Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 189 (2007) 1099–1123
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

A convergent quadratic-time lattice algorithm for pricing European-style Asian options ☆

William Wei-Yuan Hsu [a,*], Yuh-Dauh Lyuu [a,b]

[a] *Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Section 4, Roosevelt Road, Taipei 106, Taiwan*
[b] *Department of Finance, National Taiwan University, No. 1, Section 4, Roosevelt Road, Taipei 106, Taiwan*

Abstract

Asian options are strongly path-dependent derivatives. Although efficient numerical methods and approximate closed-form formulas are available, most lack convergence guarantees. Asian options can also be priced on the lattice. All efficient lattice algorithms keep only a polynomial number of states and use interpolation to compensate for the less than full representation of the states. Let the time to maturity be partitioned into n periods. This paper presents the first $O(n^2)$-time convergent lattice algorithm for pricing European-style Asian options; it is the most efficient lattice algorithm with convergence guarantees. The algorithm relies on the Lagrange multipliers to choose optimally the number of states for each node of the lattice. The algorithm is also memory efficient. Extensive numerical experiments and comparison with existing PDE, analytical, and lattice methods confirm the performance claims and the competitiveness of our algorithm. This result places the problem of European-style Asian option pricing in the same complexity class as that of the vanilla option on the lattice.

Keywords: Option pricing; Binomial model; Trinomial model; Path-dependent derivative; Asian option; Complexity; Lagrange multiplier; PDE; Lattice

1. Introduction

Path-dependent derivatives have payoffs that depend strongly on the price history of the underlying asset. In pricing such derivatives, the historical information needs to be encoded as part of the state. Although the effect tends to enlarge the state space, it may not lead to exponential complexity if done properly. For example, barrier and look back options can be efficiently priced despite the fact that they are path-dependent. For other

☆ This research was supported by the National Science Council of Taiwan under NSC Grant 93-2213-E-002-002. An early version of this paper was presented at the *IASTED International Conference on Financial Engineering and Applications* (FEA 2004) in Cambridge, MA, USA, in November 2004.
* Corresponding author.
E-mail addresses: r7526001@csie.ntu.edu.tw (W.W.-Y. Hsu), lyuu@csie.ntu.edu.tw (Y.-D. Lyuu).

0096-3003/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.11.180

path-dependent derivatives, however, the performance issue is more intricate. The Asian option is perhaps the most representative of them.

Asian options are originally traded in the Asian markets, particularly Tokyo [1]. The payoff of the Asian option depends on the arithmetic average price of the underlying asset. It is therefore useful for hedging transactions whose cost is related to the average price of the underlying asset. The price of the Asian option is also less subject to price manipulation. Hence the averaging feature is popular in many thinly traded markets and embedded in complex derivatives such as the refix clauses in convertible bonds. Asian option is first suggested by [2].

Pricing Asian options has been a practical and research problem of long standing when the underlying asset's price is lognormally distributed. The source of the difficulty lies in that the sum of lognormal random variables is no longer lognormally distributed. Several solutions have been proposed for the problem, including analytical approximations, Monte Carlo simulation, lattices, and partial differential equations (PDEs).

Approximate closed-form formulas have been derived under various assumptions. These formulas have been evaluated thoroughly in [1,3–5]. The general conclusion is that all are as good as their assumptions, and most lack convergence guarantees. For example, some formulas lose accuracy for low volatility levels, whereas others do so for high volatility levels. Lower and upper bounds on the option price are derived in [6–9]. As no simple, exact closed-form solutions exist yet, the development of efficient numerical algorithms becomes an important alternative. First, there are the popular Monte Carlo and the related quasi-Monte Carlo methods surveyed in [10]. Both the Monte Carlo approach and the analytical approach suffer from the inability to handle early exercise without bias. Although Longstaff and Schwartz have developed a least-squares Monte Carlo approach to tackle the problem [11], a convergence proof remains elusive (see [12] for a convergence proof in the case of American-style vanilla options). Other drawbacks of Monte Carlo include its probabilistic nature and relative inefficiency.

The third type of approach, the lattice and the closely related PDE methods, are more general as they can handle early exercise. The main challenge with the lattice method in the case of Asian options is its exponential nature: An exponential number of arithmetic operations seem needed for an exact evaluation. This is because every price path, which corresponds to a state (that is, the average to date, also called the running average), leads to a different average price, thus payoff as well. To reduce the complexity, all known practical lattice algorithms keep only a small subset of the states. When an option value for a missing state is called for in the pricing algorithms, it is interpolated from the option values of the neighboring states. This successful paradigm is due to Hull and White [13] and Ritchken et al. [14] and is followed by, for example, Zvan et al. [15] and Klassen [16]. We will call it the interpolation paradigm. The interpolation paradigm obviously introduces interpolation error, and a major concern is whether the magnitude of the interpolation error converges to zero. Pricing Asian options with two-dimensional PDEs also tackles the issue of exploding state space with the interpolation paradigm [17].

Partition the time to maturity into n periods. It is well-known that a binomial lattice with n periods contains about $n^{2/2}$ nodes (see Fig. 1). Observe that the binomial model has a lattice structure because it recombines. Let the average number of states (running averages) kept at each node be k, a critical adjustable parameter for lattice algorithms. As the total number of states is $kn^2/2$, the asymptotic running time of the lattice algorithm is $O(kn^2)$. An algorithm must decide upon how to distribute these $kn^2/2$ states among the nodes. The choice

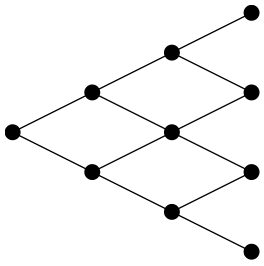


Fig. 1. Binomial lattice. Each node has two successor nodes. The number of nodes at any time i is $i+1$. The total number of nodes of an n-period binomial lattice is $\sum_{i=0}^{n}(i+1) = (n+2)(n+1)/2 \approx n^2/2$.

will determine whether a small k guarantees convergence to the true value. The ultimate goal is that a constant k – thus quadratic running time – suffices.

A uniform allocation scheme is perhaps the most straightforward scheme. It simply allocates the same number of states k for each node. Unfortunately, a uniform allocation scheme with k states per node is suboptimal. First, a small k may result in large deviations from the correct option price because of the low resolution, but a k that grows in some way with n makes the algorithm less efficient. Hence we are confronted with the conflicting demands of accuracy and speed. Second, an identical k for all nodes may be sufficient for nodes where few paths lead to, but hardly enough for nodes where exponentially many paths lead to. Intuitively, an algorithm should assign a smaller number of states to a node with a low probability of occurrence than one with a high probability of occurrence. The problem is how to find the numbers methodically. This paper addresses these issues by an optimization technique, which yields nonuniform allocation schemes. In particular, the number of states allocated for each node is determined by the Lagrange multiplier. It is then proved that a constant k suffices for convergence purposes, resulting in a running time of $O(n^2)$.

Some algorithms in the literature let the running averages or the logarithms of the running averages kept at each node be spaced at a given distance apart. (We will call the latter, more popular allocation scheme the log-linear allocation scheme.) As the range of running averages differs for different nodes, the number of states vary from node to node in both cases. Although the result is not a uniform allocation scheme, the overall idea remains ad hoc.

Rigorous convergence analysis of Asian options is relatively rare. Barraquand and Pudet [18] and Forsyth et al. [17] analyze the interpolation paradigm for European-style Asian options. For example, under plausible assumptions, Forsyth et al. demonstrate that the interpolation paradigm together with linear interpolation and the log-linear allocation scheme is convergent provided that k is proportional to $n^{1.5}$ [17]; the running time is thus $O(n^{3.5})$. Dai and Lyuu achieve the same running time with the nonuniform allocation scheme but without the assumptions [19]. In general, algorithms based on linear interpolation lack convergence guarantees unless k grows with n. This rules out the possibility of quadratic-time algorithms. Traditional PDE methods solve a two-dimensional PDE [20]. The most efficient one, due to Forsyth et al. runs in $O(n^3)$ time [17]. Since Rogers and Shi derive the first one-dimensional PDE for Asian options [6], subsequent works by, for example, Večeř [21] and Dubois and Lelièvre [22] have proposed one-dimensional PDE methods that run in $O(n^2)$ time. Numerical evaluation in the paper will show that one-dimensional PDEs require more mesh points for high volatility levels or long maturities. Another drawback of one-dimensional PDE methods is they cannot apply to American-style fixed-strike Asian options. The transform method is yet another numerical alternative. It includes the $O(n^2 \log n)$-time algorithm of Benhamou based on the fast Fourier transform for pricing discretely sampled Asian options [23] (discretely sampled Asian options monitor the price of the underlying asset at fixed time points), and Fusai based on both Fourier and Laplace transforms for pricing continuously sampled Asian options [5] (continuously sampled Asian option monitors the price of the underlying asset continuously).

Lattice algorithms that do not use approximations beyond discretization of the continuous-time model are called exact. Exact algorithms are convergent, but they traditionally require exponential time until the multiresolution algorithm of Dai and Lyuu [19]. The multiresolution algorithm maintains all possible states and is hence exact. Empirically, it runs in subexponential time. Later, Dai and Lyuu prove that a variant of the multiresolution algorithm does indeed run in $2^{O(\sqrt{n})}$ time [24]. It is the first exact lattice algorithm to break the exponential-time barrier, but its running time remains prohibitive.

This paper proposes the first $O(n^2)$-time lattice algorithm for pricing European-style fixed-strike Asian options. Under assumptions similar to those made by Forsyth et al. [17], the algorithm is guaranteed to converge to the true value of the continuous-time model. It is hence the first convergent lattice algorithm competitive with the one-dimensional PDE method in terms of efficiency. Interestingly, our algorithm's convergence rate does not go down with high volatilities or long maturities; in fact, the opposite is true. Extensive numerical experiments and comparison with existing methods back up the claims.

Our algorithm draws on four ideas. First is the methodology of Dai et al. [19,25], which establishes the advantages of nonuniform allocation schemes based on optimization principles. Second, Forsyth et al. make explicit some of the plausible assumptions adopted in our analysis [17]. These assumptions combined with the nonuniform allocation scheme allow detailed analysis of individual nodes' contribution to the total error.

Third, we only need to work with running averages below a threshold at each node. The reason is that for European-style Asian options, running averages at or over this threshold will necessarily result in the option being in the money at maturity, in which case their contribution to the option value can be given by a simple formula. Accordingly, the algorithm devotes resources only to running averages lower than the said threshold. This idea is from Aingworth et al. [26]. Finally, the algorithm adopts the 4-point polynomial interpolation instead of the common 1-point (nearest), 2-point (linear), or 3-point (quadratic) interpolation scheme. The advantages of higher-order interpolation have been documented before by, for instance, Hull and White [13], Boyle and Tian [27], Wei [28], and Zvan et al. [15].

This paper focuses on European-style fixed-strike continuously sampled Asian options. But our lattice algorithm can be modified to price discretely sampled Asian options [29]. In reality, Asian options are discretely sampled [30], but the continuously sampled version is much more intensively studied in the literature. Our algorithm can also be used to price floating-strike Asian options by a result of Henderson and Wojakowski [31]. Although this paper focuses on binomial lattices, its results can be carried over to trinomial lattices.

This paper is organized as follows. Section 2 reviews basic terms and facts. Section 3 surveys the notion of interpolation and proves properties that will become pivotal in subsequent analysis. Section 4 describes the proposed algorithm. Section 5 analyzes the algorithm's error behavior theoretically and evaluates it numerically. Section 6 compares it to other popular numerical methods. Section 7 concludes. The two appendices contain the proofs for highly technical claims stated in the main text.

2. Terminology and basic facts

Let $r \geqslant 0$ denote the continuously compounded risk-free interest rate. The continuous-time stock price dynamics is the lognormal diffusion

$$\frac{\mathrm{d}S}{S} = r\,\mathrm{d}t + \sigma\,\mathrm{d}W_t$$

in a risk-neutral economy, where W_t is the Wiener process and σ is the volatility. The risk-neutral economy allows us to calculate the option price by taking expectation of the option's payoff function and then discounting it by the risk-free interest rate [32]. The lattice model is a discrete-time version of the above model. Let τ denote the time to maturity in years. For lattice algorithms, τ is partitioned into n discrete time steps. Each time (step) has a duration of $\Delta t \equiv \tau/n$ years. Let S_i stand for the stock price at time i (hence $i\Delta t$ years from now). In particular, S_0 is the known current price. The CRR binomial model of Cox, Ross, and Rubinstein approximates the lognormal diffusion as follows [33]. First, S_{i+1} equals $S_i u$ with probability p and $S_i d$ with probability $1-p$, where

$$u = \mathrm{e}^{\sigma\sqrt{\Delta t}},$$
$$d = \mathrm{e}^{-\sigma\sqrt{\Delta t}}.$$

Second, the probability p for the up move is set to

$$\frac{R-d}{u-d},$$

where $R \equiv \mathrm{e}^{r\Delta t}$ denotes the gross riskless return per period. Both $d \leqslant R \leqslant u$ and $0 \leqslant p \leqslant 1$ must hold to avoid arbitrage opportunities, which can be satisfied with $n > r^2\tau/\sigma^2$. Fig. 2a depicts a 2-period binomial model. We shall assume that the stock does not pay dividends for ease of presentation. We remark that our general conclusion does not depend on the particular binomial or even trinomial model chosen to approximate the lognormal diffusion.

The node at time i that results from j down moves and $i-j$ up moves will be denoted by $N(i,j)$. This node is associated with the stock price $S_0 u^{i-j} d^j$ at maturity. By the binomial model, the stock price can move from node $N(i,j)$ to node $N(i+1,j)$ with probability p and to node $N(i+1,j+1)$ with probability $1-p$. This is the lattice structure illustrated in Fig. 2b. The root node is of course $N(0,0)$. Let $B(i,j,p)$ denote the probability of getting j heads when tossing a coin i times with p being the probability of getting heads. Then

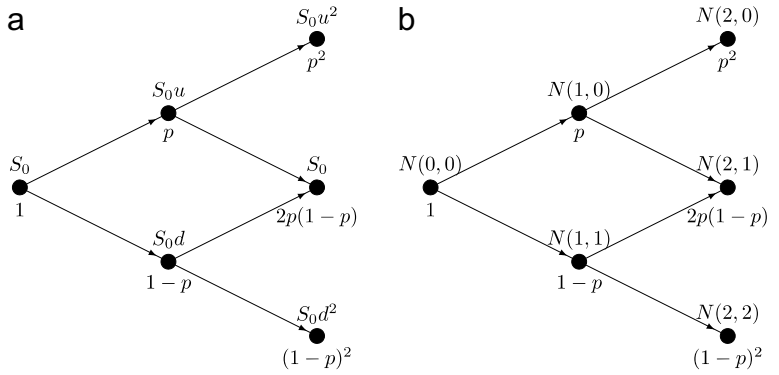


Fig. 2. A 2-period binomial model. (a) The probability of reaching each node is under the node. Note that the bulk of the probability is concentrated on the middle nodes. (b) The naming of the nodes.

$$B(i,j,p) \equiv \binom{i}{j} p^{i-j}(1-p)^j.$$

Node $N(i,j)$ can be reached from the root node with probability $B(i,j,p)$.

A path from the root node to a node at maturity contains $n+1$ prices $S_0, S_1, \ldots, S_n$. In the binomial model, there are 2^n such price paths as 2 outcomes are available for each of the n time steps. Let $X \geqslant 0$ be the strike price. The European-style Asian call has a payoff of

$$\left(\frac{1}{n+1}\sum_{i=0}^{n} S_i - X\right)^+$$

at maturity, where $(x)^+$ means $\max(x,0)$. Its arbitrage-free price is therefore

$$\frac{E\left[\left(\frac{1}{n+1}\sum_{i=0}^{n} S_i - X\right)^+\right]}{R^n}. \tag{1}$$

The European-style Asian put can be priced via the put-call parity in [34].

The option value can be evaluated by averaging the payoffs of all possible 2^n price paths $(S_0, S_1, \ldots, S_n)$. This value converges to the true value under the continuous-time lognormal diffusion model at a rate of $O(n^{-1})$ as n goes to infinity [32]. This brute-force pricing methodology results in the exponential-time algorithm alluded to in the introduction.

A price path $(S_0, S_1, \ldots, S_i)$, $0 \leqslant i \leqslant n$, has the price sum to date equal to

$$P_i \equiv S_0 + S_1 + \cdots + S_i,$$

which will be called the running sum. Its corresponding running average is $P_i/(i+1)$. In particular, $P_0 = S_0$. The tuple (i, S_i, P) captures the state in pricing the Asian option: The first element refers to the time, the second to the prevailing stock price, and the third to the running sum. Sometimes, we drop i and S_i from (i, S_i, P) and simply refer to the running sum or even the corresponding running average as the state when the context is unambiguous. The successor state $(i+1, S_{i+1}, Q)$ is related to the current state (i, S_i, P) via

$$Q = P + S_{i+1}. \tag{2}$$

For the binomial model, the state transition is:

$$(i, S_i, P) \begin{cases} \nearrow & (i+1, S_iu, P + S_iu) \quad \text{for an up move} \\ \searrow & (i+1, S_id, P + S_id) \quad \text{for a down move} \end{cases}$$

To each state (i, S_i, P) corresponds the option value $V(i, S_i, P)$. The backward-induction pricing formula stipulates that the current option value equals the discounted expected future option value

$$V(i, S_i, P) = \frac{pV(i+1, S_i u, P + S_i u) + (1-p)V(i+1, S_i d, P + S_i d)}{R}. \tag{3}$$

The sought-after option value is $V(0, S_0, S_0)$ at time 0. Formula (3) requires evaluating all possible states (that is, running sums); it yields the exponential-time brute-force algorithm.

Fortunately, for European-style Asian options, there is no need to evaluate the full range of states. A state (i, S_i, P) with a running sum $P > (n+1)X$ must end with an average price larger than X at maturity, thus in the money, because the average price at maturity equals

$$\frac{P + S_{i+1} + \cdots + S_n}{n+1} > \frac{(n+1)X + S_{i+1} + \cdots + S_n}{n+1} \geqslant X.$$

We next derive the option value corresponding to this state under the risk-neutral probability first given in [26]. If $R > 1$, the expected value of the average price at maturity, $E[P + S_{i+1} + \cdots + S_n]/(n+1)$, equals

$$\frac{P + S_i R + S_i R^2 + \cdots + S_i R^{n-i}}{n+1} = \left(P + S_i R \frac{1 - R^{n-i}}{1 - R}\right) \Big/ (n+1).$$

Because every path extending from (i, S_i, P) will end up in the money, the expected option payoff equals the above minus X. The case of $R = 1$ can be handled similarly. In summary, when $P > (n+1)X$,

$$V(i, S_i, P) = \begin{cases} [P + (n-i)S_i]/(n+1) - X & \text{if } R = 1, \\ R^{-(n-i)}\left\{[P + S_i R \frac{1-R^{n-i}}{1-R}]/(n+1) - X\right\} & \text{if } R > 1. \end{cases} \tag{4}$$

Two immediate consequences follow from formula (4). First, we only need to deal with states with a running sum in the range $[0, (n+1)X]$. This is because a state with a higher running sum can be given an option value by the formula. Second, high volatilities do not impair the accuracy of our algorithm, which is not the case with some other algorithms (see [4,35]). The reason is straightforward. A higher volatility makes it more likely for running sums to exceed $(n+1)X$, which can be priced exactly by the formula. In fact, without the upper limit of $(n+1)X$, lattice algorithms for European-style Asian options in general may have difficulty converging for large σ and/or τ unless k scales at least with n [36].

3. Polynomial Interpolation

Although formula (4) enhances the efficiency of a lattice algorithm, it alone does not render the backward-induction formula (3) practical because the number of states remains exponential. The next idea is to allocate much fewer states per node and to use interpolation to obtain $V(i, S, P)$ if the state (i, S, P) does not correspond to any state allocated by the algorithm.

We now review the notion of polynomial interpolation. Given m distinct points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ with $x_1 < x_2 < \cdots < x_m$, there exists a polynomial $p(x)$ of degree at most $m-1$ that goes through them. This polynomial is actually unique among the set of all polynomials of degree at most $m-1$. It can be expressed as

$$p(x) = \alpha_1(x)y_1 + \alpha_2(x)y_2 + \cdots + \alpha_m(x)y_m,$$

where the interpolation coefficients $\alpha_1(x), \alpha_2(x), \ldots, \alpha_m(x)$ are polynomials of degree at most $m-1$. (See [37] for more information.) The interpolation polynomial $p(x)$ can be derived either by Lagrange's interpolation formula or Newton's divided-difference formula. The former is easier to work with in proofs, whereas the latter is usually more efficient algorithmically. But as we will be working with $m = 4$, the difference is minor. We remark that nearest interpolation corresponds to $m = 1$, linear interpolation corresponds to $m = 2$, and quadratic interpolation corresponds to $m = 3$.

Suppose we are given four points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ with $x_1 < x_2 < x_3 < x_4$. In this paper, the x_i's are running averages, and the y_i's are the corresponding option values. Dai shows that the option price is convex with respect to the running average [36]; hence the four points are convex in shape. Furthermore, because the option value increases with the running average, $0 \leqslant y_1 \leqslant y_2 \leqslant y_3 \leqslant y_4$. In our algorithm, the x_i

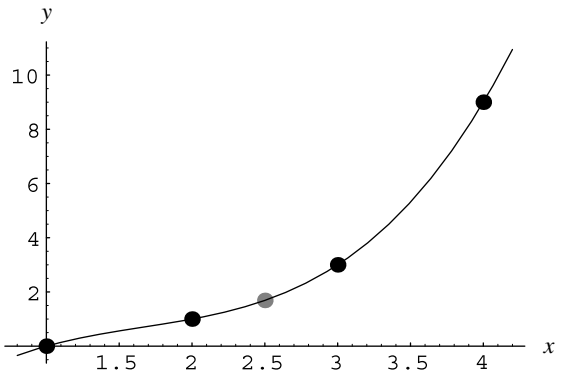


Fig. 3. Interpolation of the option value. The four data points are depicted as filled circles. They are increasing and convex. The interpolated option value at $x = 2.5$ is in gray.

are equally spaced (equidistant), which will be critical in the analysis. Let the interpolation polynomial be $p(x)$. The algorithm will be seeking $p(x)$ for $x \in [x_2, x_3]$.

It will be proved below that $p(x)$ is convex for $x \in [x_2, x_3]$. In Fig. 3, for example, the interpolation polynomial is convex for $2 \leqslant x \leqslant 3$ (but not for $1 \leqslant x \leqslant 2$). Therefore, the interpolated option value (in gray) must lie beneath any linear combination of the option values at $x = 2$ and $x = 3$. This property is consistent with the fact that the option price is convex with respect to the running average.

We next prove that $p(x)$ is indeed convex in the interval $[x_2, x_3]$. As the running averages are equidistant, we can without loss of generality let the four points be $(1, y_1), (2, y_2), (3, y_3), (4, y_4)$, where $0 \leqslant y_1 \leqslant y_2 \leqslant y_3 \leqslant y_4$ throughout the rest of the section. Given that the four points are convex,

$$y_1 + y_3 \geqslant 2y_2, \tag{5}$$

$$y_2 + y_4 \geqslant 2y_3. \tag{6}$$

The interpolation polynomial that passes through the four points is

$$p(x) = \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 y_3 + \alpha_4 y_4, \tag{7}$$

where the interpolation coefficients are

$$\alpha_1 \equiv \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} = \frac{-(x^3 - 9x^2 + 26x - 24)}{6},$$

$$\alpha_2 \equiv \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} = \frac{3(x^3 - 8x^2 + 19x - 12)}{6},$$

$$\alpha_3 \equiv \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} = \frac{-3(x^3 - 7x^2 + 14x - 8)}{6},$$

$$\alpha_4 \equiv \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)} = \frac{x^3 - 6x^2 + 11x - 6}{6}.$$

Next,

$$p'(x) = \frac{-y_1(3x^2 - 18x + 26) + 3y_2(3x^2 - 16x + 19) - 3y_3(3x^2 - 14x + 14) + y_4(3x^2 - 12x + 11)}{6}$$

$$= \frac{(-3y_1 + 9y_2 - 9y_3 + 3y_4)x^2 + (18y_1 - 48y_2 + 42y_3 - 12y_4)x - 26y_1 + 57y_2 - 42y_3 + 11y_4}{6},$$

and

$$p''(x) = \frac{2(-3y_1 + 9y_2 - 9y_3 + 3y_4)x + 18y_1 - 48y_2 + 42y_3 - 12y_4}{6}.$$

Observe that

$$p''(2) = \frac{6y_1 - 12y_2 + 6y_3}{6} \geqslant 0 \text{ by inequality (5)},$$
$$p''(3) = \frac{6y_2 - 12y_3 + 6y_4}{6} \geqslant 0 \text{ by inequality (6)}.$$

Since $p''(x)$ is a linear function in x, the above two inequalities imply $p''(x) \geqslant 0$ for $2 \leqslant x \leqslant 3$. Hence $p(x)$ is indeed convex in $2 \leqslant x \leqslant 3$.

A second key property is that the interpolation coefficients lie within $(-1, 1)$ for $x \in (2, 3)$. Indeed, it can be easily checked that

$$-0.06415 \leqslant \alpha_1 < 0,$$
$$0 < \alpha_2 < 1,$$
$$0 < \alpha_3 < 1,$$
$$-0.06415 \leqslant \alpha_4 < 0.$$

As

$$1 = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4,$$

$p(x)$ is a linear weighted average of $y_1,\ y_2,\ y_3$, and y_4 with a total weight of 1 and with the absolute value of each weight less than 1. As we will see later, this property ensures that our theoretical analysis can be built on that of Forsyth et al. in their analysis of the Hull–White algorithm that uses linear interpolation [17]. The Hull–White algorithm that uses quadratic interpolation, however, cannot follow the same analysis as the absolute values of the interpolation coefficients may exceed 1. The reason is that their running averages x_i are not equidistant.

4. Description of the algorithm

Each running sum P at a node $N(i, j)$ corresponds to the state $(i, S_0u^{i-j}d^j, P)$. Ideally we should allocate a state at each node $N(i, j)$ for each possible running sum. But the resulting exponential state count undermines this idea. Instead, the far fewer $k_{ij} + 1$ states will be allocated for node $N(i, j)$, leading to an approximation algorithm. The k_{ij} will be determined later. In view of formula (4), only states with running sums up to $(n + 1)X$ need be allocated. Thus we allocate $k_{ij} + 1$ states with running sums equally spaced between 0 and $(n + 1)X$. Two adjacent running sums therefore differ by $(n + 1)X/k_{ij}$. See Fig. 4 for illustration.

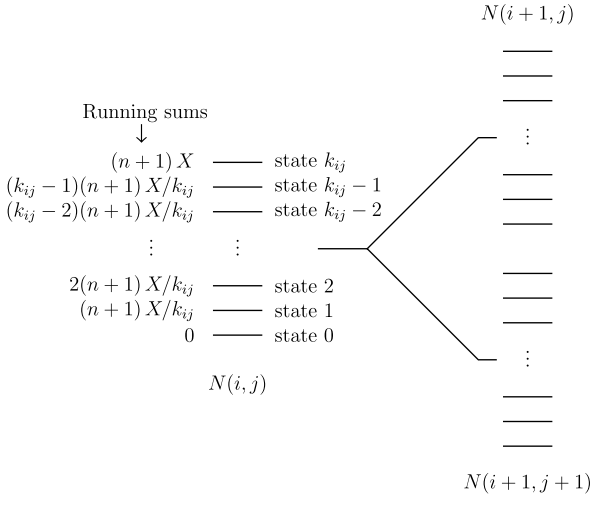


Fig. 4. Running sums and states. Each node $N(i, j)$ has $k_{ij} + 1$ states, starting from the running sum 0 and ending with the running sum $(n + 1)X$. The increments equal $(n + 1)X/k_{ij}$.

The number of allocated states $(k_{ij}+1)$ varies with the nodes. We need to determine the optimal numbers k_{ij} that minimize the total error given that only $kn^2/2$ states are available. For that purpose, we analyze every node's contribution to the total error. Once this is accomplished, an optimization technique is invoked to minimize that error. Earlier papers usually use the maximum interpolation error per time step as a node's contribution to the total error. But this worst-case analysis ignores that individual nodes' probability weights differ widely. The unequal weights to individual nodes' errors will be exploited by our scheme.

Appendix A applies the Lagrangian multipliers to determine k_{ij} as

$$k_{ij} = \frac{kn^2}{2} \frac{\left(\frac{B(i,j,p)}{i^4}\right)^{\frac{1}{5}}}{\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{B(s,t,p)}{s^4}\right)^{\frac{1}{5}}}. \tag{8}$$

To calculate the k_{ij} numerically, the bulk of the computation is spent on figuring out $B(i,j,p)$s, $0 \leqslant j \leqslant i \leqslant n$. They can be easily calculated by the following recurrence formula:

$$B(i,j,p) = \begin{cases} 1, & \text{if } i = j = 0, \\ pB(i-1,j,p), & \text{if } i \geqslant 1 \text{ and } j = 0, \\ (1-p)B(i-1,j-1,p), & \text{if } i = j \text{ and } i \geqslant 1, \\ pB(i-1,j,p) + (1-p)B(i-1,j-1,p), & \text{if } i > j \geqslant 1. \end{cases}$$

Hence it takes $O(n^2)$ time to compute the k_{ij}s. Interestingly, k_{ij} is proportional to $B(i,j,p)^{0.2}$ and inversely proportional to $i^{0.8}$. Recall that $B(i,j,p)$ denotes the probability of reaching node $N(i,j)$ and i denotes how distant node $N(i,j)$ is from today. The allocation scheme is clearly nonuniform.

Fig. 5 plots the k_{ij} for the case of $n = 20$ and $k = 50$. It shows how the roughly $kn^2/2 = 10,000$ states are distributed among the nodes for uniform and nonuniform allocation schemes. For the nonuniform allocation scheme, at any given time i, more states are given to the center nodes than the peripheral ones. This is consistent with the intuition that nodes with a heavier probability weight should be given more states. Although nodes with small probability weights are given fewer states, resulting in potentially larger errors, this negative impact will be counterbalanced by their smaller probabilities. It turns out that the choice of k_{ij} in Eq. (8) strikes the right balance.

The algorithm computes the option value for each allocated state via backward induction. Suppose the option value $V(i,S,P)$ for the state (i,S,P) at node $N(i,j)$ is desired. Consider the up move from the node to the node $N(i+1,j)$ with stock price Su. From Eq. (2), the next running sum equals $P+Su$, and the option value $V(i+1,Su,P+Su)$ needs to be calculated. In the event that $P+Su$ equals the running sum of some allocated state of node $N(i+1,j)$, then $V(i+1,Su,P+Su)$ is already available. On the other hand, if

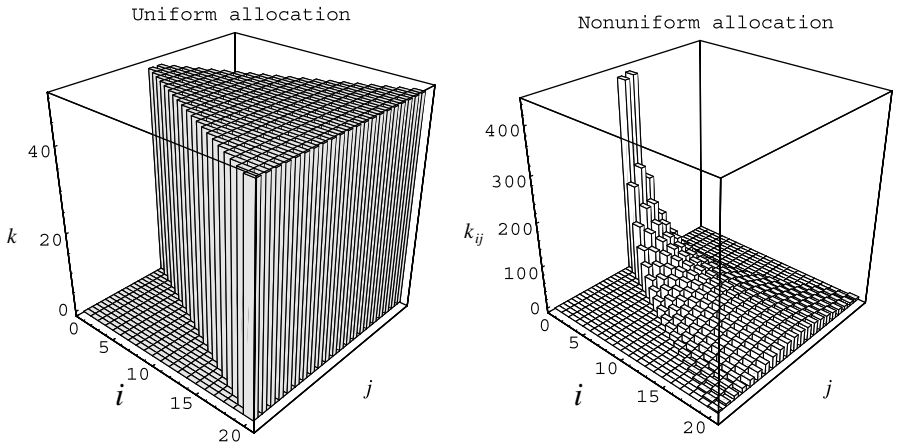


Fig. 5. Distribution of k_{ij}, the number of states for each node on the binomial lattice under the uniform and the nonuniform allocation schemes. For this plot, $p = 0.486961$, $n = 20$, and $k = 50$.

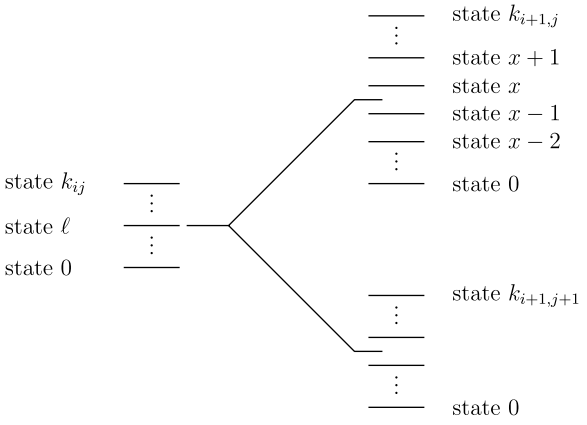


Fig. 6. Interpolation. Each of the $k_{ij}+1$ states at $N(i,j)$ moves up to a running sum of node $N(i+1,j)$ and down to a running sum of node $N(i+1,j+1)$. Typically, neither running sum corresponds to any allocated state.

$P+Su>(n+1)X$, apply formula (4) to obtain the option value. Finally, if $P+Su$ does not correspond to any running sum associated with an allocated state at node $N(i+1,j)$, interpolation is used to obtain the desired option value as follows. If $P+Su$ is sandwiched by two running sums whose corresponding option values are zero, we set $V(i+1,Su,P+Su)=0$. Otherwise, let $P+Su$ be bracketed by four running sums, two from below and two from above (see Fig. 6)

$$(x-2)\frac{(n+1)X}{k_{i+1,j}}<(x-1)\frac{(n+1)X}{k_{i+1,j}}<P+Su<x\frac{(n+1)X}{k_{i+1,j}}<(x+1)\frac{(n+1)X}{k_{i+1,j}}. \tag{9}$$

Apply the 4-point polynomial interpolation from the four bracketing running sums' corresponding option values to obtain $V(i+1,Su,P+Su)$, i.e.,

$$\begin{aligned}V(i+1,Su,P+Su)=&\alpha_1 V\left(i+1,Su,(x-2)\frac{(n+1)X}{k_{i+1,j}}\right)+\alpha_2 V\left(i+1,Su,(x-1)\frac{(n+1)X}{k_{i+1,j}}\right)\\&+\alpha_3 V\left(i+1,Su,x\frac{(n+1)X}{k_{i+1,j}}\right)+\alpha_4 V\left(i+1,Su,(x+1)\frac{(n+1)X}{k_{i+1,j}}\right),\end{aligned} \tag{10}$$

where x is defined in Eq. (9). Similarly, repeat the steps for the down move to obtain the option value $V(i+1,Sd,P+Sd)$

$$\begin{aligned}V(i+1,Sd,P+Sd)=&\alpha'_1 V\left(i+1,Sd,(y-2)\frac{(n+1)X}{k_{i+1,j+1}}\right)+\alpha'_2 V\left(i+1,Sd,(y-1)\frac{(n+1)X}{k_{i+1,j+1}}\right)\\&+\alpha'_3 V\left(i+1,Sd,y\frac{(n+1)X}{k_{i+1,j+1}}\right)+\alpha'_4 V\left(i+1,Sd,(y+1)\frac{(n+1)X}{k_{i+1,j+1}}\right).\end{aligned} \tag{11}$$

Finally, set $V(i,S,P)$ by formula (3). Recall that $|\alpha_i|<1$ and $|\alpha'_i|<1$. Hence the error introduced at a node will not explode during backward induction.

In the literature, both linear and quadratic interpolations have been used. It is also common to apply interpolation in the logarithmic domain. However, as we shall see, our higher-order interpolation combined with the right choice of k_{ij} achieves the desired goal of a convergent quadratic-time algorithm.

Other lattice models than the CRR binomial model are known, e.g., the binomial model of Jarrow and Rudd [38], the binomial model of Trigeorgis [39], the MOT binomial model, which makes the strike price lie at the middle of the lattice at maturity [40], and the various trinomial models surveyed in Lyuu [41]. Each of these alternative models enjoys certain advantages over the CRR model in pricing specific options. However, they yield qualitatively and quantitatively similar numerical results as our CRR-based algorithm. This finding is consistent with those in [16] and lends further support to the correctness of our analysis.

5. Error analysis

Throughout this paper, the computer is equipped with a 2.4 GHz Intel Pentium-4 CPU running Linux and the GNU C compiler. We start with $X = 0$ as exact option values are then available from [18]. From formula (4), the option value under the discrete-time CRR binomial model equals

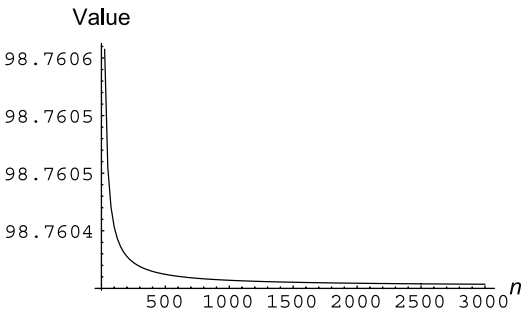
$$V(0, S_0, S_0) = R^{-n}\left\{\left[S_0 + S_0 R\frac{1-R^n}{1-R}\right] \Big/ (n+1)\right\}.$$

Fig. 7 considers a scenario whose exact option value is 98.76035189. The calculated option values under the CRR model do indeed decrease nicely towards the exact value. Fig. 8 depicts the convergence in terms of the magnitude of error. The CRR model clearly converges to the continuous-time model at the convergence rate of $O(n^{-1})$.

In general, with only $O(n^2)$ states (that is, a constant k), our algorithm has an error of $O(n^{-1})$ even for nonzero strike prices. See Appendix B for the proof. (In that appendix, we also apply the same methodology to analyze our algorithmic idea with quadratic interpolation instead of cubic interpolation. This results in a running time of $O(n^{2.167})$.) To confirm the convergence rate, Fig. 9 plots the convergence behaviors of our algorithm with the nonzero strike prices of 95 and 100, respectively, for $25 \leqslant n \leqslant 3000$. The plots support the theoretical claim of a convergence rate of $O(n^{-1})$.

Klassen has demonstrated that extrapolation can speed up the convergence dramatically [16]. Let $f_k(n)$ denote the option value calculated by our algorithm with n periods and $kn^2/2$ states. The first-order Richardson's extrapolation says the formula

$$2f_k(2n) - f_k(n)$$



n	Option value
25	98.7605576353383
50	98.7604547609814
75	98.7604204695319
100	98.7604033238297
⋮	
650	98.7603598000668
⋮	
1000	98.7603570302950
2000	98.7603544582460
3000	98.7603536018050

Fig. 7. Select option values as $n \to \infty$: the zero-strike case. The parameters are from Table 1 of [17]: $S = 100,\ X = 0,\ \tau = 0.25,\ \sigma = 0.1$, and $r = 0.1$. The exact option value is 98.76035189.

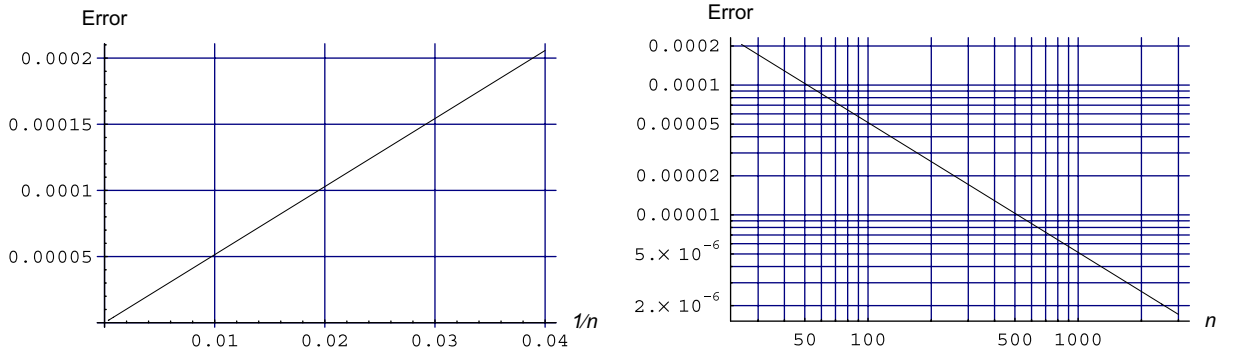


Fig. 8. Convergence of the binomial model to the exact option value as $n \to \infty$: the zero-strike case. The parameters are identical to those in Fig. 7.

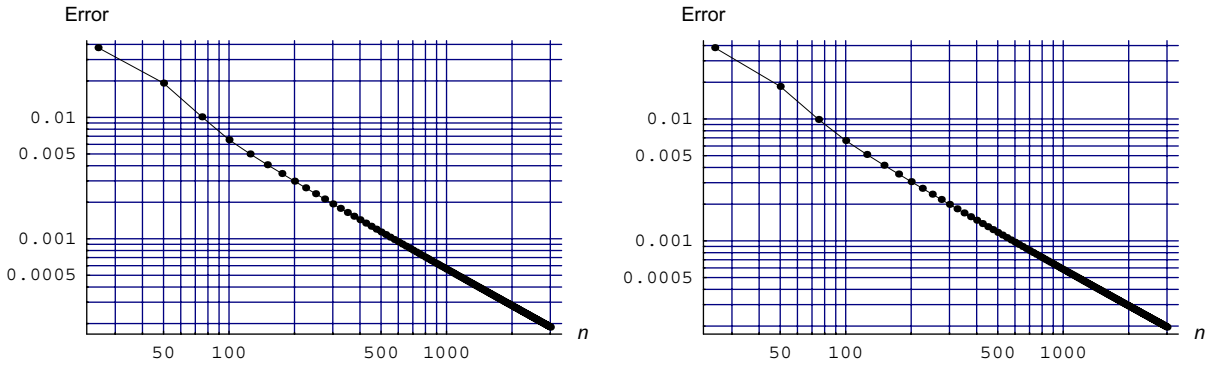


Fig. 9. Convergence to the exact option value as $n \to \infty$: two positive strike cases. The parameters are from [43]: $S = 100,\ X = 95$ (left) and 100 (right), $\tau = 1,\ \sigma = 0.5$, and $r = 0.09$. Our algorithm uses $25 \leqslant n \leqslant 3000$ and $k = 50$.

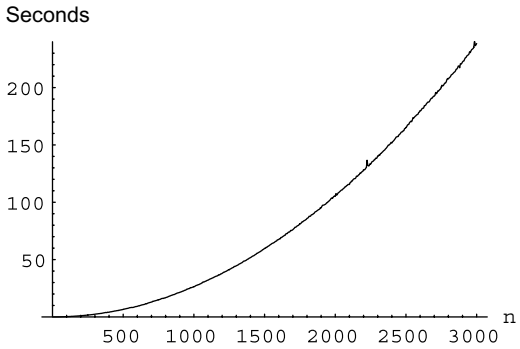


Fig. 10. Running times. The parameters are from case 1 of Table 4 in [3]: $S = 1.9,\ X = 2.0,\ \tau = 1,\ \sigma = 0.5,\ r = 0.05$, and $k = 50$.

provides a good estimate of the desired $f_\infty(\infty)$ [42]. After extensive numerical experiments, we will eventually settle for $k = 50$.

Recall that the running time is $O(n^2)$. Fig. 10 confirms the quadratic growth of the algorithm's running time. The algorithm is furthermore memory efficient as it can run for very large n without difficulties. For example, the running time is under 4 minutes even at $n = 3000$. The space complexity equals $O(n^{1.4})$. This fact can be proved by counting the number of states per time step and noting that only states at a given time step need to be allocated space, not the entire $kn^2/2$ states.

We now investigate the impacts of k, n, and σ on the convergence of the algorithm. Increasing k has the expected effect of improved accuracy and convergence speed at the expense of more running time, which is proportional to k. This phenomenon is illustrated in Fig. 11, where we find $k = 50$ yields satisfactory results. Hence all option values quoted for our algorithm are calculated based on this choice of k unless stated otherwise. The state count equals $25 \times n^2$. Next we fix n and see how increasing k affects the convergence. As expected, the error drops quickly with increasing k as shown in Fig. 12. Overall, we find $2f_{50}(400) - f_{50}(200)$ gives highly accurate extrapolated option values with reasonable running times. Hence all *extrapolated* option values are given by this formula, which has a total state count of about

$$25 \times 200^2 + 25 \times 400^2 = 5 \times 10^6.$$

We argued earlier that a high σ should not affect the accuracy of the algorithm. Fig. 13 supports this claim with σ ranging from 5% to 100%.

6. Numerical evaluation

The Hull–White algorithm runs in $O(n^{3.5})$ time and uses linear interpolation, whereas the PDE method of Forsyth et al. runs in $O(n^3)$ time and uses quadratic interpolation [17]. Both are convergent, but neither is as

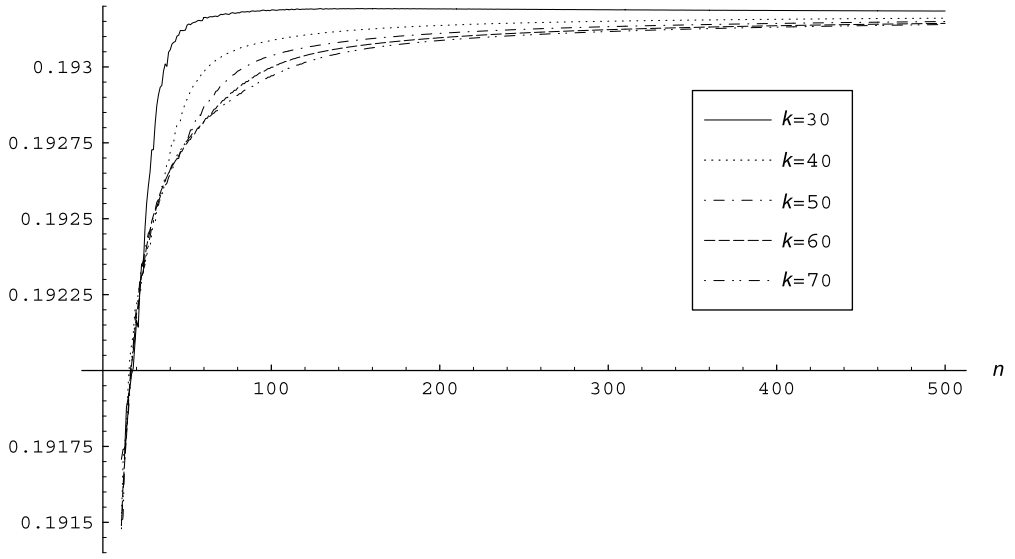


Fig. 11. Convergence as $n \to \infty$. The parameters are from case 1 of Table 4 in [3]: $S = 1.9,\ X = 2.0,\ \tau = 1,\ \sigma = 0.5$, and $r = 0.05$. The prices given in [3] range from 0.193 to 0.195. Select extrapolated option values of our algorithm are 0.193149 ($k = 30$), 0.193149 ($k = 40$), 0.193155 ($k = 50$), 0.193155 ($k = 60$), and 0.193157 ($k = 70$).

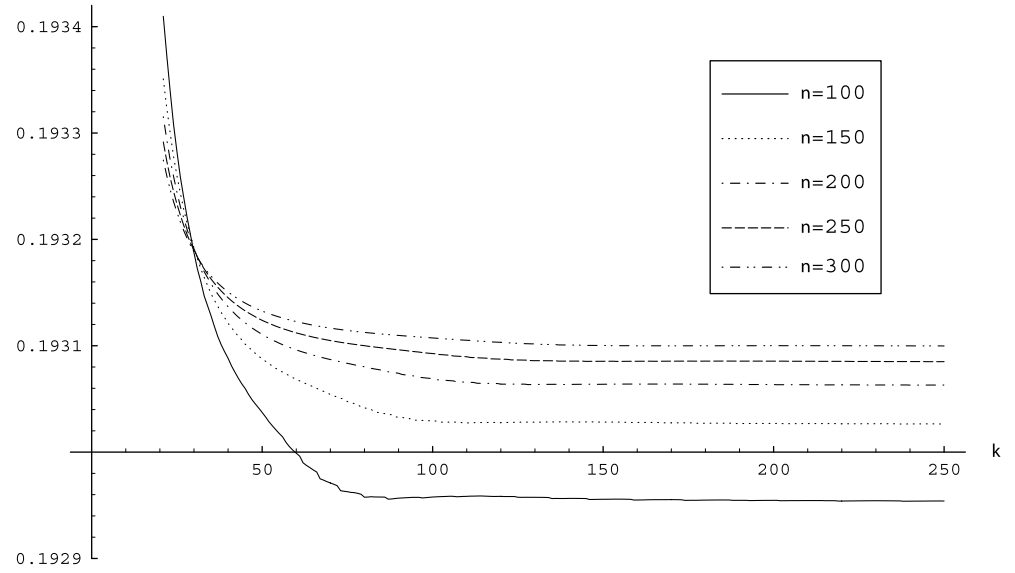


Fig. 12. Convergence as $k \to \infty$. The parameters are identical to those in Fig. 11. The prices given in [3] range from 0.193 to 0.195.

efficient as our $O(n^2)$ time bound. Table 1 compares the performance of the Hull–White algorithm, the PDE method, and our algorithm. The exact value for case 1 in Table 1 is conjectured to be 1.8515 ± 0.0001 based on 1.8514 (the extrapolated option value by the PDE method) and 1.8516 (the extrapolated option value by the Hull–White algorithm). Our algorithm's 1.8515402 falls within the band. The exact value for case 2 in Table 1 is conjectured to be 28.40525 ± 0.00015 based on 28.4051 (the extrapolated option value by the Hull–White algorithm) and 28.4054 (the extrapolated option value by the PDE method). Our algorithm's 28.4052033 again falls within their band. In contrast, the forward-shooting-grid (FSG) method of [18] fails to converge to within either band whether it uses nearest interpolation or linear interpolation [17].

We next compare our algorithm with analytical and semi-analytical approaches. As Fu et al. [3], Ju [4] and Zhang [1,43] have assessed the methods thoroughly, we will only deal with Ju's method and Zhang's various

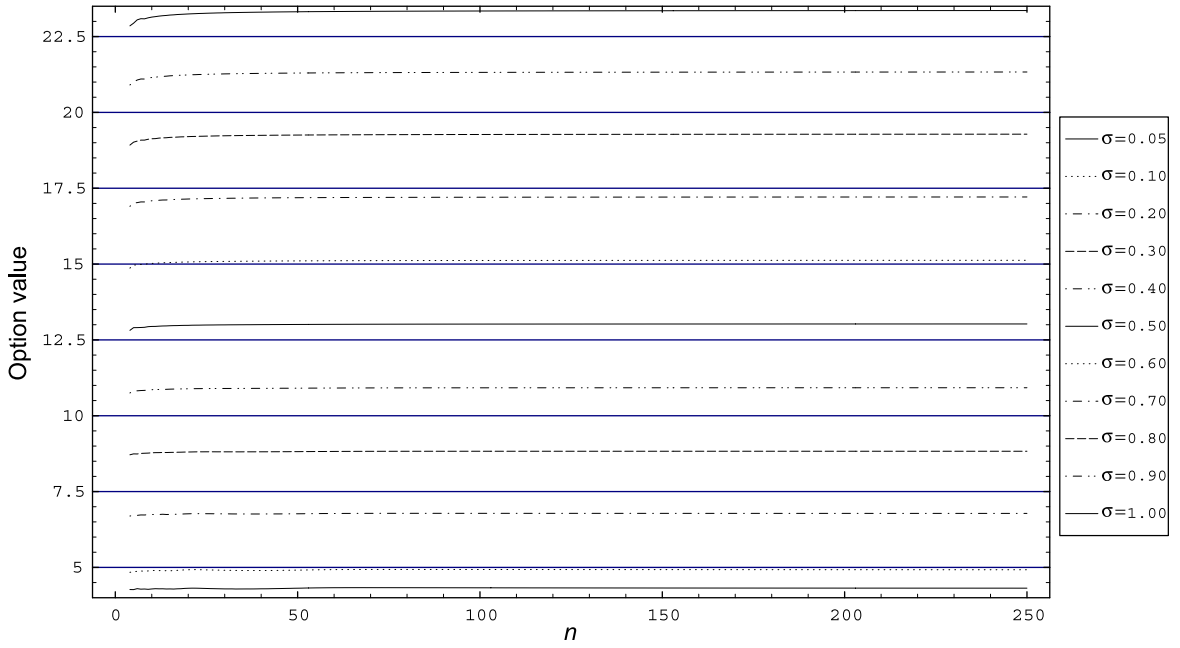


Fig. 13. Convergence to the exact option values at various σ. The parameters are: $S = 100,\ X = 100,\ \tau = 1$, and $r = 0.09$.

Table 1
Comparison with the Hull–White and PDE methods

Algorithms						
	Hull–White		PDE		Ours	
n	Option value	Time (s)	Option value	Time (s)	Option value	Time (s)
Case 1: $S = 100,\ X = 100,\ r = 0.1,\ \sigma = 0.1,\ \tau = 0.25$						
50	1.8486	18.0	1.8478	4.8	1.8714720	0.06
100	1.8501	204.0	1.8492	55.0	1.9095930	0.25
200	1.8508	2293.0	1.8503	313.0	1.8891953	1.04
400	1.8512	25918.0	1.8509	2540.0	1.8703678	4.23
∞	1.8516		1.8514		1.8515402	
Case 2: $S = 100, X = 100, r = 0.1, \sigma = 0.5, \tau = 5$						
50	28.3899	15.0	28.3573	5.5	28.3893142	0.06
100	28.3972	168.0	28.3842	36.0	28.3973455	0.26
200	28.4011	1893.0	28.3952	280.0	28.4013633	1.07
400	28.4031	21370.0	28.4003	2278.0	28.4032833	4.31
∞	28.4051		28.4054		28.4052033	

The parameters are from Tables 3 and 4 of [17]. The numbers quoted for the Hull–White are based on calculations using the finest grids. The "∞" row lists the extrapolated option values. The CPU times of [17] are based on a Sun Ultrasparc workstation (model not specified in their paper).

methods for their superior convergence behavior. Zhang is a semi-analytical method that requires the numerical evaluation of an inhomogeneous linear diffusion equation [43]. It is generally accepted that the numerical results given in [43] are exact. Ju [4] and Zhang [1] in contrast are analytical approximations.

Table 2 shows that our algorithm runs for about 5.4 s and obtains results extremely close to the exact results of [43]. The third-order approximate formula `AA2` and the fourth-order approximate formula `AA3` of [1] are more accurate than ours for small σ. However, their accuracy deteriorates with increasing σ, whereas ours does not. Table 3 investigates a wider range of volatilities. Zhang [1] says that Table 3 is not favorable to his formulas (though they are still better than others) because the series converges slower when the volatility is

Table 2
Comparison with [1,43]

X	σ	r	Exact	AA2	AA3	Ours	Time (sec)	
95	0.05	0.05	7.1777275	7.1777244	7.1777279	7.178812	5.30	
100			2.7161745	2.7161755	2.7161744	2.715613	5.37	
105			0.3372614	0.3372601	0.3372614	0.338863	5.47	
95		0.09	8.8088392	8.8088441	8.8088397	8.808717	5.39	
100			4.3082350	4.3082253	4.3082331	4.309247	5.30	
105			0.9583841	0.9583838	0.9583841	0.960068	5.34	
95		0.15	11.0940944	11.0940964	11.0940943	11.093903	5.44	
100			6.7943550	6.7943510	6.7943553	6.795678	5.31	
105			2.7444531	2.7444538	2.7444531	2.743798	5.52	
			RMSE:	0.0000041	0.0000007	0.001062		
90	0.10	0.05	11.9510927	11.9509331	11.9510871	11.951610	5.36	
100			3.6413864	3.6414032	3.6413875	3.642325	5.38	
110			0.3312030	0.3312563	0.3311968	0.331348	5.42	
90		0.09	13.3851974	13.3851165	13.3852048	13.385563	5.41	
100			4.9151167	4.9151388	4.9151177	4.914254	5.42	
110			0.6302713	0.6302538	0.6302717	0.629843	5.30	
90		0.15	15.3987687	15.3988062	15.3987860	15.398885	5.39	
100			7.0277081	7.0276544	7.0277022	7.027385	5.39	
110				1.4136149	1.4136013	1.4136161	1.414953	5.33
			RMSE:	0.0000670	0.0000072	0.000678		
90	0.20	0.05	12.5959916	12.5957894	12.5959304	12.596052	5.37	
100			5.7630881	5.7631987	5.7631187	5.763664	5.35	
110			1.9898945	1.9894855	1.9899382	1.989962	5.43	
90		0.09	13.8314996	13.8307782	13.8313482	13.831604	5.38	
100			6.7773481	6.7775756	6.7773833	6.777748	5.35	
110			2.5462209	2.5459150	2.5462598	2.546397	5.34	
90		0.15	15.6417575	15.6401370	15.6414533	15.641911	5.39	
100			8.4088330	8.4091957	8.4088744	8.408966	5.38	
110			3.5556100	3.5554997	3.5556415	3.556094	5.29	
			RMSE:	0.00063735	0.0001190	0.000301		
90	0.30	0.05	13.9538233	13.9555691	13.9540973	13.953937	5.36	
100			7.9456288	7.9459286	7.9458549	7.945918	5.44	
110			4.0717942	4.0702869	4.0720881	4.071945	5.39	
90		0.09	14.9839595	14.9854235	14.9841522	14.984037	5.35	
100			8.8287588	8.8294164	8.8289978	8.829033	5.40	
110			4.6967089	4.6956764	4.6969698	4.696895	5.34	
90		0.15	16.5129113	16.5133090	16.5128376	16.512963	5.38	
100			10.2098305	10.2110681	10.2101058	10.210039	5.36	
110			5.7301225	5.7296982	5.7303567	5.730357	5.33	
			RMSE:	0.0011016	0.0002383	0.000193		

The parameters are from Table 1 of [1]. The options are calls with $S = 100$ and $\tau = 1$.

greater. Our algorithm does not suffer from the same problem; in fact, the RMSEs in Tables 2 and 3 show that it improves with increasing volatility.

It is illuminating to compare the complexity of our algorithm with that of [43]. In [43], the PDE is solved with the Crank–Nicolson scheme, which has a linear-time complexity when dealing with a constant-bandwidth linear system. Zhang employs 4000 grid points in the spatial dimension and $\Delta t = 1/800$ in the time dimension for his choice of options in Table 2. Hence the total time is proportional to $4000 \times \tau/\Delta t = 3.2 \times 10^6 \times \tau$. In general, Zhang suggests $40,000 \times \sigma\tau^{1.5}$ grid points in the spatial dimension. The overall running time is then proportional to

Table 3
Comparison with [1,43] with a wide range of volatilities

X	σ	Exact	AA2	AA3	Ours	Time (sec)
95	0.05	8.8088392	8.80884	8.80884	8.808717	5.23
100		4.3082350	4.30823	4.30823	4.309247	5.19
105		0.9583841	0.95838	0.95838	0.960068	5.10
		RMSE:	0.00000	0.00000	0.001136	
95	0.1	8.9118509	8.91171	8.91184	8.912238	5.16
100		4.9151167	4.91514	4.91512	4.914254	5.28
105		2.0700634	2.07006	2.07006	2.072473	5.07
		RMSE:	0.00008	0.00001	0.001494	
95	0.2	9.9956567	9.99597	9.99569	9.995661	5.21
100		6.7773481	6.77758	6.77738	6.777748	5.16
105		4.2965626	4.29643	4.29649	4.297021	5.13
		RMSE:	0.00024	0.00005	0.000351	
95	0.3	11.6558858	11.65747	11.65618	11.656062	5.21
100		8.8287588	8.82942	8.82900	8.829033	5.11
105		6.5177905	6.51763	6.51802	6.518063	5.21
		RMSE:	0.00100	0.00026	0.000245	
95	0.4	13.5107083	13.51426	13.51182	13.510861	5.35
100		10.9237708	10.92507	10.92474	10.923943	5.20
105		8.7299362	8.72936	8.73089	8.730102	5.29
		RMSE:	0.00221	0.00101	0.000164	
95	0.5	15.4427163	15.44890	15.44587	15.442822	5.24
100		13.0281555	13.03015	13.03107	13.028271	5.12
105		10.9296247	10.92800	10.93253	10.929736	5.25
		RMSE:	0.00387	0.00299	0.000111	
95	0.6	–	–	–	17.406402	5.21
100		–	–	–	15.128426	5.10
105		–	–	–	13.113874	5.13
95	0.8	–	–	–	21.349949	4.98
100		–	–	–	19.288780	5.06
105		–	–	–	17.423935	5.09
95	1.0	–	–	–	25.252051	5.13
100		–	–	–	23.367535	5.15
105		–	–	–	21.638238	5.13

The parameters are from Table 2 of [1]. The options are calls with $S = 100$, $r = 0.09$, and $\tau = 1$.

$$40,000 \times \sigma\tau^{1.5}\frac{\tau}{\Delta t} = 3.2 \times 10^7 \times \sigma\tau^{2.5}.$$

As a comparison, our algorithm's running time is proportional to $25 \times n^2$. Suppose we take $\Delta t = 1/800$ as in [43], then the running time of our algorithm becomes proportional to

$$25 \times \left(\frac{\tau}{\Delta t}\right)^2 = 1.6 \times 10^7 \times \tau^2.$$

The above two running-time formulas indicate that Zhang's method is expected to lag in speed compared with ours as τ increases. Methods that deteriorate for *small* $\sigma^2\tau$ are mentioned in [3].

Volatilities never exceed 50% in [1,4]. But as shown in Fig. 13, our algorithm has no difficulties handling volatilities as large as 100%; it actually improves with higher σ. To our best knowledge, [26] is the only other

work that calculates European-style Asian option prices for $\sigma > 50\%$. Not surprisingly, it also takes advantage of formula (4).

Table 4 compares our algorithm with the exact one of [43] and the approximate formula of [4] for the case of the longer maturity of 3 years. Our algorithm obtains lower RMSEs here than in Table 3. The explanation is straightforward: Longer maturity means more paths will have running sums exceeding $(n+1)X$, thus eligible for the exact pricing formula (4). Our algorithm again enjoys an advantage over others as σ increases.

One-dimensional PDEs result in much more efficient algorithms than the two-dimensional PDE used in Table 1. We will focus on the following one-dimensional PDE of [21]:

$$\frac{\partial u}{\partial t} + r\left(1 - \frac{t}{\tau} - z\right)\frac{\partial u}{\partial z} + \frac{\left(1 - \frac{t}{\tau} - z\right)^2 \sigma^2}{2}\frac{\partial^2 u}{\partial z^2} = 0 \tag{12}$$

with the terminal condition $u(\tau, z) = \max(z, 0)$ and the Neumann bundary conditions $\partial^2 u/\partial z^2 = 0$ at $z = a$ and $z = 1$, where $a < 0$. To discretize Eq. (12), we employ n grid points in the time dimension and m grid points in

Table 4
Comparison with [4,43]

X	σ	Exact	TE6	Ours	Time (s)
95	0.05	15.1162646	15.11626	15.116230	5.17
100		11.3036080	11.30360	11.304034	5.29
105		7.5533233	7.55335	7.554073	5.25
		RMSE:	0.00002	0.000498	
95	0.1	15.2138005	15.21396	15.213921	5.10
100		11.6376573	11.63798	11.637813	5.09
105		8.3912219	8.39140	8.391189	5.19
		RMSE:	0.00023	0.000115	
95	0.2	16.6372081	16.63942	16.637276	5.25
100		13.7669267	13.76770	13.767043	5.20
105		11.2198706	11.21879	11.220047	5.19
		RMSE:	0.00149	0.000128	
95	0.3	19.0231619	19.02652	19.023236	5.29
100		16.5861236	16.58509	16.586222	5.18
105		14.3929780	14.38751	14.393083	5.04
		RMSE:	0.00375	0.000093	
95	0.4	21.7409242	21.74461	21.740973	5.28
100		19.5882516	19.58355	19.588307	5.16
105		17.6254416	17.61269	17.625501	5.23
		RMSE:	0.00813	0.000055	
95	0.5	24.5718705	24.57740	24.571913	5.32
100		22.6307858	22.62276	22.630828	5.18
105		20.8431853	20.82213	20.843226	5.27
		RMSE:	0.01340	0.000042	
95	0.6	–	–	27.425278	5.26
100		–	–	25.655297	5.13
105		–	–	24.013011	5.18
95	0.8	–	–	33.031740	5.02
100		–	–	31.535716	5.05
105		–	–	30.133505	5.04
95	1.0	–	–	38.361352	5.01
100		–	–	37.085174	5.04
105		–	–	35.881483	5.04

The exact values are based on and quoted from Table 7 of [43]. Ju's Taylor expansion method is denoted as TE6. The parameters are from Table 2 of [4] and Table 7 of [43]. The options are calls with $S = 100$, $r = 0.09$, and $\tau = 3$.

the spatial dimension. Following [21], we use a Crank–Nicolson discretization scheme and obtain an $O(nm)$ running time. Table 5 compares (1) the method of [43], (2) the PDE method with $n = 100$, $m = 2000$, and $t \times z \in [0, \tau] \times [-1, 1]$ (called PDE1), (3) the PDE method with $n = 100$, $m = 10,000$, and $t \times z \in [0, \tau] \times [-9, 1]$ (called PDE2), and (4) our lattice algorithm. PDE1 is very accurate for low to medium volatility levels. For high volatility levels and/or long maturities, however, m must be increased to obtain accuracy comparable to our lattice algorithm. For example, PDE2 contains $m = 10,000$ grid points in the spatial dimension and over a wider range of $[-9, 1]$ vs. PDE1's $[-1, 1]$. Observe that PDE2 contains 10^6 grid points, whereas our lattice algorithm contains 5×10^6 states. Our implementation of PDE2 takes about 9.62 s to complete, whereas our lattice algorithm takes about 5.5 s to complete. The data again confirm the extreme accuracy achieved by our lattice algorithm for high volatility levels and/or long maturities.

Rogers and Shi give very efficient lower and upper bounds for the Asian option [6]. The lower bounds in particular are known to be extremely accurate. Table 6 compares our algorithm with the lower bounds of [6].

Table 5
Comparison with the one-dimensional PDE method of [21]

X	σ	$\tau = 1$				$\tau = 3$			
		Exact	PDE1	PDE2	Ours	Exact	PDE1	PDE2	Ours
95	0.05	8.8088392	8.8088241	8.8088241	8.808717	15.1162646	15.1162526	15.1162526	15.116230
100		4.3082350	4.3080602	4.3080602	4.309247	11.3036080	11.3035792	11.3035792	11.304034
105		0.9583841	0.9583277	0.9583277	0.960068	7.5533233	7.5531978	7.5531978	7.554073
		RMSE:	0.0001064	0.0001064	0.001136		0.0000747	0.0000747	0.000498
95	0.1	8.9118509	8.9118054	8.9118054	8.912238	15.2138005	15.2137661	15.2137661	15.213921
100		4.9151167	4.9150253	4.9150253	4.914254	11.6376573	11.6376011	11.6376011	11.637813
105		2.0700634	2.0700251	2.0700251	2.072473	8.3912219	8.3911498	8.3911498	8.391189
		RMSE:	0.0000630	0.0000630	0.001494		0.0000564	0.0000564	0.000115
95	0.2	9.9956567	9.9956323	9.9956323	9.995661	16.6372081	16.6371770	16.6371770	16.637276
100		6.7773481	6.7773279	6.7773279	6.777748	13.7669267	13.7668950	13.7668950	13.767043
105		4.2965626	4.2964614	4.2964614	4.297021	11.2198706	11.2198412	11.2198412	11.220047
		RMSE:	0.0000612	0.0000612	0.000351		0.0000307	0.0000307	0.000128
95	0.3	11.6558858	11.6558892	11.6558892	11.656062	19.0231619	19.0230953	19.0231388	19.023236
100		8.8287588	8.8287699	8.8287699	8.829033	16.5861236	16.5860134	16.5861083	16.586222
105		6.5177905	6.5178134	6.5178134	6.518063	14.3929780	14.3927638	14.3929591	14.393083
		RMSE:	0.0000148	0.0000148	0.000245		0.0001443	0.0000194	0.000093
95	0.4	13.5107083	13.5107373	13.5107373	13.510861	21.7409242	21.7359140	21.7409067	21.740973
100		10.9237708	10.9238047	10.9238049	10.923943	19.5882516	19.5801909	19.5882367	19.588307
105		8.7299362	8.7299785	8.7299789	8.730102	17.6254416	17.6129231	17.6254290	17.625501
		RMSE:	0.0000355	0.0000357	0.000164		0.0090699	0.0000151	0.000055
95	0.5	15.4427163	15.4427436	15.4427631	15.442822	24.5718705	24.5164835	24.5718583	24.571913
100		13.0281555	13.0281668	13.0282104	13.028271	22.6307858	22.5534589	22.6307744	22.630828
105		10.9296247	10.9295940	10.9296853	10.929736	20.8431853	20.7378307	20.8431724	20.843226
		RMSE:	0.0000246	0.0000544	0.000111		0.0819487	0.0000122	0.000042
95	0.6	–	17.4057119	17.4063840	17.406402		27.1922830	27.4252385	27.425278
100		–	15.1272033	15.1284092	15.128426		25.3547907	25.6552489	25.655297
105		–	13.1117954	13.1138637	13.113874		23.6323908	24.0129680	24.013011
95	0.8	–	21.3206229	21.3500057	21.349949		31.8446547	33.0316957	33.031740
100		–	19.2465024	19.2888389	19.288780		30.1240393	31.5356736	31.535716
105		–	17.3646285	17.4239955	17.423935		28.4742281	30.1334450	30.133505
95	1.0	–	25.0465250	25.2521580	25.252051		35.4451734	38.3595938	38.361352
100		–	23.1006194	23.3676388	23.367535		33.7509030	37.0830464	37.085174
105		–	21.2980435	21.6383464	21.638238		32.1020703	35.8789184	35.881483

PDE1 is based on the 100×2000 grid over $[0, \tau] \times [-1, 1]$. PDE2 is based on the $100 \times 10,000$ grid over $[0, \tau] \times [-9, 1]$. The parameters and numerical data for Exact and Ours are from Tables 3 and 4. The numerical data for PDE1 and PDE2 are from [35]. The options are calls with $S = 100$ and $r = 0.09$.

Table 6
Comparison with the lower bounds of [6]

X	σ	Lower bound	Ours	Fusai	Exact	Monte Carlo
95	0.05	8.8088	8.808717	8.80885	8.8088392	8.81
100		4.3082	4.309246	4.30824	4.3082350	4.31
105		0.9583	0.960069	0.95839	0.9583841	0.95
95	0.10	8.9118	8.912238	8.91185	8.9118509	8.91
100		4.9150	4.914254	4.91512	4.9151167	4.91
105		2.0699	2.072473	2.07007	2.0700634	2.06
90	0.30	14.9827	14.984037	14.98396	–	14.96
100		8.8275	8.829033	8.82876	8.8287588	8.81
110		4.6949	4.696895	4.69671	–	4.68
90	0.50	18.1829	18.188933	18.18885	–	18.14
100		13.0225	13.028271	13.02816	13.0281555	12.98
110		9.1179	9.124414	9.12432	–	9.10
90	0.60	19.9542	19.964542	–	–	19.94
100		15.1186	15.128426	–	–	15.13
110		11.3323	11.342769	–	–	11.36
90	0.80	23.5980	23.622784	–	–	23.61
100		19.2655	19.288780	–	–	19.33
110		15.7164	15.739790	–	–	15.74
90	1.00	27.2565	27.305012	–	–	27.25
100		23.3220	23.367535	–	–	23.36
110		20.0069	20.051542	–	–	20.03

The parameters, lower bounds, and Monte Carlo results for $\sigma \leqslant 0.5$ are from Table 3 of [6], Table 1 of [7] and Table 3 of [9]. The Monte Carlo results for $\sigma > 0.5$ are based on 2×10^6 simulation paths. The options are calls with $S = 100$, $r = 0.09$, and $\tau = 1$. Our algorithm's computed option values and the exact option values are from Table 3. `Fusai` is based on the data in Table 3 of [5] computed using the most computing times. The two boxed numbers are slightly lower than the lower bounds.

The values from [5] are listed for comparison. Exact values, wherever available, are copied from Table 3. The numbers in the table affirm the numerical accuracy of the various algorithms. Our algorithm produces values slightly lower than those of Rogers and Shi in two cases (one at $\sigma = 0.05$ and the other at $\sigma = 0.1$). The situations can be rectified with a slightly higher k of 70 and 65, respectively.

A way to raise our algorithm's accuracy further at low volatilities without increasing k starts with two observations. First, the maximum running sum at each node, $(n+1)X$, is unlikely to be breached for most nodes. Second, the minimum running sum at each node is likely to be much higher than zero. In other words, for most nodes, the minimum running sum and the maximum running sum can be considerably tighter than the current algorithm's $[0, (n+1)X]$ particularly when σ is small. This is because the sizes of stock price moves are more constrained. The numerical accuracy can thus be improved by allocating the $k_{ij} + 1$ states at node $N(i, j)$ for running sums between the *actual* minimum running sum,

Table 7
Comparison with the lower bounds of [6] using the revised algorithm

X	σ	Lower bound	Ours	Ours (revised)	Exact
95	0.05	8.8088	8.808717	8.808855	8.8088392
100		4.3082	4.309246	4.308307	4.3082350
105		0.9583	0.960069	0.958552	0.9583841
		RMSE:	0.001136	0.000106	
95	0.10	8.9118	8.912238	8.912392	8.9118509
100		4.9150	4.914254	4.916203	4.9151167
105		2.0699	2.072473	2.071247	2.0700634
		RMSE:	0.001494	0.000979	

The data and parameters are from Table 6 except that the revised algorithm incorporates tighter running-sum ranges.

$$S_0 \frac{1-d^{i+1}}{1-d} + S_0 d^i u \frac{1-u^{j-i}}{1-u}$$

and the smaller of $(n+1)X$ and the *actual* maximum running sum,

$$S_0 \frac{1-u^{j-i+1}}{1-u} + S_0 u^{j-i} d \frac{1-d^i}{1-d}$$

(see [41] for the straightforward derivations of the above formulas). The added complexity to calculate the two running sums is minuscule. When the low-volatility cases of Table 6 ($\sigma = 0.05, 0.1$) are calculated with the revised algorithm, the results are shown in Table 7. Not only are all the lower bounds observed, the prices are also more accurate than those in Table 6.

7. Conclusions

We have presented an extremely accurate lattice algorithm. It is furthermore provable to run in quadratic time and approach the true value with a linear convergence rate. The algorithm is also faster than other lattice algorithms and two-dimensional PDE methods. Compared with the most efficient one-dimensional PDE methods, it is competitive in running time and overall accuracy. The algorithm relies on a novel methodology to choose the number of states for each node based on the Lagrange multipliers. This idea analyzes how individual nodes' error and probability contribute to the overall interpolation error. In contrast, previous efforts analyze the worst-case error without taking into account its probability, which exaggerates the effects of the error. Extensive numerical comparisons with highly accurate PDE, analytical, and lattice methods confirm the claims about our algorithm's performance. The Lagrange-multiplier methodology has also yielded highly efficient convergent lattice algorithms for discretely sampled Asian options (see [29]).

Appendix A. Derivation of the nodes' state sizes

We shall make the following plausible assumptions along the directions of [17]. The exact solution has continuous bounded derivatives up to the fourth-order (in contrast to second order in [17]) with respect to the stock price S and the running average A, and up to the second order with respect to time t. These derivatives are independent of Δt but may depend on τ and σ.

When deriving explicit error bounds, we shall discard the R^{-n} factor in formula (1) for convenience. This omission serves only to overstate our algorithm's error. The distance between two adjacent states' running averages at node $N(i,j)$ is

$$\Delta a \equiv \frac{(n+1)X}{(i+1)k_{ij}} \leqslant \frac{nX'}{ik_{ij}}, \tag{13}$$

where we let $X\prime \equiv 2X$. The error term of the interpolation polynomial $p(x)$ to the desired $f(x)$ with m points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, $x_1 \leqslant x \leqslant x_m$, is given by

$$\left| \frac{(x-x_1)(x-x_2)\cdots(x-x_m)}{m!} f^{(m)}(\varepsilon) \right| \tag{14}$$

for some ε between x_1 and x_m, where $f^{(m)}$ represents the mth derivative [37]. For our algorithm, $x_1, x_2, \ldots, x_m$ are equidistant running averages with

$$x_2 - x_1 = x_3 - x_2 = \cdots = x_m - x_{m-1} = \Delta a$$

by virtue of Eq. (13). Although our algorithm adopts $m = 4$, we will continue to use m for generality.

By our assumption on the boundedness of the fourth derivative of the option value, $|f^{(m)}(x)| \leqslant M$ for some positive constant M. Define $\Delta x \equiv nX'/ik_{ij}$. Inequality (14) is now bounded above by

$$\left| \frac{(x-x_1)(x-x_2)\cdots(x-x_m)}{m!} M \right| \leqslant (\Delta a)^m M \leqslant (\Delta x)^m M.$$

(The upper bound can be lowered to $(6/19)(\Delta x)^4 M$ as our algorithm adopts $m = 4$ and $x_2 \leqslant x \leqslant x_3$; see [37], p. 156.) The error for each state at node $N(i,j)$ is hence at most

$$(\Delta x)^m M = M\left(\frac{nX'}{ik_{ij}}\right)^m = M(X')^m\left(\frac{n}{ik_{ij}}\right)^m. \tag{15}$$

Any option value $V(i,S,P)$ at node $N(i,j)$ is the discounted value of

$$\begin{aligned} & p\left[\alpha_1 V\left(i+1, Su, (x-2)\frac{(n+1)X}{k_{i+1,j}}\right) + \alpha_2 V\left(i+1, Su, (x-1)\frac{(n+1)X}{k_{i+1,j}}\right)\right. \\ & \left. + \alpha_3 V\left(i+1, Su, x\frac{(n+1)X}{k_{i+1,j}}\right) + \alpha_4 V\left(i+1, Su, (x+1)\frac{(n+1)X}{k_{i+1,j}}\right)\right] \\ & + (1-p)\left[\alpha_1' V\left(i+1, Sd, (y-2)\frac{(n+1)X}{k_{i+1,j+1}}\right) + \alpha_2' V\left(i+1, Sd, (y-1)\frac{(n+1)X}{k_{i+1,j+1}}\right)\right. \\ & \left. + \alpha_3' V\left(i+1, Sd, y\frac{(n+1)X}{k_{i+1,j+1}}\right) + \alpha_4' V\left(i+1, Sd, (y+1)\frac{(n+1)X}{k_{i+1,j+1}}\right)\right] \end{aligned} \tag{16}$$

from Eqs. (10) and (11). As $0 < |\alpha_i|,\ |\alpha_i'| < 1$, the contribution of $V(i,S,P)$ to the total interpolation error is at most

$$pM(X')^m\left(\frac{n}{ik_{ij}}\right)^m + (1-p)M(X')^m\left(\frac{n}{ik_{ij}}\right)^m = M(X')^m\left(\frac{n}{ik_{ij}}\right)^m$$

by Eqs. (15) and (16). Forsyth et al. show that we can ignore the truncation error, which is $O(n^{-1})$, and concentrate on the interpolation error [17]. Because node $N(i,j)$ has probability $B(i,j,p)$, the total error of our algorithm is now

$$\text{error} \leqslant \sum_{i=1}^{n}\sum_{j=0}^{i} B(i,j,p)M(X')^m\left(\frac{n}{ik_{ij}}\right)^m = M(X')^m\sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)n^m}{i^m k_{ij}^m}. \tag{17}$$

Note that we account for each node's contribution to the overall interpolation error weighted by its probability. In contrast, most papers in the literature work out only the maximum error per time step, which is much coarser.

The running time of our algorithm is proportional to the total state count. Thus the time complexity can be measured as

$$\sum_{i=1}^{n}\sum_{j=0}^{i} k_{ij} = \frac{kn^2}{2}. \tag{18}$$

We must solve the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)n^m}{i^m k_{ij}^m} \\ \text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=0}^{i} k_{ij} = \frac{kn^2}{2}. \end{aligned}$$

We next introduce the Lagrangian multiplier λ and rewrite our objective function as

$$f = \sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)n^m}{i^m k_{ij}^m} + \lambda\left(\sum_{i=1}^{n}\sum_{j=0}^{i} k_{ij} - \frac{kn^2}{2}\right). \tag{19}$$

Set the partial derivatives of the objective function (19) to zeros

$$\frac{\partial f}{\partial k_{ij}} = 0, \tag{20}$$

$$\frac{\partial f}{\partial \lambda} = 0. \tag{21}$$

Eq. (20) yields

$$\frac{\partial f}{\partial k_{ij}} = -m\frac{B(i,j,p)n^m}{i^m}k_{ij}^{-(m+1)} + \lambda = 0,$$

which implies

$$k_{ij} = \left(\frac{mn^m B(i,j,p)}{i^m \lambda}\right)^{\frac{1}{m+1}}. \tag{22}$$

Eq. (21) gives equality (18). Substitute k_{ij} from Eq. (22) into Eq. (18) to obtain

$$\sum_{i=1}^{n}\sum_{j=0}^{i}\left(\frac{mn^m B(i,j,p)}{i^m \lambda}\right)^{\frac{1}{m+1}} = \frac{kn^2}{2}.$$

Hence

$$\sum_{i=1}^{n}\sum_{j=0}^{i}\left(\frac{mn^m B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}} = \frac{kn^2}{2}\lambda^{\frac{1}{m+1}},$$

which yields

$$\lambda = \left(\frac{\sum_{i=1}^{n}\sum_{j=0}^{i}\left(\frac{mn^m B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}}{\frac{kn^2}{2}}\right)^{m+1}. \tag{23}$$

Substitute λ from Eq. (23) into Eq. (22) to obtain

$$\begin{aligned} k_{ij} &= \left(\frac{mn^m B(i,j,p)}{i^m \lambda}\right)^{\frac{1}{m+1}} = \left(\frac{mn^m B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}} \frac{\frac{kn^2}{2}}{\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{mn^m B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}} \\ &= \frac{kn^2}{2}\frac{\left(\frac{mn^m B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}}{\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{mn^m B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}} = \frac{kn^2}{2}\frac{\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}}{\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}}. \end{aligned} \tag{24}$$

Finally, substitute $m = 4$ to obtain the desired result.

Appendix B. Analysis of the error bound

Substitute k_{ij} in Eq. (24) into the error term (17) to obtain

$$\begin{aligned} \text{error} &\leqslant M(X')^m \sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)n^m}{i^m\left(\frac{kn^2}{2}\frac{\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}}{\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}}\right)^m} \\ &= M(X')^m \frac{n^m}{\left(\frac{kn^2}{2}\right)^m}\sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)}{i^m\left(\frac{\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}}{\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}}\right)^m} \\ &= M(X')^m \left(\frac{2}{nk}\right)^m \sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)\left(\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}\right)^m}{i^m\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{m}{m+1}}} \end{aligned}$$

$$= M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{s=1}^{n}\sum_{t=0}^{s}\left(\frac{B(s,t,p)}{s^m}\right)^{\frac{1}{m+1}}\right)^m \sum_{i=1}^{n}\sum_{j=0}^{i}\frac{B(i,j,p)}{i^m\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{m}{m+1}}}$$

$$= M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n}\sum_{j=0}^{i}\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}\right)^m \sum_{i=1}^{n}\sum_{j=0}^{i}\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}$$

$$= M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n}\sum_{j=0}^{i}\left(\frac{B(i,j,p)}{i^m}\right)^{\frac{1}{m+1}}\right)^{m+1}. \tag{25}$$

We will take advantage of the following asymptotic result of [44]:

$$\sum_{k=0}^{n/2}\binom{n}{k}^r \sim 2^{nr-1}\left(\frac{2}{n\pi}\right)^{\frac{r-1}{2}} r^{-1/2}. \tag{26}$$

As $\binom{n}{k} = \binom{n}{n-k}$, relation (26) implies

$$\sum_{k=0}^{n}\binom{n}{k}^r \sim 2\left[2^{nr-1}\left(\frac{2}{n\pi}\right)^{\frac{r-1}{2}} r^{-1/2}\right] = 2^{nr}\left(\frac{2}{n\pi}\right)^{\frac{r-1}{2}} r^{-1/2} = O\left(2^{nr} n^{-\frac{r-1}{2}}\right). \tag{27}$$

The summation $\sum_{j=0}^{i} B(i,j,p)^{\frac{1}{m+1}}$ is maximized when $p = 1/2$. So the error (25) can be upper-bounded by choosing $p = 1/2$. Now,

$$\sum_{j=0}^{i} B\left(i,j,\frac{1}{2}\right)^{\frac{1}{m+1}} = \sum_{j=0}^{i}\left[\binom{i}{j}2^{-i}\right]^{\frac{1}{m+1}} = 2^{-\frac{i}{m+1}}\sum_{j=0}^{i}\binom{i}{j}^{\frac{1}{m+1}} = \mathrm{O}\left(2^{-\frac{i}{m+1}} 2^{\frac{i}{m+1}} i^{-\frac{\frac{1}{m+1}-1}{2}}\right) \quad \text{by formula (27)} = \mathrm{O}\left(i^{\frac{m}{2(m+1)}}\right). \tag{28}$$

Formula (25) now becomes

$$\text{error} \leqslant M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n}\frac{\sum_{j=0}^{i} B\left(i,j,\frac{1}{2}\right)^{\frac{1}{m+1}}}{i^{\frac{m}{m+1}}}\right)^{m+1}$$

$$= M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n}\frac{O\left(i^{\frac{m}{2(m+1)}}\right)}{i^{\frac{m}{m+1}}}\right)^{m+1} \quad \text{by formula (28)} \quad = O\left(M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n}\frac{i^{\frac{m}{2(m+1)}}}{i^{\frac{m}{m+1}}}\right)^{m+1}\right)$$

$$= O\left(M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n} i^{-\frac{m}{2(m+1)}}\right)^{m+1}\right).$$

We next upper-bound the summation above

$$M(X')^m \left(\frac{2}{nk}\right)^m \left(\sum_{i=1}^{n} i^{-\frac{m}{2(m+1)}}\right)^{m+1} \leqslant M(X')^m \left(\frac{2}{nk}\right)^m \left(\int_{i=0}^{n} x^{-\frac{m}{2(m+1)}}\,\mathrm{d}x\right)^{m+1}$$

$$= M(X')^m \left(\frac{2}{nk}\right)^m \left(\frac{2(m+1)}{m+2} n^{\frac{m+2}{2(m+1)}}\right)^{m+1}$$

$$= M(X')^m \left(\frac{2}{nk}\right)^m \left(\frac{2(m+1)}{m+2}\right)^{m+1} n^{\frac{m+2}{2}}$$

$$= M(X')^m \left(\frac{2}{k}\right)^m \left(\frac{2(m+1)}{m+2}\right)^{m+1} n^{\frac{-m+2}{2}}.$$

Hence,

$$\text{error} = \mathrm{O}\left(M(X')^m \left(\frac{2}{k}\right)^m \left(\frac{2(m+1)}{m+2}\right)^{m+1} n^{-0.5m+1} \right). \tag{29}$$

Recall $m = 4$ and $X\prime = 2X$ to obtain

$$\text{error} = \mathrm{O}\left(16 \times MX^4 \left(\frac{2}{k}\right)^4 \left(\frac{5}{3}\right)^5 n^{-1} \right) = \mathrm{O}(n^{-1}).$$

Hence 4-point interpolation results in the desired convergence rate of $\mathrm{O}(n^{-1})$.

It does not pay to pick $m > 4$ to reduce the interpolation error below $\mathrm{O}(n^{-1})$ as the discretization error alone is $\mathrm{O}(n^{-1})$. Suppose we adopt the familiar quadratic interpolation ($m = 3$). It is easy to show that the absolute values of the interpolation coefficients are less than 1 as long as the x_i are equidistant. With $m = 3$, we pick $k = n^{1/6}$ in Eq. (29) to obtain an error of $\mathrm{O}(n^{-1})$. This results in an excellent running time of $\mathrm{O}(n^{2.167})$. Hence even quadratic interpolation benefits from our optimization methodology.

References

[1] J. Zhang, Pricing continuously sampled Asian options with perturbation method, Journal of Futures Markets 23 (6) (2003) 535–560.
[2] J. Ingersoll, Theory of Financial Decision Making, Rowman & Littlefield, Savage, MD, 1987.
[3] M. Fu, D. Dilip, T. Wang, Pricing continuous Asian options: a comparison of Monte Carlo and Laplace transform inversion methods, Journal of Computational Finance 2 (2) (1999) 49–74.
[4] N. Ju, Pricing Asian and basket options via Taylor expansion, Journal of Computational Finance 5 (3) (2002) 79–103.
[5] G. Fusai, Pricing Asian options via Fourier and Laplace transforms, Journal of Computational Finance 7 (3) (2004) 87–105.
[6] L. Rogers, Z. Shi, The value of an Asian option, Journal of Applied Probability 32 (1995) 1077–1088.
[7] G. Thompsons, Fast narrow bounds on the value of Asian options, centre for Financial Research, Judge Institute of Management, University of Cambridge, 1999.
[8] J. Nielsen, K. Sandmann, Pricing bounds on Asian options, Journal of Financial and Quantitative Analysis 38 (2) (2003) 449–473.
[9] K.-W. Chen, Y.-D. Lyuu, Accurate pricing formulas for Asian options, Applied Mathematics and Computation, in press, doi:10.1016/j.amc.2006.11.032.
[10] P. Boyle, M. Broadie, P. Glasserman, Monte Carlo methods for security pricing, Journal of Economic Dynamics & Control 21 (1997) 1267–1321.
[11] F. Longstaff, E. Schwartz, Valuing American options by simulation: a simple least-squares approach, Review of Financial Studies 14 (2001) 113–147.
[12] E. Clément, D. Lamberton, P. Protter, An analysis of a least squares regression method for American option pricing, Finance and Stochastics 6 (2002) 449–471.
[13] J. Hull, A. White, Efficient procedures for valuing European and American path-dependent options, Journal of Derivatives 1 (1) (1993) 21–31.
[14] P. Ritchken, L. Sankarasubramanian, A. Vijh, The valuation of path dependent contracts on the average, Management Science 39 (10) (1993) 1202–1213.
[15] R. Zvan, K. Vetzal, P. Forsyth, Discrete Asian barrier options, Journal of Computational Finance 3 (1) (1999) 41–67.
[16] T. Klassen, Simple, fast and flexible pricing of Asian options, Journal of Computational Finance 4 (2001) 89–124.
[17] P. Forsyth, K. Vetzal, R. Zvan, Convergence of numerical methods for valuing path-dependent options using interpolation, Review of Derivatives Research 5 (2002) 273–314.
[18] J. Barraquand, T. Pudet, Pricing of American path-dependent contingent claims, Mathematical Finance 6 (1996) 17–51.
[19] T.-S. Dai, Y.-D. Lyuu, Extremely accurate and efficient algorithms for Asian options with error bounds, in: Proceedings of Quantitative Methods in Finance, Cairns, Australia, 2002.
[20] D. Tavella, C. Randall, Pricing Financial Instruments: The Finite Difference Method, Wiley, New York, 2000.
[21] J. Večeř, A new PDE approach for pricing arithmetic average Asian options, Journal of Computational Finance 4 (4) (2001) 105–113.
[22] F. Dubois, T. Leliévre, Efficient pricing of Asian options by the PDE approach, Journal of Computational Finance 8 (2) (2005) 55–63.
[23] E. Benhamou, Fast Fourier transform for discrete Asian options, Journal of Computational Finance 6 (1) (2002) 49–61.
[24] T.-S. Dai, Y.-D. Lyuu, Efficient, exact algorithms for Asian options with multiresolution lattices, Review of Derivatives Research 5 (2002) 181–203.
[25] T.-S. Dai, G.-S. Huang, Y.-D. Lyuu, An efficient convergent lattice algorithm for European Asian options, Applied Mathematics and Computation 169 (2) (2005) 1458–1471.
[26] D. Aingworth, R. Motawani, J. Oldham, Accurate approximations for Asian options, in: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 2000, pp. 891–900.
[27] P. Boyle, Y. Tian, An explicit finite difference approach to the pricing of barrier options, Applied Mathematical Finance 5 (1998) 17–43.

[28] J.Z. Wei, Valuation of discrete barrier options by interpolations, The Journal of Derivatives 6 (1) (1998) 51–73.
[29] W.W. Hsu, Y.-D. Lyuu, Efficient pricing of discrete Asian options, in: Proceedings of the 3rd IASTED International Conference on Financial Engineering and Applications (FEA 2006), Cambridge, MA, 2004.
[30] J. Dewynne, P. Wilmott, A note on average rate options with discrete sampling, SIAM Journal on Applied Mathematics 55 (1) (1995) 267–276.
[31] V. Henderson, R. Wojakowski, On the equivalence of floating- and fixed-strike Asian options, Journal of Applied Probability 32 (2) (2002) 391–394.
[32] D. Duffie, Dynamic Asset Pricing Theory, second ed., Princeton University Press, Princeton, NJ, 1996.
[33] J. Cox, S. Ross, M. Rubinstein, Option pricing: a simplified approach, Journal of Financial Economics 7 (3) (1979) 229–264.
[34] L. Bouaziz, E. Briys, M. Crouhy, The pricing of forward-starting Asian options, Journal of Banking & Finance 18 (1994) 823–839.
[35] C.-Y. Hsu, Adaptive-mesh finite-volume methods for pricing path-dependent options, Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2005.
[36] T.-S. Dai, Pricing Asian options with lattices, Ph.D. thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 2004.
[37] K. Atkinson (An Introduction to Numerical Analysis), second ed., Wiley, New York, 1989.
[38] R. Jarrow, A. Rudd, Option Pricing, Irwin, Homewood, IL, 1983.
[39] L. Trigeorgis, A log-transformed binomial numerical analysis method for valuing complex multi-option investments, Journal of Financial and Quantitative Analysis 26 (3) (1991) 309–326.
[40] D. Leisen, Pricing the American put option: a detailed convergence analysis for binomial models, Journal of Economic Dynamics & Control 22 (1998) 1419–1444.
[41] Y.-D. Lyuu, Financial Engineering & Computation: Principles, Mathematics, Algorithms, Cambridge University Press, Cambridge, UK, 2002.
[42] P. Kloeden, E. Platen, Numerical Solution of Stochastic Differential Equations, Springer-Verlag, Berlin, 1992.
[43] J. Zhang, A semi-analytical method for pricing and hedging continuously sampled arithmetic average rate options, Journal of Computational Finance 5 (1) (2001) 59–79.
[44] E. Bender, Asymptotic methods in enumeration, SIAM Review 16 (4) (1974) 485–515.