d e^{-r_{n,j\Delta t}}$$

with

$$p_u + p_m + p_d = 1$$

If more than one path leads to the same node, the corresponding contributions are added.

3.3 Zero-coupon bond prices on the tree

State prices give the model-implied value of a zero-coupon bond directly. The price at time 0 of a bond maturing at t_n is

$$P(0, t_n) = \sum_j Q_{n,j}$$

Through this relation the tree is connected to the observed term structure. If the state prices are known, then the discount factors from the model can be obtained directly.

The same bond can also be priced by backward induction. If $B_{n,j}^{(m)}$ denotes at node (n, j) the value of a zero-coupon bond maturing at t_m, then

$B_{m,j}^{(m)} = 1$.

For $n = m - 1, \ldots, 0$, the bond value is found recursively.

In a binomial tree,

$$B_{n,j}^{(m)} = e^{-r_{n,j\Delta t}} \left[p_{n,j} B_{n+1,j+1}^{(m)} + (1 - p_{n,j}) B_{n+1,j}^{(m)} \right]$$

In a trinomial tree,

$$B_{n,j}^{(m)} = e^{-r_{n,j}\Delta t}\left[p_u B_{n+1,j+1}^{(m)} + p_m B_{n+1,j}^{(m)} + p_d B_{n+1,j-1}^{(m)}\right]$$

The price at the root must agree with the value obtained from state prices.

3.4 Exact fit of the initial term structure

The lattice is required to satisfy

$$P^{model}(0, t_n) = P^{market}(0, t_n), \qquad n = 1, \dots, N$$

This condition is imposed for all maturities included in the calibration set. In terms of state prices, the fitting condition becomes

$$\sum_j Q_{n,j} = P^{market}(0, t_n)$$

A convenient way to express the short rate is

$$r_{n,j} = a_n + x_{n,j}$$

The term $x_{n,j}$ determines the branching structure of the tree, while a_n is a time-dependent adjustment. The sequence a_n is chosen in order for the model to match the observed zero-coupon bond prices at time 0. The exact recursive formula depends on whether we use a binomial or trinomial tree, and this will be shown in the next chapter.

Exact fit is necessary here because the pricing of bond options and other interest-rate derivatives starts from the same discount curve. If the lattice does not match the initial term structure, pricing errors appear before derivative valuation even begins.

3.5 Derivative pricing by backward induction

We write $V_{n,j}$ for the value of the derivative at node (n, j). If we know the payoff at maturity t_N, then the value at earlier nodes can be found recursively by moving backwards through the tree.

In a binomial tree,

$$V_{n,j} = e^{-r_{n,j}\Delta t}\left[p_{n,j}V_{n+1,j+1} + (1 - p_{n,j})V_{n+1,j}\right]$$

In a trinomial tree,

$$V_{n,j} = e^{-r_{n,j}\Delta t}\left[p_u V_{n+1,j+1} + p_m V_{n+1,j} + p_d V_{n+1,j-1}\right]$$

This recursion is used later for bond options and for other interest-rate derivatives once the payoff has been defined at the relevant exercise or maturity date.

4. Construction and Calibration of One-Factor Short-Rate Lattices for Interest-Rate Derivative Pricing

4.1 Binomial Short-Rate Tree

In the binomial specification, the short rate at node (n, j) is written as

$$r_{n,j} = a_n + x_{n,j}$$

for $n = 0, 1, \dots, N \qquad j = 0, 1, \dots, n$

The term a_n is a time-dependent shift, while $x_{n,j}$ determines the position of the node in the tree.

The time grid is

$$t_n = n\Delta t$$

And the state variable is constructed with constant spacing

$$\Delta x = \sigma\sqrt{\Delta t}$$

At time $t_n$the node values are

$x_{n,j} = (2j - n)\Delta x, \qquad j = 0, 1, \ldots, n$

From node (n, j), the next step leads to either $(n + 1, j)$ or $(n + 1, j + 1)$. The corresponding values of the state variable are

$$x_{n+1,j+1} = x_{n,j} + \Delta x$$

And

$$x_{n+1,j} = x_{n,j} - \Delta x \ .$$

Hence,

$$x_{n+2,j+1} = x_{n,j} + \Delta x - \Delta x = x_{n,j} = x_{n,j} - \Delta x + \Delta x$$

So that an upward movement followed by a downward movement leads to the same node as a downward movement followed by an upward movement. The tree is therefore recombining.

The binomial benchmark that we use here is symmetric, with probability of an upward movement

$$p = \frac{1}{2}$$

And probability of a downward movement

$$1 - p = \frac{1}{2}$$

Therefore the branching structure is determined by Δx, and the fit of the initial term structure is separately imposed through the sequence a_n

Let $P_{n,j}(t_m)$ denote the value at node (n, j) of a zero coupon bond maturing at time t_m, $n < m$. At maturity,

$$P_{m,j}(t_m) = 1.$$

For $n = m - 1, m - 2, \ldots, 0$, the bond pricing recursion is

$$P_{n,j}(t_m) = e^{-r_{n,j}\Delta t}\left[\frac{1}{2}P_{n+1,j+1}(t_m) + \frac{1}{2}P_{n+1,j}(t_m)\right]$$

If $V_{n,j}$ denotes the value of a derivative security at node (n, j) , then backward induction gives

$$V_{n,j} = e^{-r_{n,j}\Delta t}\left[\frac{1}{2}V_{n+1,j+1} + \frac{1}{2}V_{n+1,j}\right]$$

Which is subject to the payoff that is specified at the relevant maturity or exercise date.

The sequence a_n is determined recursively from the exact-fit condition for the initial zero-coupon curve, and is not fixed in advance.

4.2 Trinomial Short-Rate Tree

In the trinomial specification, the short rate at node (n, j) is written as

$$r_{n,j} = a_n + x_{n,j}$$

The time grid is

$$t_n = n\Delta t$$

And the state variable is

$$x_{n,j} = j\Delta x, \qquad j = -n, -n+1, \ldots, n-1, n$$

From node (n, j) , the next step leads to $(n+1, j+1), (n+1, j)$, or $(n+1, j-1)$. TThe tree is therefore recombining.

The node spacing is taken as

$$\Delta x = \sigma\sqrt{3\Delta t}$$

where σ is the volatility parameter.

Let $\mu_{n,j}$ denote the local drift of the state variable at node (n, j). In one factor mean-reverting specification,

$$\mu_{n,j} = -ax_{n,j},$$

Where a is the mean reversion parameter.

Let $p_u(n,j), p_m(n,j)$, and $p_d(n,j)$ denote the probabilities of an upward, middle, and downward movement from node (n,j). These probabilities are obtained from

$p_u(n,j) + p_m(n,j) + p_d(n,j) = 1,$

$$\Delta x[p_u(n,j) - p_d(n,j)] = \mu_{n,j}\Delta t,$$

and

$$\Delta x^2[p_u(n,j) + p_d(n,j)] = \sigma^2\Delta t + (\mu_{n,j}\Delta t)^2$$

Solving for the probabilities gives

$$p_u(n,j) = \frac{1}{2}\left[\frac{\sigma^2\Delta t + (\mu_{n,j}\Delta t)^2}{\Delta x^2} + \frac{\mu_{n,j}\Delta t}{\Delta x}\right]$$

$$p_d(n,j) = \frac{1}{2}\left[\frac{\sigma^2\Delta t + (\mu_{n,j}\Delta t)^2}{\Delta x^2} - \frac{\mu_{n,j}\Delta t}{\Delta x}\right]$$

$$p_m(n,j) = 1 - \frac{\sigma^2\Delta t + (\mu_{n,j}\Delta t)^2}{\Delta x^2}$$

With the choice $\Delta x = \sigma\sqrt{3\Delta t}$, define

$$q_{n,j} = \frac{\mu_{n,j}\Delta t}{\Delta x}$$

Then

$$p_u(n,j) = \frac{1}{6} + \frac{q_{n,j}}{2} + \frac{q_{n,j}^2}{2}$$

$$p_m(n,j) = \frac{2}{3} - q_{n,j}^2$$

$$p_d(n,j) = \frac{1}{6} - \frac{q_{n,j}}{2} + \frac{q_{n,j}^2}{2}$$

Admissability condition:

$$0 \leq p_u(n,j), p_m(n,j), p_d(n,j) \leq 1$$

For some combinations of Δt and parameter values, this condition may be violated, even when the initial term structure is fitted exactly.

The construction of the interior node that we performed above defines the unrestricted trinomial tree. In our empirical implementation, we added boundary adjustments in order to preserve the probability admissibility at the extreme nodes.

4.3 Calibration to the Initial Zero Coupon Curve

Calibration to the observed zero-coupon curve is carried out through the sequence α_n in

$$r_{n,j} = a_n + x_{n,j}$$

At first the branching structure of the tree is determined by the binomial or trinomial specification. After this, the values of a_n are calculated recursively in order for the model to reproduce the market discount factors at the grid dates. In this way the initial term structure is fitted exactly, maturity by maturity, without modifying the recombining lattice structure.

Let

$$0 = t_0 < t_1 < \cdots < t_N, \quad t_n = n\Delta t$$

Let $P^{market}(0, t_n)$ denote market discount factor for maturity t_n. The requirement is

$$P^{model}(0,t_n) = P^{market}(0,t_n), \quad n = 1,2,\dots,N$$

In order to express this condition on the lattice, we define $Q_{n,j}$ as the state price at node (n,j). This represents the value at time 0 of one unit that is paid if the node (n,j) is reached and zero in all other cases. With this notation, the model price of a zero-coupon bond with maturity t_n is obtained by adding the state prices at time n.

$$P^{model}(0,t_n) = \sum_j Q_{n,j}$$

so that the fitting condition becomes

$$\sum_j Q_{n,j} = P^{market}(0,t_n), \qquad n = 1,2,\dots,N$$

Suppose that a_0, a_1, …, a_{n-1} have already been determined. Then the short-rate levels $r_{k,j}$ are known up to time t_{n-1}, and the state prices $Q_{n,j}$ can be computed forward through the tree. The only unown quantity in the one step discount from $t_n \rightarrow t_{n+1}$is a_n. Since

$$r_{n,j} = a_n + x_{n,j}$$

Hence

$$e^{-r_{n,j}\Delta t} = e^{-\alpha_n \Delta t} e^{-x_{n,j}\Delta t}$$

Where $e^{-a_n \Delta t}$ is common across nodes at time t_n

In the trinomial tree, the state-price recursion is

$$Q_{n+1,j} = e^{-r_{n,j-1}\Delta t} Q_{n,j-1} p_u(n,j-1) + e^{-r_{n,j}\Delta t} Q_{n,j} p_m(n,j) + e^{-r_{n,j+1}\Delta t} Q_{n,j+1} p_d(n,j+1)$$

The corresponding modified probabilities are being used at the boundary nodes.

Hence,

$$P^{model}(0, t_{n+1}) = \sum_j Q_{n+1,j}$$

By using state-price recursion and then summing over the nodes at time $t_n$we have

$$P^{model}(0, t_{n+1}) = \sum_j Q_{n,j} e^{-r_{n,j}\Delta t}$$

Since $r_{n,j} = a_n + x_{n,j}$,

$$P^{model}(0, t_{n+1}) = e^{-a_n \Delta t} \sum_j Q_{n,j} e^{-x_{n,j}\Delta t}$$

Imposing exact fit at maturity t_{n+1},

$$P^{market}(0, t_{n+1}) = e^{-a_n \Delta t} \sum_j Q_{n,j} e^{-x_{n,j}\Delta t}$$

Therefore,

$$a_n = -\frac{1}{\Delta t} \ln\left(\frac{P^{market}(0, t_{n+1})}{\sum_j Q_{n,j} e^{-x_{n,j}\Delta t}}\right)$$

Thus, a_n is determined recursively once state prices become known at time t_n

At the initial node,

$$Q_{0,0} = 1, \qquad x_{0,0} = 0$$

Therefore the first calibration step gives

$$P^{market}(0, t_1) = e^{-a_0 \Delta t}$$

And hence

$$a_0 = -\frac{1}{\Delta t} \ln P^{market}(0, t_1)$$

For $n \geq 1$, the remaining values of a_n are obtained from the same recursion.

Same principle applies in the case of the binomial tree. Once we hace fixed the symmetric branching structure, we generate the state prices through

$$Q_{n+1,j} = \frac{1}{2} e^{-r_{n,j-1} \Delta t} Q_{n,j-1} + \frac{1}{2} e^{-r_{n,j} \Delta t} Q_{n,j}$$

With adjustment at extreme nodes. We then determine recursively the sequence a_n from

$$P^{market}(0, t_{n+1}) = e^{-a_n \Delta t} \sum_j Q_{n,j} e^{-x_{n,j} \Delta t}$$

The initial zero-coupon curve is thus imposed in both of the lattices through the deterministic shift sequence

The sequence a_n is therefore obtained recursively from the market discount factors. The calibrated tree then reproduces the initial zero-coupon curve at the grid dates and is used for bond pricing and derivative valuation by backward induction.

4.4 Pricing by Backward Induction

With the calibrated short-rate tree through a_n, bond and derivative values are obtained by backward induction.

Let $B_{n,j}(t_m)$ denote the value at node (n, j) of a zero-coupon bond that pays 1 at time t_m, with $n \leq m$. At maturity,

$$B_{m,j}(t_m) = 1$$

For the binomial tree, the recursion is

$$B_{n,j}(t_m) = e^{-r_{n,j}\Delta t}\left[\frac{1}{2}B_{n+1,j+1}(t_m) + \frac{1}{2}B_{n+1,j}(t_m)\right], \qquad j = 0, \dots, n$$

For $n = m-1, m-2, \dots, 0$

For the trinomial tree, let active nodes at time n be

$$\mathcal{J}_n = \{-\min(n, J), \dots, \min(n, J)\}$$

And at interior nodes, for

$$j = -\min(n, J) + 1, \dots, \min(n, J) - 1$$

the bond-price recursion is

$$B_{n,j}(t_m) = e^{-r_{n,j}\Delta t}\left[p_u(n,j)B_{n+1,j+1}(t_m) + p_m(n,j)B_{n+1,j}(t_m) + p_d(n,j)B_{n+1,j-1}(t_m)\right]$$

At the boundary nodes, we use the modified transition probabilities, specifically, at the upper boundary $j = J$, we have

$$B_{n,J}(t_m) = e^{-r_{n,J}\Delta t}\left[p_0(n,J)B_{n+1,J}(t_m) + p_1(n,J)B_{n+1,J-1}(t_m) + p_2(n,J)B_{n+1,J-2}(t_m)\right]$$

And at the lower boundary $j = -J$ we have

$$B_{n,-J}(t_m) = e^{-r_{n,-J}\Delta t}\left[p_0(n,-J)B_{n+1,-J}(t_m) + p_1(n,-J)B_{n+1,-J+1}(t_m) + p_2(n,-J)B_{n+1,-J+2}(t_m)\right]$$

In the trinomial tree we carry out backward induction through the interior probabilities $p_u(n,j)$,$p_m(n,j)$, $p_d(n,j)$ at the interior nodes and through the modified probabilities at the boundary nodes.

The derivative prices are obtained through the same backward induction procedure. Let $V_{n,j}$ denote the value at node (n,j) of a derivative security with payoff specified at time t_s. In the binomial tree,

$$V_{n,j} = e^{-r_{n,j}\Delta t}\left[\frac{1}{2}V_{n+1,j+1} + \frac{1}{2}V_{n+1,j}\right], \quad j = 0, \dots, n$$

For $n = s-1, s-2, \dots, 0$

In the trinomial tree, for the interior nodes we have

$$V_{n,j} = e^{-r_{n,j}\Delta t}\left[p_u(n,j)V_{n+1,j+1} + p_m(n,j)V_{n+1,j} + p_d(n,j)V_{n+1,j-1}\right]$$

At j = J we have

$$V_{n,j} = e^{-r_{n,J}\Delta t}\left[p_0(n,J)V_{n+1,J} + p_1(n,J)V_{n+1,J-1} + p_2(n,J)V_{n+1,J-2}\right]$$

And at $j = -J$ we have

$$V_{n,-J} = e^{-r_{n,-J}\Delta t}\left[p_0(n,-J)V_{n+1,-J} + p_1(n,-J)V_{n+1,-J+1} + p_2(n,-J)V_{n+1,-J+2}\right]$$

For a European call option on a zero-coupon bond, with option maturity t_s, bond maturity t_m, and Strike K, the payoff at the exercise date is

$$V_{s,j} = max\{B_{s,j}(t_m) - K, 0\}$$

We obtain the option price at time 0 by applying the corresponding backward induction recursion from time t_s to the initial node, hence, bond and derivative values are computed on the calibrated lattice under the risk-neutral measure, with the trinomial tree having the incorporaton of the truncation as well as the boundary adjustment that we described above.

4.5 Remarks on Calibration and Numerical Issues

The interior trinoial probabilities are obtained from the local mean and the local second moment conditions. Let

$$q_{n,j} = \frac{\mu_{n,j}\Delta t}{\Delta x}, \quad \mu_{n,j} = -ax_{n,j}$$

Since $x_{n,j} = j\Delta x$, then

$$q_{n,j} = -aj\Delta t$$

Interior probabilities

$$p_u(n,j) = \frac{1}{6} + \frac{1}{2}q_{n,j} + \frac{1}{2}q_{n,j}^2$$

$$p_m(n,j) = \frac{2}{3} - q_{n,j}^2$$

$$p_d(n,j) = \frac{1}{6} - \frac{1}{2}q_{n,j} + \frac{1}{2}q_{n,j}^2$$

Condition for admissibility is

$$|q_{n,j}| \leq \sqrt{\frac{2}{3}}$$

Using $q_{n,j} = -aj\Delta t$ the condition becomes

$$|j| \leq \frac{\sqrt{\frac{2}{3}}}{a\Delta t}$$

Define

$$J = \left\lfloor \frac{\sqrt{2/3}}{a\Delta t} \right\rfloor$$

At time t_n, we have the active nodes restrictions to

$$j = -\min(n,J), -\min(n,J)+1, \ldots, \min(n,J)$$

For the interior nodes, which means for $-J < j < J$,

$$j = -\min(n,J), -\min(n,J)+1, \ldots, \min(n,J)-1$$

the transition probabilities remain as $p_u(n,j), p_m(n,j)$ and $p_d(n,j)$.

At the upper boundary $j = J$, the branches are redirected to the nodes $J, J-1, J-2$ with the following probabilities:

$$p_0(n,J) = \frac{7}{6} + \frac{3}{2}q_{n,J} + \frac{1}{2}q_{n,J}^2$$

$$p_1(n,J) = -\frac{1}{3} - 2q_{n,J} - q_{n,J}^2$$

$$p_2(n,J) = \frac{1}{6} + \frac{1}{2}q_{n,J} + \frac{1}{2}q_{n,J}^2$$

While at the lower boundary where $j = -J$, the branches are redirected to the nodes $j, j+1, j+2$, with the following probabilities:

$$p_0(n,-J) = \frac{7}{6} - \frac{3}{2}q_{n,-J} + \frac{1}{2}q_{n,-J}^2$$

$$p_1(n,-J) = -\frac{1}{3} + 2q_{n,-J} - q_{n,-J}^2$$

$$p_2(n,-J) = \frac{1}{6} - \frac{1}{2}q_{n,-J} + \frac{1}{2}q_{n,-J}^2$$

With this modification we manage to preserve the recombination and keep our empirical implementation, under the parameters that we will use in Chapter 5, admissible. Since the sequence a_n is still obtained recursively from the state-price condition, the exact fit of $P^{market}(0,t_n)$ is not altered, but instead the change is our treatment of the extreme nodes, where we replace the unrestricted branching rule with transitions under a boundary adjustment.

5. Numerical Analysis and Empirical Study

5.1 Numerical Analysis

This section examines the numerical behaviour of the calibrated binomial and trinomial trees. The comparison is carried out under the same initial term structure and the same parameter values.

For a maturity horizon T, the interval $[0,T]$ is divided into N time steps, with

$$\Delta t = \frac{T}{N}$$

In both trees, the short rate is written as

$$r_{n,j} = a_n + x_{n,j}$$

With a_n chosen so that the initial zero-coupon curve is fited exactly. Thus, the comparison does not concern the fit of $P(0, t_n)$, but the prices obtained from backward induction when the grid is changed.

Let $V_0^{(N)}$denote the price at the root node of a given claim on a lattice with N time steps. The numerical analysis is based on the behavior of $V_0^{(N)}$ as N increases. When a reference value V_0^{ref} is computed on a finer grid, the pricing error is written as

$$Error(N) = \left| V_0^{(N)} - V_0^{ref} \right|$$

This allows a direct comparison of the two trees for the same pricing problem.

The admissibility condition is also relevant. In the trinomial tree we have,

$$0 \leq p_u(n,j), p_m(n,j), p_d(n,j) \leq 1$$

For some choices of Δt and parameter values, this condition may be violated. Price stability under different discretizations is also examined.

5.2 Data of the Empirical Study

In the empirical analysis we use the zero-coupon term structure observed at the valuation date. The lattice model is calibrated in order to reproduce exactly the initial discount curve. For this reason, the main market input of the model is the set of discount factors $P(0, t_n)$ for the maturities considered in the study.

The data have been obtained from the ECB Data Portal and correspond to the date 31 January 2024. The series used are the AAA-rated euro area government bond yield curve spot rates, expressed with continuous compounding. The maturities used

in the analysis are 3 months, 6 months, 1 year, 2 years, 3 years, 5 years, 7 years, 10 years, 15 years, 20 years, and 30 years.

Because the ECB reports the data as yields, the zero-coupon bond prices are computed before the calibration procedure. We denote by $y(0, t_n)$ the spot rate for maturity t_n, then the market discount factor should be written as

$$P^{market}(0, t_n) = e^{-y(0,t_n)t_n}.$$

The lattice model is established on the discrete grid $t_n = n\Delta t$.In the case of chosen observed market maturities not coinciding precisely with this grid, interpolation is being used to obtain the requisite values of $P^{market}(0, t_n)$. Subsequently, these discount factors are used in the recursive determination of the sequence a_n, as a means for the reproduction of the initial term structure by the calibrated tree.

Therefore the observed zero-coupon curve is being used as the common input for both the binomial and the trinomial tree in our empirical study. In the following sections, the comparison that takes place is based on the prices that were generated by the two calibrated lattices under the same term-structure data.

5.3 Model Parameter Specification

For the empirical part of the thesis the two trees are built using the same market structure. The idea is simply that the model should reproduce the discount factors that we observe in the market. For this reason the lattice is calibrated so that the model prices at the grid dates coincide with the market ones.

The data that we use come from the ECB Data Portal. In particular we downloaded the AAA euro area government bond yield curve. The valuation date of the dataset is 31 January 2024. From the full curve we keep several maturities which are used in the empirical exercise. These are 3 months, 6 months, 1 year, 2 years, 3 years, 5 years, 7 years, 10 years, 15 years, 20 years, and 30 years.

The ECB reports the information in the form of yields. Because of this we first transform the yields into discount factors before using them in the lattice. Assuming continuous compounding, the discount factor corresponding to maturity t is calculated as

$$P^{market}(0,t) = e^{-y(0,t)t}$$

In some cases the maturity from the dataset does not fall exactly on one of the lattice dates. When this happens we estimate the required value by interpolating the initial zero-coupon curve

For the numerical implementation we take a horizon equal to 30 years. The time step is chosen equal to 0.25 years. With this choice the lattice contains 120 time steps. The volatility parameter that is used in both trees is

$$\sigma = 0.01.$$

For the binomial model we use the symmetric specification that was described earlier. The distance between neighboring states of the short rate process is taken equal to

$$\Delta x = \sigma\sqrt{\Delta t}$$

The probabilities for an upward or downward movement are both equal to one half.

For the trinomial lattice we work with the truncated recombining tree. The step between neighboring states of the process is

$$\Delta x = \sigma\sqrt{3\Delta t}$$

Mean reversion is introduced through a drift term and the parameter used in the empirical case is 0.05. Inside the tree the standard trinomial probabilities are applied, while at the outer nodes they are modified slightly in order to keep them admissible.

European call options on zero-coupon bonds are used in the pricing exercise. The maturity pairs analysed are $(1,5), (2,10),$ and $(5,20)$.

$$V_{s,j} = \max\{B_{s,j}(t_m) - K, 0\},$$

The strike convention used in the baseline specification is forward at the money strike, defined by

$$K = \frac{P^{market}(0, t_m)}{P^{market}(0, t_s)}$$

The root value $V_{0,0}$ is then obtained by backward induction on the calibrated tree.

In the next section we compare the results from the two trees. Both are calibrated to the same curve, so the differences in prices come from the different model specifications and not from the fit of the initial term structure.

5.4 Pricing Results on the Calibrated Trees

The calibration of the lattice models was tested by comparing the bond prices from the trees with the market zero-coupon curve. For the maturities included in the empirical analysis we compared the root prices of the bonds generated by the calibrated trees with the market discount factors $P^{market}(0, t_n)$. In the binomial tree, the maximum absolute deviation was $3.33\ x\ 10^{-16}$. For the truncated trinomial tree the maximum absolute deviation was $4.44\ x\ 10^{-16}$. These deviations are extremely small. Most probably they are caused by numerical rounding in the calculations. Because of this both models reproduce the initial term structure almost exactly.

The trinomial lattice prices used in this chapter are obtained from a truncated recombining tree with adjusted boundaries. In the unrestricted trinomial case we also obtained exact fit of the initial curve. However, some transition probabilities were inadmissible in that specification. By applying truncation these problems were removed while the exact fit remained. Therefore the derivative prices presented later are based on this admissible specification.

In the baseline pricing exercise we consider three European call options on zero-coupon bonds. These are a 1-year option on a 5-year bond, a 2-year option on a 10-year bond and a 5-year option on a 20-year bond. In all three cases the strike price is

equal at the forward ATM bond price. The option price at the root node is obtained by backward induction on the calibrated lattices.

In the case of the 1 year option on the 5-year bond, our binomial tree gives us a price of 0.013696 while our truncated trinomial tree gives us 0.012095. In the case of the 2-year option on the 10 year bond, our results of the corresponding prices are 0.035625 and 0.028011, respectively. In the case of the 5 year option on the 20 year bond, our results of the corresponding prices are 0.081591 and 0.051295. In all of the three cases, we see that the binomial price is higher than the price of the trinomial model.

The absolute differences between our two lattice models are reported as follows: 0.001601, 0.007614 and 0.030296. We see that the difference is minor in our shortest contract and there exists a positive association between the size of the contract and the increase of the maturity date of the option and the bond. This pattern is received as an empirical feature of the reported contract and the current parameterization taking place.

Taking into account that both of our trees have matched the same initial zero coupon curve exactly in essence, it follows that the cause of the different prices is not a different fit of the $P^{market}(0, t_n)$. The prices differences reflect differences between the two implemented specifications that have taken place, which are the symmetric exact fit binomial benchmark and the truncated mean reverting trinomial tree. Therefore we see that the comparison should not be interpreted as an isolation of the effect of branching structure alone.

Under the baseline specification that took place, we see the production of lower option values by the truncated trinomial tree, in comparison to the binomial benchmark, in all reported cases. We also see that this effect remains stable across all three contracts that we considered here, and provides the inception for the sensitivity analysis that is going to follow.

5.5 Sensitivity and Robustness Analysis

Our baseline results were examined further through the change of numerical inputs, one at a time, and subsequently through the recalibration of the lattices on the same market discount curve.

Our reference case remains as follows:

- $T = 30$
- $\Delta t = 0.25$
- $N = 120$
- $a = 0.05$
- $\sigma = 0.01$
- Forward at the money strikes K_{ATM} for the three European call options on zero-coupon bonds

In our sensitivity analysis we considered changes in $\Delta t, a, \sigma$ and K. Even though in the case of the unrestricted trinomial tree we encountered the construction of inadmissible probabilities, under the baseline specification, no such violations were encountered in our truncated implementation for the grids as well as the parameter values that are reported in this section.

We start by testing the time step Δt. Three cases are used. The grid is 0.50, after that 0.25, and finally 0.125. The option prices change when we refine the grid, but the change is not large.

For the 1-year option on the 5-year bond the binomial price changes from 0.012965 to 0.013696 and then to 0.014078. The truncated trinomial price becomes 0.011260, then 0.012095 and then 0.012463. For the 2-year option on the 10-year bond the binomial model gives 0.034861, 0.035625 and 0.035937. The trinomial model gives 0.027443, 0.028011 and 0.028213. For the 5-year option on the 20-year bond the binomial results are 0.082111, 0.081591 and 0.080976, while the trinomial results are 0.051687, 0.051295 and 0.050951.

Since the results do not change very much we keep $\Delta t = 0.25$ as the main case in the analysis.

After this we study the mean-reversion parameter a. In the binomial model the prices stay constant even if a changes. But in the truncated trinomial model the effect is visible.

For the 1-year option the price decreases from 0.012951 when $a = 0.02$ to 0.012095 when $a = 0.05$ and then to 0.010812 when $a = 0.010$. For the 2-year option the prices are 0.032233, 0.028011 and 0.022391. For the longest option

they are 0.067224, 0.051295 and 0.033917. This means higher mean reversion leads to lower option value.

We also see that the sensitivity to σ is more direct. We notice that as the volatility rises, there is an increase in the option values in both of the lattices. In the case of the 1-year option on the 5-year bond, we see an increase in the binomial price from 0.006810 at $\sigma = 0.005$ to 0.013696 at $\sigma = 0.01$ and 0.020656 at $\sigma = 0.015$, while in the case of the truncated trinomial lattice the prices are $0.006018, 0.012095$ and 0.018230, respectively. In the case of the 2-year option on the 10-year bond, the binomial prices are $0.017640, 0.035625$ and 0.053873, and the truncated trinomial prices are $0.013895, 0.028011$ and 0.042309. In the case of the 5 year option on the 20 year bond, the binomial prices rise from 0.040722 to 0.081591 and 0.120782, and the trinomial prices rise from 0.025480 to 0.051295 and 0.076917. Our results remain consistent with the role of σ in widening the dispersion of future short-rate paths, which leeds to the increase of the value of the payoff of the option.

The strike sensitivity gives us the expected ordering. When we reduce the strike from K_{ATM} to $0.95K_{ATM}$, both trees experience an increase in the option prices, while when we increase the strike from K_{ATM} to $1.05K_{ATM}$, both trees experience a decrease in the option price. For the 2-year option on the 10-yaear bond, change in the binomial price is from 0.059504 at $0.95K_{ATM}$ to 0.035625 at K_{ATM} and 0.020953 at $1.05K_{ATM}$, and the trinomial prices are 0.052184, 0.028011 and 0.012993, respectively. We see a similar pattern in the other two contracts, and the same ordering is preserved in both lattices. When we have a lower strike price, the option value tends to be higher, and inversely, when we have a higher strike price, the option value tends to be lower.

Our results of the sensitivity analysis remained broadly consistent with the baseline findings. Our two lattice models continue reproducing the same initial term structure, under the reported parameter changes that took place, while the implementation of the truncated trinomial model remains admissible, and the price of the trinomial model is exceeded by the price of the binomial model in all contracts that we considered. Therefore we deem that the difference in price must be read as arising from the two implemented lattice specifications combined, rather than from the branching structure alone.

5.6 Comparison of Binomial and Trinomial Trees

Our numerical results showcase that our two lattice specifications have an analogous starting point. Both of the lattice reproduce the market zero-coupon curve almost precisely. In both of our models, we have calibration errors that are of the order of machine precision, and therefore the comparison between the two trees does not take place under a different fit of the initial term structure.

In the case of the unrestricted trinomial tree, while the initial term structed was still fitted exactly, certain transition probabilities became inadmissible under the baseline specification. That is the reason our empirical implementation uses a truncated recombining trinomial tree, adjusted with boundaries. In that way, we can retain the exact fit of the initial curve while removing the probability violations that take place in the unrestricted trinomial tree.

With the baseline specification, we see that the binomial tree gives higher option prices compared to the truncated trinomial tree in all of the three contracts. In the case of the 1-year option on the 5-year bond, the pries are 0.013696 and 0.012095, respectively. In the case of the 2-year option on the 10 year bond, the corresponding prices are 0.035625 and 0.028011. Lastly, in the 5-year option on the 20-year bond, the prices are 0.081591 and 0.051295. We see that the absolute difference is positive in every case, to be exact, 0.001601, 0.007614 and 0.030296, respectively, and that the difference becomes larger as the maturity increases.

Furthermore, we see that the sensitivity analysis is not differentiated from this pattern. When we change the time step from $\Delta t = 0.5$ to $\Delta t = 0.25$, and then to $\Delta t = 0.125$, there is a moderate movement in the price levels, and the binomial price remains above the truncated trinomial price in all of the three contracts. The same observation takes place when the strike changes around the forward ATM level, as well as when the σ is varied. In the case of volatility, we see that the option values increase in both trees as σ rises, and that the price of the binomial model remains higher than the price of the truncated trinomial model, all throughout.

In the case of the effect of the mean reversion parameter a, we observe that it is different across the two specifications in our present implementation. We observe that there is an inverse relationship between the prices of the trinomial model and the value of the mean reversion parameter, while there is no effect of α in the binomial prices, because the current binomial specification does not include mean reversion. This is immensely important for our interpretation of the comparison.

These results should not be interpreted as evidence that the cause of the pricing differences is solely the number of branches in the lattice models, because although we have calibrated both trees to the same initial zero-coupon curve, there is difference in their dynamic structure. The binomial model has been treated as a symmetric exact-fit benchmark, whereas the truncated trinomial tree model additionally includes mean reversion, through the local drift and transition probabilities, under the boundary truncation that takes place as a necessity for the admissibility of the model. Therefore, we conclude that the cause of the difference in price should be considered the difference between the specifications of each implemented model.

In all reported cases, the binomial benchmark produces a higher yield in the bond option price, compared to the truncated trinomial tree. This relationship remains stable under the baseline parameterization, as well as throughout the sensitivity analysis.

6. Conclusions

This dissertation studies recombining short-rate trees which are used for pricing interest-rate derivatives. The modelling framework is a one-factor arbitrage-free term structure model. In the analysis two different lattice specifications are used and both are calibrated using the same observed zero-coupon curve. The first tree is a symmetric binomial tree which fits the initial term structure exactly. The second tree is a trinomial model where mean reversion is included. In both cases the initial term structure is imposed by calculating recursively the shift parameter a_n. In this way the model reproduces the market discount factors at the grid dates.

The modelling approach follows mainly the framework proposed by Hull and White (1993). In that work the authors explain how a recombining trinomial tree can be constructed. In this dissertation more attention is given to this trinomial construction and to the calibration procedure used in derivative pricing. In addition a symmetric binomial tree is implemented and used as a benchmark model.

The empirical analysis uses the AAA-rated euro area government bond yield curve for the date 31 January 2024. The market discount factors are obtained from the observed spot rates and then interpolated on the lattice grid. After calibration the trees are used for pricing European call options on zero-coupon bonds by backward induction.

The numerical results show that both lattice models reproduce the market discount factors with negligible error. Therefore the exact-fit property is achieved. However the unrestricted trinomial tree leads to transition probabilities which are not admissible under the baseline parameterization. For this reason a truncated recombining trinomial tree with boundary adjustments is used. With this modification the exact fit of the initial curve is preserved while admissible probabilities are obtained.

In the empirical results the binomial tree gives higher option prices compared to the truncated trinomial tree for all contracts that are examined. The same ordering is also observed in the sensitivity analysis. Changing the time step does not affect the comparison. When volatility increases the option prices increase in both models. In the mean-reversion exercise only the trinomial prices change because mean reversion appears only in that specification.

The results should be interpreted with some care. The comparison presented here is not strictly a comparison only between binomial and trinomial branching. The binomial tree is used as a symmetric benchmark which fits exactly the initial term structure. In contrast, the trinomial tree also contains mean reversion and a truncation rule with boundary adjustments so that the probabilities remain admissible. For this reason, the pricing differences observed in the empirical analysis should be viewed as differences between the two implemented lattice models under the same initial term structure.

There are also some limitations in the present analysis. The empirical study is based on only one valuation date and uses one-factor short-rate models. Furthermore, the derivative contracts examined are only European call options on zero-coupon bonds. Another limitation is that the extension of the Hull–White method for fitting the initial volatility data was not implemented. Possible extensions of the work could include the study of additional interest-rate derivatives and also comparisons with more complex models, such as multi-factor models or alternative lattice frameworks.

7. References

Papers

1. Black, F., Derman, E. and Toy, W. (1990), ‘A One-Factor Model of Interest Rates and Its Application to Treasury Bond Options’, Financial Analysts Journal, Vol. 46, No. 1, pp. 33–39.
2. Boyle, P. P. (1986), ‘Option Valuation Using a Three-Jump Process’, International Options Journal, Vol. 3, pp. 7–12.
3. Chan, J. H., Joshi, M., Tang, R. and Yang, C. (2008), ‘Trinomial or Binomial: Accelerating American Put Option Price on Trees’, Journal of Futures Markets, Vol. 29, No. 9, pp. 826–839.
4. Cox, J. C., Ingersoll, J. E. and Ross, S. A. (1985), ‘A Theory of the Term Structure of Interest Rates’, Econometrica, Vol. 53, No. 2, pp. 385–407.
5. Ho, T. S. Y. and Lee, S.-B. (1986), ‘Term Structure Movements and Pricing Interest Rate Claims’, Journal of Finance, Vol. 41, No. 5, pp. 1011–1029.
6. Hull, J. and White, A. (1993), ‘One-Factor Interest-Rate Models and the Valuation of Interest-Rate Derivative Securities’, Journal of Financial and Quantitative Analysis, Vol. 28, No. 2, pp. 235–254.
7. Vasicek, O. A. (1977), ‘An Equilibrium Characterization of the Term Structure’, Journal of Financial Economics, Vol. 5, No. 2, pp. 177–188.

Books

1. Hull, J. C. (2018) Options, Futures, and Other Derivatives. 10th edn. Pearson: Harlow.
2. McKinney, W. (2022) Python for Data Analysis. 3rd edn. O’Reilly Media: Sebastopol, CA.

Python Libraries

1. pandas
2. numpy
3. pathlib

8. Appendix – Python Code

Preparing the Initial Yield Curve

```
valuation_date = "2024-01-31"
df_date = df[df["DATE"] == valuation_date].copy()


if df_date.empty:
    raise ValueError(f"No row found for DATE = {valuation_date}")



if len(df_date) != 1:
    raise ValueError(f"Expected 1 row for {valuation_date}, found {len(df_date)}
rows")


column_map = {
    "Yield curve spot rate, 3-month maturity - Government bond, nominal, all
issuers whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_3M)": "3M",
    "Yield curve spot rate, 6-month maturity - Government bond, nominal, all
issuers whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_6M)": "6M",
    "Yield curve spot rate, 1-year maturity - Government bond, nominal, all issuers
whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_1Y)": "1Y",
    "Yield curve spot rate, 2-year maturity - Government bond, nominal, all issuers
whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_2Y)": "2Y",
    "Yield curve spot rate, 3-year maturity - Government bond, nominal, all issuers
whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_3Y)": "3Y",
    "Yield curve spot rate, 5-year maturity - Government bond, nominal, all issuers
whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_5Y)": "5Y",
    "Yield curve spot rate, 7-year maturity - Government bond, nominal, all issuers
whose rating is triple A - Euro area (changing composition)
(YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_7Y)": "7Y",
```

```
    "Yield curve spot rate, 10-year maturity - Government bond, nominal, all issuers whose rating is triple A - Euro area (changing composition) (YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_10Y)": "10Y",
    "Yield curve spot rate, 15-year maturity - Government bond, nominal, all issuers whose rating is triple A - Euro area (changing composition) (YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_15Y)": "15Y",
    "Yield curve spot rate, 20-year maturity - Government bond, nominal, all issuers whose rating is triple A - Euro area (changing composition) (YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_20Y)": "20Y",
    "Yield curve spot rate, 30-year maturity - Government bond, nominal, all issuers whose rating is triple A - Euro area (changing composition) (YC.B.U2.EUR.4F.G_N_A.SV_C_YM.SR_30Y)": "30Y",
}
curve_row = df_date[list(column_map.keys())].rename(columns=column_map)
maturity_years = {
    "3M": 0.25,
    "6M": 0.50,
    "1Y": 1.0,
    "2Y": 2.0,
    "3Y": 3.0,
    "5Y": 5.0,
    "7Y": 7.0,
    "10Y": 10.0,
    "15Y": 15.0,
    "20Y": 20.0,
    "30Y": 30.0,
}


curve_series = curve_row.iloc[0]


curve_table = pd.DataFrame({
    "maturity_label": curve_series.index,
    "maturity_years": [maturity_years[label] for label in curve_series.index],
    "spot_rate_percent": curve_series.values
})
```

```
curve_table["spot_rate_decimal"] = curve_table["spot_rate_percent"] / 100.0
curve_table["discount_factor"] = np.exp(
   -curve_table["spot_rate_decimal"] * curve_table["maturity_years"]
)

curve_table = curve_table.sort_values("maturity_years").reset_index(drop=True)

print("\nClean curve table for", valuation_date)
print(curve_table)



output_file = results_path / "curve_2024_01_31.csv"
curve_table.to_csv(output_file, index=False)

print(f"\nSaved cleaned curve to: {output_file}")
curve = pd.read_csv(input_file)

print("Input curve:")
print(curve)
T = 30.0
delta_t = 0.25
grid_times = np.arange(delta_t, T + delta_t, delta_t)

if grid_times[-1] > T + 1e-12:
   raise ValueError("Grid exceeds final horizon T")

observed_times = curve["maturity_years"].to_numpy()
observed_rates = curve["spot_rate_decimal"].to_numpy()

interp_rates = np.interp(grid_times, observed_times, observed_rates)

interp_discount_factors = np.exp(-interp_rates * grid_times)
```

```
interp_table = pd.DataFrame({
    "t": grid_times,
    "spot_rate_decimal": interp_rates,
    "discount_factor": interp_discount_factors
})

print("\nInterpolated baseline curve:")
print(interp_table.head(12))
print("\nLast rows:")
print(interp_table.tail(12))
output_file = results_path / "interpolated_curve_baseline_dt_0_25.csv"
interp_table.to_csv(output_file, index=False)
print(f"\nSaved interpolated baseline curve to: {output_file}")
print(f"Number of grid points: {len(interp_table)}")
```

Exact-Fit Binomial Tree Calculation

```
curve = pd.read_csv(input_file)

t_grid = curve["t"].to_numpy()
P_market = curve["discount_factor"].to_numpy()

N = len(t_grid)
delta_t = t_grid[0]

if not np.allclose(np.diff(t_grid), delta_t):
    raise ValueError("The time grid is not equally spaced.")



sigma = 0.01
delta_x = sigma * np.sqrt(delta_t)
```

```
Q_tree = [np.array([1.0])]
x_tree = []
r_tree = []
p_tree = []
a_vec = []
tree_discount_factors = []


def x_nodes_binomial(n, delta_x):
    j_vals = np.arange(n + 1)
    return (2 * j_vals - n) * delta_x


def p_up_binomial(n):
    return np.full(n + 1, 0.5)


for n in range(N):
    Q_n = Q_tree[n]
    x_n = x_nodes_binomial(n, delta_x)

    denom = np.sum(Q_n * np.exp(-x_n * delta_t))
    a_n = -(1.0 / delta_t) * np.log(P_market[n] / denom)

    r_n = a_n + x_n
    p_n = p_up_binomial(n)

    x_tree.append(x_n.copy())
    r_tree.append(r_n.copy())
    p_tree.append(p_n.copy())
    a_vec.append(a_n)

    discounted_mass = Q_n * np.exp(-r_n * delta_t)
```

```
    Q_next = np.zeros(n + 2)

    for j in range(n + 1):
        Q_next[j] += (1.0 - p_n[j]) * discounted_mass[j]
        Q_next[j + 1] += p_n[j] * discounted_mass[j]

    Q_tree.append(Q_next)
    tree_discount_factors.append(Q_next.sum())



calibration_table = pd.DataFrame({
    "n": np.arange(1, N + 1),
    "t": t_grid,
    "P_market": P_market,
    "P_tree": np.array(tree_discount_factors),
})

calibration_table["abs_error"] = np.abs(
    calibration_table["P_market"] - calibration_table["P_tree"]
)



a_table = pd.DataFrame({
    "n": np.arange(N),
    "t_n": np.arange(N) * delta_t,
    "a_n": np.array(a_vec)
})



print("\nBinomial calibration table (first rows):")
print(calibration_table.head(12))
```

```
print("\nBinomial calibration table (last rows):")
print(calibration_table.tail(12))

print("\nMaximum absolute calibration error:")
print(calibration_table["abs_error"].max())

print("\nFirst values of a_n:")
print(a_table.head(12))

calibration_file = results_path / "binomial_calibration_check.csv"
a_file = results_path / "binomial_shift_a.csv"

calibration_table.to_csv(calibration_file, index=False)
a_table.to_csv(a_file, index=False)

print(f"\nSaved calibration table to: {calibration_file}")
print(f"Saved a_n table to: {a_file}")
```

Truncated Trinomial Tree Calculation

```
curve = pd.read_csv(input_file)
t_grid = curve["t"].to_numpy()
P_market = curve["discount_factor"].to_numpy()

N = len(t_grid)
delta_t = t_grid[0]

if not np.allclose(np.diff(t_grid), delta_t):
    raise ValueError("The time grid is not equally spaced.")

mean_reversion = 0.05
sigma = 0.01
delta_x = sigma * np.sqrt(3.0 * delta_t)
```

```
J = int(np.floor(np.sqrt(2.0 / 3.0) / (mean_reversion * delta_t)))

if J < 2:
    raise ValueError("J is too small. Change mean_reversion or delta_t.")

def interior_probabilities(q):
    pu = 1.0 / 6.0 + 0.5 * q + 0.5 * q**2
    pm = 2.0 / 3.0 - q**2
    pd = 1.0 / 6.0 - 0.5 * q + 0.5 * q**2
    return pu, pm, pd


def upper_boundary_probabilities(q):
    p_to_j   = 7.0 / 6.0 + 1.5 * q + 0.5 * q**2
    p_to_jm1 = -1.0 / 3.0 - 2.0 * q - q**2
    p_to_jm2 = 1.0 / 6.0 + 0.5 * q + 0.5 * q**2
    return p_to_j, p_to_jm1, p_to_jm2

def lower_boundary_probabilities(q):
    p_to_j   = 7.0 / 6.0 - 1.5 * q + 0.5 * q**2
    p_to_jp1 = -1.0 / 3.0 + 2.0 * q - q**2
    p_to_jp2 = 1.0 / 6.0 - 0.5 * q + 0.5 * q**2
    return p_to_j, p_to_jp1, p_to_jp2


Q_tree = [np.array([1.0])]
a_vec = []
tree_discount_factors = []
all_probabilities = []
```

```
for n in range(N):
    m_n = min(n, J)
    j_vals = np.arange(-m_n, m_n + 1)

    Q_n = Q_tree[n]
    if len(Q_n) != len(j_vals):
        raise ValueError(f"Mismatch at step n={n}: len(Q_n)={len(Q_n)},
len(j_vals)={len(j_vals)}")

    x_n = j_vals * delta_x

    denom = np.sum(Q_n * np.exp(-x_n * delta_t))
    a_n = -(1.0 / delta_t) * np.log(P_market[n] / denom)
    a_vec.append(a_n)

    r_n = a_n + x_n
    discounted_mass = Q_n * np.exp(-r_n * delta_t)

    m_next = min(n + 1, J)
    j_next = np.arange(-m_next, m_next + 1)
    Q_next = np.zeros(len(j_next))

    for i, j in enumerate(j_vals):
        x_current = j * delta_x
        q = (-mean_reversion * x_current) * delta_t / delta_x

        if (-J < j < J):
            p_up, p_mid, p_down = interior_probabilities(q)
            all_probabilities.extend([p_up, p_mid, p_down])

            Q_next[(j + 1) - j_next[0]] += p_up * discounted_mass[i]
            Q_next[j - j_next[0]] += p_mid * discounted_mass[i]
            Q_next[(j - 1) - j_next[0]] += p_down * discounted_mass[i]
        elif j == J:
```

```
            p0, p1, p2 = upper_boundary_probabilities(q)
            all_probabilities.extend([p0, p1, p2])

            Q_next[j - j_next[0]] += p0 * discounted_mass[i]
            Q_next[(j - 1) - j_next[0]] += p1 * discounted_mass[i]
            Q_next[(j - 2) - j_next[0]] += p2 * discounted_mass[i]

        elif j == -J:
            p0, p1, p2 = lower_boundary_probabilities(q)
            all_probabilities.extend([p0, p1, p2])

            Q_next[j - j_next[0]] += p0 * discounted_mass[i]
            Q_next[(j + 1) - j_next[0]] += p1 * discounted_mass[i]
            Q_next[(j + 2) - j_next[0]] += p2 * discounted_mass[i]

    Q_tree.append(Q_next)
    tree_discount_factors.append(Q_next.sum())



calibration_table = pd.DataFrame({
    "n": np.arange(1, N + 1),
    "t": t_grid,
    "P_market": P_market,
    "P_tree": np.array(tree_discount_factors)
})
calibration_table["abs_error"] = np.abs(
    calibration_table["P_market"] - calibration_table["P_tree"]
)

a_table = pd.DataFrame({
    "n": np.arange(N),
    "t_n": np.arange(N) * delta_t,
    "a_n": np.array(a_vec)
```

```
})

all_probabilities = np.array(all_probabilities)

probability_summary = pd.DataFrame({
    "mean_reversion": [mean_reversion],
    "sigma": [sigma],
    "delta_t": [delta_t],
    "delta_x": [delta_x],
    "J": [J],
    "probability_min": [all_probabilities.min()],
    "probability_max": [all_probabilities.max()],
    "number_of_violations": [int(np.sum((all_probabilities < 0) | (all_probabilities > 1)))]
})


print("\nTruncated trinomial calibration table (first rows):")
print(calibration_table.head(12))

print("\nTruncated trinomial calibration table (last rows):")
print(calibration_table.tail(12))

print("\nMaximum absolute calibration error:")
print(calibration_table["abs_error"].max())

print("\nFirst values of a_n:")
print(a_table.head(12))

print("\nProbability summary:")
print(probability_summary)

calibration_file = results_path / "trinomial_calibration_check_truncated.csv"
a_file = results_path / "trinomial_shift_a_truncated.csv"
```

```
prob_file = results_path / "trinomial_probability_summary_truncated.csv"

calibration_table.to_csv(calibration_file, index=False)
a_table.to_csv(a_file, index=False)
probability_summary.to_csv(prob_file, index=False)

print(f"\nSaved calibration table to: {calibration_file}")
print(f"Saved a_n table to: {a_file}")
print(f"Saved probability summary to: {prob_file}")
```

Zero-Coupon Bond Pricing Calculation Bond Option Pricing Calculation

```
import pandas as pd
import numpy as np
from pathlib import Path
pd.set_option("display.max_columns", None)
pd.set_option("display.width", 200)
pd.set_option("display.max_colwidth", None)

#
curve = pd.read_csv(input_file)

t_grid = curve["t"].to_numpy()
P_market = curve["discount_factor"].to_numpy()

N = len(t_grid)
delta_t = t_grid[0]

if not np.allclose(np.diff(t_grid), delta_t):
    raise ValueError("The time grid is not equally spaced.")
```

```
mean_reversion = 0.05
sigma = 0.01

def build_binomial_tree(P_market, delta_t, sigma):
    N = len(P_market)
    delta_x = sigma * np.sqrt(delta_t)

    Q_tree = [np.array([1.0])]
    a_vec = []
    r_tree = []
    p_up_tree = []

    for n in range(N):
        j_idx = np.arange(n + 1)
        x_n = (2 * j_idx - n) * delta_x
        Q_n = Q_tree[n]

        denom = np.sum(Q_n * np.exp(-x_n * delta_t))
        a_n = -(1.0 / delta_t) * np.log(P_market[n] / denom)

        r_n = a_n + x_n
        p_up_n = np.full(n + 1, 0.5)

        a_vec.append(a_n)
        r_tree.append(r_n)
        p_up_tree.append(p_up_n)

        discounted_mass = Q_n * np.exp(-r_n * delta_t)
        Q_next = np.zeros(n + 2)

        for j in range(n + 1):
            Q_next[j] += (1.0 - p_up_n[j]) * discounted_mass[j]
            Q_next[j + 1] += p_up_n[j] * discounted_mass[j]
```

```
        Q_tree.append(Q_next)

    return {
        "delta_t": delta_t,
        "sigma": sigma,
        "delta_x": delta_x,
        "a_vec": np.array(a_vec),
        "r_tree": r_tree,
        "p_up_tree": p_up_tree,
    }

def backward_binomial(tree, maturity_step, terminal_values, stop_step=0):
    values = terminal_values.copy()
    delta_t = tree["delta_t"]

    for n in range(maturity_step - 1, stop_step - 1, -1):
        r_n = tree["r_tree"][n]
        p_up_n = tree["p_up_tree"][n]
        current = np.zeros(n + 1)

        for j in range(n + 1):
            expected_next = (
                (1.0 - p_up_n[j]) * values[j]
                + p_up_n[j] * values[j + 1]
            )
            current[j] = np.exp(-r_n[j] * delta_t) * expected_next

        values = current

    return values

def price_zcb_binomial(tree, maturity_step):
```

```
    terminal_values = np.ones(maturity_step + 1)
    return backward_binomial(tree, maturity_step, terminal_values,
stop_step=0)[0]


def bond_values_at_option_maturity_binomial(tree, option_step,
bond_maturity_step):
    terminal_values = np.ones(bond_maturity_step + 1)
    return backward_binomial(tree, bond_maturity_step, terminal_values,
stop_step=option_step)


def price_bond_option_binomial(tree, option_step, bond_maturity_step, strike):
    bond_values = bond_values_at_option_maturity_binomial(tree, option_step,
bond_maturity_step)
    payoff = np.maximum(bond_values - strike, 0.0)
    return backward_binomial(tree, option_step, payoff, stop_step=0)[0]


def interior_probabilities(q):
    p_up = 1.0 / 6.0 + 0.5 * q + 0.5 * q**2
    p_mid = 2.0 / 3.0 - q**2
    p_down = 1.0 / 6.0 - 0.5 * q + 0.5 * q**2
    return p_up, p_mid, p_down


def upper_boundary_probabilities(q):
    p0 = 7.0 / 6.0 + 1.5 * q + 0.5 * q**2
    p1 = -1.0 / 3.0 - 2.0 * q - q**2
    p2 = 1.0 / 6.0 + 0.5 * q + 0.5 * q**2
    return p0, p1, p2


def lower_boundary_probabilities(q):
    p0 = 7.0 / 6.0 - 1.5 * q + 0.5 * q**2
    p1 = -1.0 / 3.0 + 2.0 * q - q**2
    p2 = 1.0 / 6.0 - 0.5 * q + 0.5 * q**2
    return p0, p1, p2


def build_truncated_trinomial_tree(P_market, delta_t, mean_reversion, sigma):
```

```
N = len(P_market)
delta_x = sigma * np.sqrt(3.0 * delta_t)
J = int(np.floor(np.sqrt(2.0 / 3.0) / (mean_reversion * delta_t)))

if J < 2:
    raise ValueError("J is too small. Change mean_reversion or delta_t.")

Q_tree = [np.array([1.0])]
a_vec = []
r_tree = []
j_tree = []

for n in range(N):
    m_n = min(n, J)
    j_vals = np.arange(-m_n, m_n + 1)
    x_n = j_vals * delta_x
    Q_n = Q_tree[n]

    denom = np.sum(Q_n * np.exp(-x_n * delta_t))
    a_n = -(1.0 / delta_t) * np.log(P_market[n] / denom)
    r_n = a_n + x_n

    a_vec.append(a_n)
    r_tree.append(r_n)
    j_tree.append(j_vals)

    m_next = min(n + 1, J)
    j_next = np.arange(-m_next, m_next + 1)
    Q_next = np.zeros(len(j_next))

    discounted_mass = Q_n * np.exp(-r_n * delta_t)

    for i, j in enumerate(j_vals):
```

```
        x_current = j * delta_x
        q = (-mean_reversion * x_current) * delta_t / delta_x

        if -J < j < J:
            p_up, p_mid, p_down = interior_probabilities(q)

            Q_next[(j + 1) - j_next[0]] += p_up * discounted_mass[i]
            Q_next[j - j_next[0]] += p_mid * discounted_mass[i]
            Q_next[(j - 1) - j_next[0]] += p_down * discounted_mass[i]

        elif j == J:
            p0, p1, p2 = upper_boundary_probabilities(q)

            Q_next[j - j_next[0]] += p0 * discounted_mass[i]
            Q_next[(j - 1) - j_next[0]] += p1 * discounted_mass[i]
            Q_next[(j - 2) - j_next[0]] += p2 * discounted_mass[i]

        elif j == -J:
            p0, p1, p2 = lower_boundary_probabilities(q)

            Q_next[j - j_next[0]] += p0 * discounted_mass[i]
            Q_next[(j + 1) - j_next[0]] += p1 * discounted_mass[i]
            Q_next[(j + 2) - j_next[0]] += p2 * discounted_mass[i]

    Q_tree.append(Q_next)

return {
    "delta_t": delta_t,
    "mean_reversion": mean_reversion,
    "sigma": sigma,
    "delta_x": delta_x,
    "J": J,
    "a_vec": np.array(a_vec),
```

```
    "r_tree": r_tree,
    "j_tree": j_tree,
  }

def backward_trinomial(tree, maturity_step, terminal_values, stop_step=0):
  values = terminal_values.copy()
  delta_t = tree["delta_t"]
  mean_reversion = tree["mean_reversion"]
  delta_x = tree["delta_x"]
  J = tree["J"]

  for n in range(maturity_step - 1, stop_step - 1, -1):
    j_vals = tree["j_tree"][n]
    j_next = np.arange(-min(n + 1, J), min(n + 1, J) + 1)
    r_n = tree["r_tree"][n]
    current = np.zeros(len(j_vals))

    for i, j in enumerate(j_vals):
      x_current = j * delta_x
      q = (-mean_reversion * x_current) * delta_t / delta_x

      if -J < j < J:
        p_up, p_mid, p_down = interior_probabilities(q)

        expected_next = (
          p_up * values[(j + 1) - j_next[0]]
          + p_mid * values[j - j_next[0]]
          + p_down * values[(j - 1) - j_next[0]]
        )

      elif j == J:
        p0, p1, p2 = upper_boundary_probabilities(q)
```

```
            expected_next = (
                p0 * values[j - j_next[0]]
                + p1 * values[(j - 1) - j_next[0]]
                + p2 * values[(j - 2) - j_next[0]]
            )

        elif j == -J:
            p0, p1, p2 = lower_boundary_probabilities(q)

            expected_next = (
                p0 * values[j - j_next[0]]
                + p1 * values[(j + 1) - j_next[0]]
                + p2 * values[(j + 2) - j_next[0]]
            )

        current[i] = np.exp(-r_n[i] * delta_t) * expected_next

    values = current

  return values


def price_zcb_trinomial(tree, maturity_step):
  m = min(maturity_step, tree["J"])
  terminal_values = np.ones(2 * m + 1)
  return backward_trinomial(tree, maturity_step, terminal_values,
stop_step=0)[0]


def bond_values_at_option_maturity_trinomial(tree, option_step,
bond_maturity_step):
  m = min(bond_maturity_step, tree["J"])
  terminal_values = np.ones(2 * m + 1)
  return backward_trinomial(tree, bond_maturity_step, terminal_values,
stop_step=option_step)
```

```
def price_bond_option_trinomial(tree, option_step, bond_maturity_step, strike):
    bond_values = bond_values_at_option_maturity_trinomial(tree, option_step,
bond_maturity_step)
    payoff = np.maximum(bond_values - strike, 0.0)
    return backward_trinomial(tree, option_step, payoff, stop_step=0)[0]


binomial_tree = build_binomial_tree(P_market, delta_t, sigma)
trinomial_tree = build_truncated_trinomial_tree(P_market, delta_t,
mean_reversion, sigma)


validation_maturities = [0.25, 0.50, 1.0, 2.0, 3.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0]
validation_rows = []


for T_mat in validation_maturities:
    m_step = int(round(T_mat / delta_t))
    market_price = P_market[m_step - 1]


    price_bin = price_zcb_binomial(binomial_tree, m_step)
    price_tri = price_zcb_trinomial(trinomial_tree, m_step)


    validation_rows.append({
        "bond_maturity": T_mat,
        "P_market": market_price,
        "P_binomial": price_bin,
        "P_trinomial": price_tri,
        "abs_error_binomial": abs(market_price - price_bin),
        "abs_error_trinomial": abs(market_price - price_tri),
    })


validation_table = pd.DataFrame(validation_rows)


contracts = [
    (1.0, 5.0),
    (2.0, 10.0),
```

```
    (5.0, 20.0),
]

option_rows = []

for option_maturity, bond_maturity in contracts:
    s_step = int(round(option_maturity / delta_t))
    m_step = int(round(bond_maturity / delta_t))

    if m_step <= s_step:
        raise ValueError("Bond maturity must be strictly greater than option
maturity.")

    P0_s = P_market[s_step - 1]
    P0_m = P_market[m_step - 1]
    strike_atm_forward = P0_m / P0_s

    price_bin = price_bond_option_binomial(
        binomial_tree, s_step, m_step, strike_atm_forward
    )
    price_tri = price_bond_option_trinomial(
        trinomial_tree, s_step, m_step, strike_atm_forward
    )

    option_rows.append({
        "option_maturity": option_maturity,
        "bond_maturity": bond_maturity,
        "strike": strike_atm_forward,
        "price_binomial": price_bin,
        "price_trinomial": price_tri,
        "abs_difference": abs(price_bin - price_tri),
    })

option_table = pd.DataFrame(option_rows)
```

```
print("\nBond-pricing validation table:")
print(validation_table)

print("\nMaximum binomial bond-pricing error:")
print(validation_table["abs_error_binomial"].max())

print("\nMaximum trinomial bond-pricing error:")
print(validation_table["abs_error_trinomial"].max())

print("\nBaseline bond-option pricing table:")
print(option_table.to_string(index=False))



validation_file = results_path / "bond_pricing_validation.csv"
option_file = results_path / "bond_option_pricing_baseline.csv"

validation_table.to_csv(validation_file, index=False)
option_table.to_csv(option_file, index=False)

print(f"\nSaved bond validation table to: {validation_file}")
print(f"Saved baseline option pricing table to: {option_file}")
```

Sensitivity Analysis Calculation

```
lib import Path

pd.set_option("display.max_columns", None)
pd.set_option("display.width", 220)
pd.set_option("display.max_colwidth", None)
```

```
curve_obs = pd.read_csv(input_file)


observed_times = curve_obs["maturity_years"].to_numpy()
observed_rates = curve_obs["spot_rate_decimal"].to_numpy()


T = 30.0
baseline_delta_t = 0.25
baseline_mean_reversion = 0.05
baseline_sigma = 0.01


contracts = [
    (1.0, 5.0),
    (2.0, 10.0),
    (5.0, 20.0),
]

delta_t_values = [0.50, 0.25, 0.125]
mean_reversion_values = [0.02, 0.05, 0.10]
sigma_values = [0.005, 0.01, 0.015]
strike_multipliers = [0.95, 1.00, 1.05]


def interpolate_market_curve(T, delta_t, observed_times, observed_rates):
    N = int(round(T / delta_t))
    t_grid = np.arange(1, N + 1) * delta_t
    interp_rates = np.interp(t_grid, observed_times, observed_rates)
    P_market = np.exp(-interp_rates * t_grid)
    return t_grid, interp_rates, P_market
```

```
def build_binomial_tree(P_market, delta_t, sigma):
    N = len(P_market)
    delta_x = sigma * np.sqrt(delta_t)

    Q_tree = [np.array([1.0])]
    a_vec = []
    r_tree = []
    p_up_tree = []

    for n in range(N):
        j_idx = np.arange(n + 1)
        x_n = (2 * j_idx - n) * delta_x
        Q_n = Q_tree[n]

        denom = np.sum(Q_n * np.exp(-x_n * delta_t))
        a_n = -(1.0 / delta_t) * np.log(P_market[n] / denom)

        r_n = a_n + x_n
        p_up_n = np.full(n + 1, 0.5)

        a_vec.append(a_n)
        r_tree.append(r_n)
        p_up_tree.append(p_up_n)

        discounted_mass = Q_n * np.exp(-r_n * delta_t)
        Q_next = np.zeros(n + 2)

        for j in range(n + 1):
            Q_next[j] += (1.0 - p_up_n[j]) * discounted_mass[j]
            Q_next[j + 1] += p_up_n[j] * discounted_mass[j]

        Q_tree.append(Q_next)
```

```
    return {
        "delta_t": delta_t,
        "sigma": sigma,
        "delta_x": delta_x,
        "a_vec": np.array(a_vec),
        "r_tree": r_tree,
        "p_up_tree": p_up_tree,
    }

def backward_binomial(tree, maturity_step, terminal_values, stop_step=0):
    values = terminal_values.copy()
    delta_t = tree["delta_t"]

    for n in range(maturity_step - 1, stop_step - 1, -1):
        r_n = tree["r_tree"][n]
        p_up_n = tree["p_up_tree"][n]
        current = np.zeros(n + 1)

        for j in range(n + 1):
            expected_next = (
                (1.0 - p_up_n[j]) * values[j]
                + p_up_n[j] * values[j + 1]
            )
            current[j] = np.exp(-r_n[j] * delta_t) * expected_next

        values = current

    return values

def bond_values_at_option_maturity_binomial(tree, option_step,
bond_maturity_step):
    terminal_values = np.ones(bond_maturity_step + 1)
    return backward_binomial(tree, bond_maturity_step, terminal_values,
stop_step=option_step)
```

```
def price_bond_option_binomial(tree, option_step, bond_maturity_step, strike):
    bond_values = bond_values_at_option_maturity_binomial(tree, option_step,
bond_maturity_step)
    payoff = np.maximum(bond_values - strike, 0.0)
    return backward_binomial(tree, option_step, payoff, stop_step=0)[0]



def interior_probabilities(q):
    p_up = 1.0 / 6.0 + 0.5 * q + 0.5 * q**2
    p_mid = 2.0 / 3.0 - q**2
    p_down = 1.0 / 6.0 - 0.5 * q + 0.5 * q**2
    return p_up, p_mid, p_down

def upper_boundary_probabilities(q):
    p0 = 7.0 / 6.0 + 1.5 * q + 0.5 * q**2
    p1 = -1.0 / 3.0 - 2.0 * q - q**2
    p2 = 1.0 / 6.0 + 0.5 * q + 0.5 * q**2
    return p0, p1, p2

def lower_boundary_probabilities(q):
    p0 = 7.0 / 6.0 - 1.5 * q + 0.5 * q**2
    p1 = -1.0 / 3.0 + 2.0 * q - q**2
    p2 = 1.0 / 6.0 - 0.5 * q + 0.5 * q**2
    return p0, p1, p2

def build_truncated_trinomial_tree(P_market, delta_t, mean_reversion, sigma):
    N = len(P_market)
    delta_x = sigma * np.sqrt(3.0 * delta_t)
    J = int(np.floor(np.sqrt(2.0 / 3.0) / (mean_reversion * delta_t)))

    if J < 2:
        raise ValueError("J is too small. Change mean_reversion or delta_t.")
```

```
Q_tree = [np.array([1.0])]
a_vec = []
r_tree = []
j_tree = []
all_probabilities = []

for n in range(N):
    m_n = min(n, J)
    j_vals = np.arange(-m_n, m_n + 1)
    x_n = j_vals * delta_x
    Q_n = Q_tree[n]

    denom = np.sum(Q_n * np.exp(-x_n * delta_t))
    a_n = -(1.0 / delta_t) * np.log(P_market[n] / denom)
    r_n = a_n + x_n

    a_vec.append(a_n)
    r_tree.append(r_n)
    j_tree.append(j_vals)

    m_next = min(n + 1, J)
    j_next = np.arange(-m_next, m_next + 1)
    Q_next = np.zeros(len(j_next))

    discounted_mass = Q_n * np.exp(-r_n * delta_t)

    for i, j in enumerate(j_vals):
        x_current = j * delta_x
        q = (-mean_reversion * x_current) * delta_t / delta_x

        if -J < j < J:
            p_up, p_mid, p_down = interior_probabilities(q)
            all_probabilities.extend([p_up, p_mid, p_down])
```

```
            Q_next[(j + 1) - j_next[0]] += p_up * discounted_mass[i]
            Q_next[j - j_next[0]] += p_mid * discounted_mass[i]
            Q_next[(j - 1) - j_next[0]] += p_down * discounted_mass[i]

        elif j == J:
            p0, p1, p2 = upper_boundary_probabilities(q)
            all_probabilities.extend([p0, p1, p2])

            Q_next[j - j_next[0]] += p0 * discounted_mass[i]
            Q_next[(j - 1) - j_next[0]] += p1 * discounted_mass[i]
            Q_next[(j - 2) - j_next[0]] += p2 * discounted_mass[i]

        elif j == -J:
            p0, p1, p2 = lower_boundary_probabilities(q)
            all_probabilities.extend([p0, p1, p2])

            Q_next[j - j_next[0]] += p0 * discounted_mass[i]
            Q_next[(j + 1) - j_next[0]] += p1 * discounted_mass[i]
            Q_next[(j + 2) - j_next[0]] += p2 * discounted_mass[i]

    Q_tree.append(Q_next)

all_probabilities = np.array(all_probabilities)

return {
    "delta_t": delta_t,
    "mean_reversion": mean_reversion,
    "sigma": sigma,
    "delta_x": delta_x,
    "J": J,
    "a_vec": np.array(a_vec),
    "r_tree": r_tree,
```

```
        "j_tree": j_tree,
        "probability_min": float(all_probabilities.min()),
        "probability_max": float(all_probabilities.max()),
        "number_of_violations": int(np.sum((all_probabilities < 0) |
(all_probabilities > 1))),
    }

def backward_trinomial(tree, maturity_step, terminal_values, stop_step=0):
    values = terminal_values.copy()
    delta_t = tree["delta_t"]
    mean_reversion = tree["mean_reversion"]
    delta_x = tree["delta_x"]
    J = tree["J"]

    for n in range(maturity_step - 1, stop_step - 1, -1):
        j_vals = tree["j_tree"][n]
        j_next = np.arange(-min(n + 1, J), min(n + 1, J) + 1)
        r_n = tree["r_tree"][n]
        current = np.zeros(len(j_vals))

        for i, j in enumerate(j_vals):
            x_current = j * delta_x
            q = (-mean_reversion * x_current) * delta_t / delta_x

            if -J < j < J:
                p_up, p_mid, p_down = interior_probabilities(q)

                expected_next = (
                    p_up * values[(j + 1) - j_next[0]]
                    + p_mid * values[j - j_next[0]]
                    + p_down * values[(j - 1) - j_next[0]]
                )

            elif j == J:
```

```
            p0, p1, p2 = upper_boundary_probabilities(q)

            expected_next = (
                p0 * values[j - j_next[0]]
                + p1 * values[(j - 1) - j_next[0]]
                + p2 * values[(j - 2) - j_next[0]]
            )

        elif j == -J:
            p0, p1, p2 = lower_boundary_probabilities(q)

            expected_next = (
                p0 * values[j - j_next[0]]
                + p1 * values[(j + 1) - j_next[0]]
                + p2 * values[(j + 2) - j_next[0]]
            )

        current[i] = np.exp(-r_n[i] * delta_t) * expected_next

    values = current

  return values


def bond_values_at_option_maturity_trinomial(tree, option_step,
bond_maturity_step):
  m = min(bond_maturity_step, tree["J"])
  terminal_values = np.ones(2 * m + 1)
  return backward_trinomial(tree, bond_maturity_step, terminal_values,
stop_step=option_step)


def price_bond_option_trinomial(tree, option_step, bond_maturity_step, strike):
  bond_values = bond_values_at_option_maturity_trinomial(tree, option_step,
bond_maturity_step)
  payoff = np.maximum(bond_values - strike, 0.0)
```

```
    return backward_trinomial(tree, option_step, payoff, stop_step=0)[0]


#
def price_contracts_for_scenario(T, delta_t, mean_reversion, sigma, contracts,
strike_multiplier=1.0):
    t_grid, _, P_market = interpolate_market_curve(T, delta_t, observed_times,
observed_rates)


    binomial_tree = build_binomial_tree(P_market, delta_t, sigma)
    trinomial_tree = build_truncated_trinomial_tree(P_market, delta_t,
mean_reversion, sigma)


    rows = []


    for option_maturity, bond_maturity in contracts:
        s_step = int(round(option_maturity / delta_t))
        m_step = int(round(bond_maturity / delta_t))


        if not np.isclose(s_step * delta_t, option_maturity):
            raise ValueError(f"Option maturity {option_maturity} is not aligned with
delta_t = {delta_t}")


        if not np.isclose(m_step * delta_t, bond_maturity):
            raise ValueError(f"Bond maturity {bond_maturity} is not aligned with
delta_t = {delta_t}")


        if m_step <= s_step:
            raise ValueError("Bond maturity must be strictly greater than option
maturity.")


        P0_s = P_market[s_step - 1]
        P0_m = P_market[m_step - 1]
        K_atm = P0_m / P0_s
        strike = strike_multiplier * K_atm
```

```
        price_binomial = price_bond_option_binomial(binomial_tree, s_step, m_step,
strike)
        price_trinomial = price_bond_option_trinomial(trinomial_tree, s_step,
m_step, strike)


        rows.append({
            "option_maturity": option_maturity,
            "bond_maturity": bond_maturity,
            "delta_t": delta_t,
            "N": int(round(T / delta_t)),
            "mean_reversion": mean_reversion,
            "sigma": sigma,
            "strike_multiplier": strike_multiplier,
            "strike": strike,
            "price_binomial": price_binomial,
            "price_trinomial": price_trinomial,
            "abs_difference": abs(price_binomial - price_trinomial),
            "J": trinomial_tree["J"],
            "probability_min": trinomial_tree["probability_min"],
            "probability_max": trinomial_tree["probability_max"],
            "number_of_violations": trinomial_tree["number_of_violations"],
        })

    return pd.DataFrame(rows)



delta_t_tables = []

for dt in delta_t_values:
    table_dt = price_contracts_for_scenario(
        T=T,
        delta_t=dt,
        mean_reversion=baseline_mean_reversion,
        sigma=baseline_sigma,
```

```
        contracts=contracts,
        strike_multiplier=1.0,
    )
    delta_t_tables.append(table_dt)

delta_t_sensitivity = pd.concat(delta_t_tables, ignore_index=True)



mean_reversion_tables = []

for a_val in mean_reversion_values:
    table_a = price_contracts_for_scenario(
        T=T,
        delta_t=baseline_delta_t,
        mean_reversion=a_val,
        sigma=baseline_sigma,
        contracts=contracts,
        strike_multiplier=1.0,
    )
    mean_reversion_tables.append(table_a)

mean_reversion_sensitivity = pd.concat(mean_reversion_tables,
ignore_index=True)



sigma_tables = []

for sigma_val in sigma_values:
    table_sigma = price_contracts_for_scenario(
        T=T,
        delta_t=baseline_delta_t,
        mean_reversion=baseline_mean_reversion,
        sigma=sigma_val,
        contracts=contracts,
```

```
        strike_multiplier=1.0,
    )
    sigma_tables.append(table_sigma)

sigma_sensitivity = pd.concat(sigma_tables, ignore_index=True)



strike_tables = []

for k_mult in strike_multipliers:
    table_k = price_contracts_for_scenario(
        T=T,
        delta_t=baseline_delta_t,
        mean_reversion=baseline_mean_reversion,
        sigma=baseline_sigma,
        contracts=contracts,
        strike_multiplier=k_mult,
    )
    strike_tables.append(table_k)

strike_sensitivity = pd.concat(strike_tables, ignore_index=True)



print("\nDelta t sensitivity:")
print(delta_t_sensitivity.to_string(index=False))

print("\nMean reversion sensitivity:")
print(mean_reversion_sensitivity.to_string(index=False))

print("\nSigma sensitivity:")
print(sigma_sensitivity.to_string(index=False))

print("\nStrike sensitivity:")
```

```
print(strike_sensitivity.to_string(index=False))
```