Valuation of Discrete Vanilla Options Using a Recursive Algorithm in a Trinomial Tree Setting

Dennis G. Llemit
Department of Mathematics
Adamson University
1000 San Marcelino Street, Ermita, Manila

Abstract

We present an extension or modification of a recursive algorithm to valuate discrete vanilla options in a trinomial tree setting. The algorithm only uses the terminal values of the option as opposed to the standard method of simulating all nodal values for the entire tree. We then show that the option price under the said algorithm converges to the Black-Scholes price confirming its validity.

Keywords: vanilla options, recursive algorithm, trinomial tree

1 Introduction

An option is a derivative contract which confers to the owner (or buyer) the right but not the obligation to buy or sell certain amounts of an underlying at a future time at a predetermined price. It was first formally traded in the Chicago Board Options Exchange (CBOE) in 1973 but there have been historical accounts that it was used before, specifically in ancient Greece where it was used to speculate on the price of olive oil. In modern times, options have grown to become necessary financial tools in trading and many consider their impact in the industry as revolutionary.

Primarily, an option is used to hedge against risks brought by the inherently uncertain nature of trading. The hedging is done by setting up a replicating portfolio consisting of units of bonds and stocks. The price of the option is given by the law of one price - the unique smallest amount or wealth that is necessary to set up the replicating portfolio[6]. In this set up, the famous Black-Scholes model [1] is used to compute the unique price. For discrete-valued market prices, lattice valuation methods are also employed and two popular alternatives to the Black-Scholes are the Cox-Ross-Rubinstein (CRR) binomial model and the family of trinomial tree lattices.

Options can be classified according to exercise time. European or vanilla type options are derivative contracts that can only be exercised at maturity while American type derivative contracts can be exercised within the securities' lifespan. A third classification which also receives significant interest are path dependent options. These are securities whose prices are contingent on the trajectories of their underlying. Asian and barrier options are two examples of this class of securities.

In the field of computational finance, studies are devoted on developing pricing algorithms that are efficient. Many pricing algorithms have been proposed to implement the Black-Scholes as well as lattice models. Usually, algorithms for pricing options in the lattice framework requires that the option values have to be simulated for the entire tree. In this scheme, the worst case time complexity of the CRR is known to be $O(2^n)$. In 2009, Tina Sol[7] together with her adviser Dr. van der Weide of Technical University of Delft, constructed an algorithm that computes the price of barrier options exactly the same as the CRR model. This algorithm is very simple in that it only requires the terminal values of the option and recursively computes the option premium. In 2015, Llemit [4] completely verified the accuracy of the Sol-van der Weide algorithm versus the CRR and showed that its time complexity is $\Theta(n^2)$.

In this paper we intend to

(1) modify the Sol-van der Weide algorithm in order for it to be applicable to trinomial trees, and

(2) determine the modified algorithm's time complexity.

The first goal is obvious since we want to extend the algorithm to a more general lattice model. As the lattice model contains more nodes, it approximates the dynamics of the Black-Scholes. Thus, the modified algorithm hews closer to the true dynamics compared to algorithms in the binomial framework. The second goal is to check whether the time complexity changes as the algorithm is modified.

2 Trinomial Tree Models

It was Phelim Boyle who considered moving from the dichotomous states of the binomial model in 1986 [2]. Instead of two states, up and down, we allow the underlying to move up, down and stay the same with jump steps u, d, and m, respectively. This is shown by the figure below.

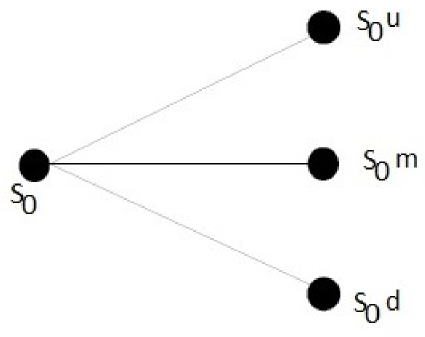

Figure 1: Three Possible States of the Underlying at $t = 1$

To derive the pricing formula, we need to match the first two moments of the underlying S_t which is assumed to follow a geometric Brownian motion. That is, the mean must be equal to

$$\mathbb{E}\left[S_{t+\Delta t}|S_t\right] = up_u + mp_m + dp_d = e^{r\Delta t} \tag{1}$$

and the variance

$$\begin{aligned} Var\left[S_{t+\Delta t}|S_t\right] &= \frac{\mathbb{E}\left[S_{t+\Delta t}^2\right]}{S(t)^2} - \frac{\mathbb{E}\left[S_{t+\Delta t}\right]^2}{S(t)^2} \\ &= u^2 p_u + p_m + d^2 p_d \\ &= e^{2r\Delta t + \sigma^2 \Delta t} - e^{2r\Delta t} \end{aligned} \tag{2}$$

where p_u, p_m, and p_d are transition probabilities, r is the risk-free rate, and σ is the volatility.

Next, we impose an additional constraint

$$u = 1/d \tag{3}$$

in order to preserve the centrality or recombining nature of the tree. The other jump step is usually taken to be $m = 1$. Then, we solve equations (1) , (2) with (3) simultaneously to get

$$p_u = \frac{\left(e^{2r\Delta t+\sigma^2\Delta t} - e^{r\Delta t}\right) u - \left(e^{r\Delta t} - 1\right)}{(u-1)\left(u^2-1\right)} \tag{4}$$

and

$$p_d = \frac{\left(e^{2r\Delta t+\sigma^2\Delta t} - e^{r\Delta t}\right) u^2 - \left(e^{r\Delta t} - 1\right) u^3}{(u-1)\left(u^2-1\right)} \tag{5}$$

while we set

$$p_m = 1 - p_u - p_d. \tag{6}$$

It should be obvious that the values of the transition probabilities are not unique and that other values can be deduced by imposing different constraints. One way is to assign different values for u and d. This implies that there exists a family of trinomial trees that we can use to price options.

In the end, the price of a European type option is given recursively by

$$V_{n-1} = e^{-r\Delta t}\left[p_u V_n^u + p_m V_n^m + p_d V_n^d\right] \tag{7}$$

where n is a nonnegative integer, V_n^u denotes the option price when the underlying takes the value uS one period later. Similar meanings hold for V_n^m and V_n^d.

3 Sol - van der Weide Algorithm

The Sol - van der Weide algorithm was conceptualized in 2009 by Tina Sol with her adviser at Technical University of Delft[7]. The said algorithm computes the price of knock-out call barrier options exactly the same as computations made using the CRR model. The algorithm is efficient in terms of codes since it is very simple. It can be divided into two subprocedures. The first part of the algorithm is called the initialization subprocedure since it essentially initializes the vectors representing the underlying and pay-off values at maturity T for a number of time steps n.

Initialization Subprocedure:

$$S_T = \begin{pmatrix} S_T(n,n) \\ S_T(n,n-2) \\ \vdots \\ S_T(n,-n+2) \\ S_T(n,-n) \end{pmatrix} = \begin{pmatrix} S_0u^n \\ S_0u^{n-1}d \\ \vdots \\ S_0ud^{n-1} \\ S_0d^n \end{pmatrix}$$

$$V_T = \begin{pmatrix} V_T(n,n) \\ V_T(n,n-2) \\ \vdots \\ V_T(n,-n+2) \\ V_T(n,-n) \end{pmatrix} = (K - S_T)^+ \,.\times\, (S_T < B)\,.$$

Here, $.\times$ represents pointwise vector multiplication. The vector $(S_T < B)$ contains logicals 0 or 1. If the underlying is below the barrier B then the vector contains only $1's$. Otherwise, $0's$ will be the only entries.

The second part is called the recursion subprocedure since it recursively runs the operation until it obtains the single entry vector V_0 which is the option premium.

Recursion Subprocedure:
For $i = 1, 2, \cdots, n$ and $h = T/n$ update the pay-off vector

$$V_{T-ih} = e^{-rih} \cdot \left(pV_i^{up} + qV_i^{down}\right) . \times \left(S_{T-ih} < B\right). \qquad (8)$$

Run recursively until V_0 is obtained.

Here, h is the length of each time step. The updating of vectors requires that they must be based from the original vectors V_T and S_T. The vector S_{T-ih} is attained by using either one of the two formulas:

$$S_{T-ih} = u^i.S_i^{down} \qquad (9)$$

or

$$S_{T-ih} = d^i.S_i^{up} \qquad (10)$$

where vectors S_i^{down} and S_i^{up} are obtained by deleting the first i and the last i entries, respectively, of S_T. The purpose of the multiplier u or d is to preserve the centrality or recombining nature of the tree. The same is true for vectors V_i^{up} and V_i^{down}. The deletion of entries goes on until they become single entry vectors or 1×1 matrices.

4 Trinomial Tree Adaptation

In this section, we present the trinomial tree adaptation of the algorithm. We note that we are considering a European-type vanilla call option. The resulting adaptation still consists of two subprocedures.

4.1 Algorithm Modifications

Similar to Sol[7], the initialization subprocedure involves setting the terminal values of the underlying and the option values. We present the said subprocedure below:

Initialization Subprocedure:

$$S_T = \begin{pmatrix} S_T(n,n) \\ S_T(n,n-1) \\ \vdots \\ S_T(n,-n+1) \\ S_T(n,-n) \end{pmatrix} = \begin{pmatrix} S_0u^n \\ S_0u^{n-1}d \\ \vdots \\ S_0ud^{n-1} \\ S_0d^n \end{pmatrix}$$

$$V_T = \begin{pmatrix} V_T(n,n) \\ V_T(n,n-1) \\ \vdots \\ V_T(n,-n+1) \\ V_T(n,-n) \end{pmatrix} = (S_T - K)^+$$

where n is the number of time increments.

Noticeable in this set up is the absence of the vector $(S_T < B)$ because we are working with vanilla options. The said vector is used to terminate the contract in the barrier options setting.

As for the recursion subprocedure, we need only to modify the updating vector V_{T-ih} in order to reflect the three states of the underlying - S_0u, S_0m, and S_0d. Unlike the original algorithm, there is no need to modify the vector S_T because the trinomial tree is recombining even if we delete some nodes from it per iteration.

Recursion Subprocedure:
For $i = 1, 2, \cdots, n$ and $h = \frac{T}{n}$ update the pay-off vector

$$V_{T-ih} = e^{-rih} \cdot \left(p_u V_i^u + p_m V_i^m + q V_i^d\right). \tag{11}$$

Run recursively until V_0 is obtained.

Again, h is the length of each time step and p_u, p_m, and p_d are the transition probabilities. The vector V_i^u is obtained by deleting the last two nodal values of V_T. The vector V_i^m is obtained from V_T by deleting the first and the last entry of from V_T and for V_i^d, deleting the first two entries of V_T.

4.2 Time Complexity Analysis

In this section, we are going to check whether the time complexity was affected by the modifications imposed on the Sol - van der Weide algorithm. To begin our analysis, we define the following measures of time complexity according to Cormen [3] :

Definition (Big O Complexity)**.** *For any monotonic functions $f(n)$ and $g(n)$ where $n \geq 0$, we say that $f(n) = O(g(n))$ when there exist constants $c > 0$ and $n_0 > 0$ such that $f(n) \leq c \cdot g(n)$, for all $n \geq n_0$.*

Definition (Big Omega Complexity)**.** *For any monotonic functions $f(n)$ and $g(n)$ where $n \geq 0$, we say that $f(n) = \Omega(g(n))$ if there exist a constant c such that $f(n) \geq c \cdot g(n)$ for all sufficiently large n.*

Definition (Big Theta Complexity)**.** *For any monotonic functions $f(n)$ and $g(n)$ where $n \geq 0$, we say that $f(n) = \Theta(g(n))$ if there exist constants c_1 and c_2 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all sufficiently large n.*

Then we state a series of theorems from Rosen [5].

Theorem 1. *For any monotonic functions $f(n)$ and $g(n)$ where $n \geq 0$, we say that $f(n) = \Theta(g(n))$ if and only $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ for all sufficiently large n.*

Theorem 2. *Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, where $a_0, a_1, \cdots, a_n$ are real numbers with $a_n \neq 0$. Then $f(x)$ is of order x^n.*

Theorem 3. *Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O\left(\max\left(\left|g_1(x)\right|, \left|g_2(x)\right|\right)\right)$.*

Theorem 4. *Suppose that $f_1(x)$ is $\Omega(g_1(x))$ and $f_2(x)$ is $\Omega(g_2(x))$. Then $(f_1 + f_2)(x)$ is $\Omega\left(\max\left(\left|g_1(x)\right|, \left|g_2(x)\right|\right)\right)$.*

Now, suppose that $MS(n)$ is the running time function for the modified Sol - van der Weide algorithm. We have the following time complexity analyses.

For the worst-case time complexity, we argue that both vectors S_T and V_T have lengths $2n + 1$ in the initialization subprocedure. Putting them together, they contribute $4n+2$ instructions in this subprocedure. Hence, by Theorem (2),

$$\text{initialization subprocedure time complexity } = 4n+2 \in O(n).$$

As for the recursion subprocedure, it now involves a single vector, V_{T-ih} which contract by length two every iteration. This reduction in length can be expressed as

$$\begin{aligned}(2n-1)+(2n-3)+\ldots+3+1 &= \frac{n}{2}\left[2+2(n-1)\right] \\ &= n^2.\end{aligned}$$

According to Theorem (2), $n^2 \in O(n^2)$. Thus the worst-case time complexity of the entire algorithm according to theorem (3) is

$$\begin{aligned}MS(n) &= \text{initialization subprocedure time complexity} \\ &\quad + \text{recursion subprocedure time complexity} \\ &= O(n)+O(n^2) \\ &= O(n^2).\end{aligned}$$

As for the best-case time complexity, we note that the initialization subprocedure has to process all the input elements and reducing its length will affect the accuracy of the computation. Hence, we conclude that the initialization subprocedure has a linear time complexity. That is

$$\text{initialization subprocedure time complexity} = \Omega(n)$$

For the recursion subprocedure, we conclude that it is also quadratic similar to its worst-case counterpart for the following reasons:

1.) the recursion is iterative and contains no conditional statement.

2.) the recursion is based on a closed form equation (11).

Therefore, by Theorem (4), the best-case time complexity of the entire algorithm is

$$\begin{aligned}MS(n) &= \text{initialization subprocedure time complexity} \\ &\quad + \text{recursion subprocedure time complexity} \\ &= \Omega(n)+\Omega(n^2) \\ &= \Omega(n^2).\end{aligned}$$

Since, $MS(n) = O(n^2)$ and $MS(n) = \Omega(n^2)$, we conclude that $MS(n) = \Theta(n^2)$ by Theorem (2). Hence, the time complexity remains the same and consistent to the analysis in Llemit[4].

5 Implementation and Results

We implemented the modified Sol - van der Weide algorithm on a desktop computer with an installed random access memory (RAM) of 2.0 GB and an Intel Core 2 Duo processors with speeds of 2.0 GHz and 1.99 GHz. The popular values for the transition probabilities were used and these are

$$p_u = \left(\frac{e^{(r-\gamma)\frac{\Delta t}{2}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}} \right)^2 \tag{12}$$

and

$$p_d = \left(\frac{e^{-\sigma\sqrt{\frac{\Delta t}{2}}} - e^{(r-\gamma)\frac{\Delta t}{2}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}} \right)^2 \tag{13}$$

where γ stands for the dividend yield and σ the volatility rate. As for the middle transition probability, as usual, we set it to $p_m = 1 - p_u - p_d$.

We use the following test values: $S = 5$ (stock price), $K = 3$ (strike price), $r = 0.15$ (risk-free interest rate), $T = 0.25$ (maturity), $\sigma = 0.5$, and $\gamma = 0.1$ (dividend yield rate). The corresponding Black-Scholes value is 1.993111420725652 and is obtained using the built-in *Matlab* command *blsprice*. We then plot the true error of the MSWA versus time-steps n.

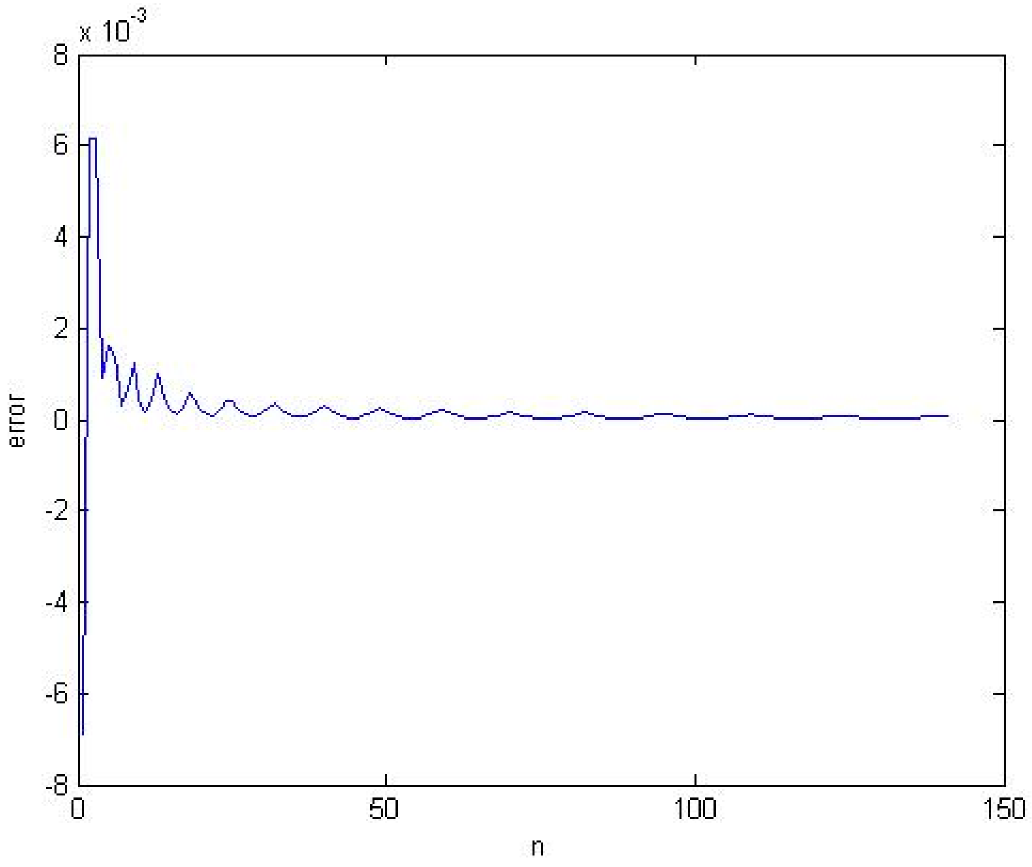

Figure 2: True Error of MSWA versus time steps n

Observe that the error tends to zero as the number of time steps n becomes large as shown by the graph. This confirms the convergence of the MSWA to the Black - Scholes and validates our modifications to the Sol - van der Weide Algorithm.

6 Conclusions

This paper had two goals. Firstly, it aimed to modifiy the Sol - van der Weide algorithm in order for it to be applicable to trinomial tree models. As we can see, the difference between the algorithm computations versus the

Black-Scholes price approches zero as we increase the time steps n. This confirms the validity of the modified Sol-van der Weide algorithm. Secondly, it intended to check whether the time complexity of the modified algorithm was affected by the modifications. From our time complexity analyses, we found that the time complexity remains to be $\Theta(n^2)$.

As further works, it would be interesting to determine the algorithm's space complexity and subsequently, whether it is optimizable in terms of the number of multiplications and additions. Next would be to modify the Sol - van der Weide algorithm to be applied to barrier options in a trinomial tree set-up.

References

[1] Black, F. and Scholes, M. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 1973.

[2] Boyle, P. Option Valuation Using a Three-Jump Process. *International Options Journal* , 1986.

[3] Cormen, T.H., Leiserson, C.E., Rivest, R., Stein, C. *Introduction to Algorithms, 3rd Ed.* The MIT Press, 2009.

[4] Llemit, D.G. On A Recursive Algorithm For Pricing Discrete Barrier Options. *Communications in Mathematical Finance(forthcoming)*, 2015.

[5] Rosen, K.H. *Discrete Mathematics and Its Applications, 6th Ed.* McGraw-Hill, 2008.

[6] Shreve, S.E. *Stochastic Calculus for Finance I: The Binomial Asset Pricing Model.* Springer Finance, 2005.

[7] Tina Sol. Pricing Barrier Options in Discrete Time. Bachelor Thesis, Technische Universiteit Delft, 2009.

Appendices

A Modified Sol - van der Weide Algorithm Code

```
function TRINOMIALCALLSWAprice(S, K, r, T, n, sigma, gamma)
%r = risk free rate
%gamma= yield rate
%sigma = volatility
%S=initial asset price
%K=strike price
%B=barrier level
%T=lifetime
%n=number of time partitions
h=T/n;
u=exp(sigma*sqrt(2*h));
d=1/u;

%transition probabilities
pu = (exp((r-gamma)*h/2)-exp(-sigma*sqrt(h/2)))^2/
(exp(sigma*sqrt(h/2))-exp(-sigma*sqrt(h/2)))^2;
pd = ((exp(sigma*sqrt(h/2)))-exp(((r-gamma)*h)/2))^2/
(exp(sigma*sqrt(h/2))-exp(-sigma*sqrt(h/2)))^2;
pm = 1-pu-pd;

%tic;
 Sf=S*u.^(n:-1:-n);
 price=(Sf-K).*(Sf>K);
 for i=1:1:n
     Sf=Sf(2:end-1);
     price=exp(-r*h)*(pu*price(1:end-2)+ pm*price(2:end-1)
+ pd*price(3:end));
 end
 price
%t2=toc
```