Some Important Optimizations of Binomial and Trinomial Option Pricing Models, Implemented in MATLAB

Juri Kandilarov, Slavi Georgiev

Abstract: *In this paper the well-known binomial and trinomial option pricing models are considered. First the assumptions on the market and the equations which the models are based on are presented. The heart of the paper consists of some crucial optimizations by memory and time, implemented on the programming code of MATLAB. Next, the trinomial model is briefly discussed and some important notes are made. Finally, the main ideas of the paper are summarized and commented.*

Key words: *binomial and trinomial option pricing models, European and American option, optimization, programming, MATLAB.*

INTRODUCTION

Ultimately it is the market which decides the value of an option. If we try to find a reasonable price of an option, we need first a mathematical model of the market. Here we will accept the model named after Black, Merton and Scholes, which is both successful and widely spread. Before talking on the mathematical side of the model itself, we will present the assumptions, which are made on the market:

- There are no arbitrage opportunities – this means that no riskless income is possible. This statement leads further to the assumption that all participants on the market have equal and immediate access to all information available at any time;
- The market is frictionless – this means that there are no transaction costs (fees or taxes), the interest rates for borrowing and lending money are equal, and all securities and credits are available at any time in any size. Consequently, all variables are perfectly divisible – they may take any real number. Further, individual trading would not influence the market;
- The asset price follows a geometric Brownian motion;
- r and σ are constants for $0 \le t \le T$. No dividends are paid in that time period.

The option is European. (It is possible some of the last assumptions to be loosened. For example, r and σ may be functions of t, and dividends could be incorporated in the model [2], but here this is out of our view.)

Before we start modelling, we ought to make some comments. We will consider the (S,t) domain on the half strip $0 \le t \le T,\ S > 0$, where T is the maturity of the option and S is the price of the underlying asset. With $V(S_t,t)$ we will denote the price of the option in time t and current asset price S_t at that time t.

Now let us present the considered algorithm in this paper, known as the binomial method due to Cox, Ross and Rubinstein, which is robust and widely applicable.

In practice we are often interested in the one value $V(S_0,0)$ of an option at current time and spot price. Then it is unnecessarily costly to calculate the surface $V(S,t)$ for the entire domain only to extract the required information for $V(S_0,0)$. This relatively small task of calculating $V(S_0,0)$ could be comfortably solved using the binomial model. This method is based on a tree-type grid applying appropriate binary rules at each grid point. The grid is not predefined but is constructed by the method [4].

Now we begin with discretizing the continuous time t by replacing it with equidistant time instances t_i. Let us introduce the notations: M: number of time steps; $\Delta t := {}^{T}/_{M},\ t_i := i\Delta t,\ i = 0,\dots,M,\ S_i := S_{t_i} := S(t_i)$.

So far the domain of the (S,t) half strip is semidiscretized in that it is replaced by parallel straight lines, leading to a discrete-time model. The next step of discretization replaces the continuous values S_i along the parallel $t = t_i$ by discrete values S_{ji} for all i and appropriate j.

Now we will assume the following rules, over which the binomial method is built:

1. The price S over each time period Δt can only have two possible outcomes – the value S could either evolve up to Su or down to Sd with $0 < d < u$. Here u is the factor of an upward movement and d is the factor of a downward movement. Very often, due to computational effectiveness and convenience, we use a recombining tree. This means that an upward movement, followed by a downward movement would result in the same price as a downward movement, followed by an upward movement.

2. The probability of an upward movement is p, i. e. $\mathbb{P}(\text{up}) = p$.

Due to the fact that we would like to derive equations for the so far undetermined parameters u, d and p, we will assume further

3. The expectation and variance of S refer to the continuous counterparts, evaluated for the risk-free interest rate r.

This assumptions lead to equations about u, d and p. The resulting probability $\mathbb{P}$ of 2. does not reflect the expectations of an individual in the market. Rather $\mathbb{P}$ is an artificial risk-neutral probability that matches 3. The expectation $\mathbb{E}$ used below refers to this probability [4].

The derivation of the equations will be given in very short:

First, let us recall the definition of the expectation for the discrete and continuous cases:

$$\mathbb{E}(S_{i+1}) = pS_i u + (1-p)S_i d, \quad \mathbb{E}(S_{i+1}) = S_i e^{r\Delta t}.$$

Equating gives

$$e^{r\Delta t} = pu + (1-p)d,$$

which leads to

$$p = \frac{e^{r\Delta t} - d}{u - d}.$$

Now let us equate the variances:

$$\mathrm{Var}(S_{i+1}) = S_i^2 e^{2r\Delta t}\left(e^{\sigma^2\Delta t} - 1\right),$$

$$\mathrm{Var}(S_{i+1}) = p(S_i u)^2 + (1-p)(S_i d)^2 - S_i^2(pu + (1-p)d)^2.$$

Applying the first equation, we derive the second:

$$e^{2r\Delta t + \sigma^2\Delta t} = pu^2 + (1-p)d^2.$$

Since we have two relations for the three unknown parameters, we are free to impose an arbitrary third equation. The classical example is the plausible equation

$$ud = 1,$$

which reflects a symmetry between upward and downward movement of the asset price.

Now the parameters u, d and p are fixed and they depend on r, σ and Δt. Next we analyze the grid.

The above rules are applied to each grid line $i = 0, \dots, M$, starting at $t_0 = 0$ with current spot price $S = S_0$. Doing it for the subsequent values of t_i builds the tree with values $Su^j d^k$, where $j + k = i$. In this way, specific discrete values S_{ji} of S_i are defined. As we mentioned before that the tree is recombining, after time period $2\Delta t$ the considered asset price could only take three values instead of four. It does not matter which of the two possible paths we take to reach Sud. Consequently, the defined binomial process is path independent [4]. Accordingly at the expiration time $T = M\Delta t$ the price S could take only the $(M+1)$ discrete values $Su^j d^{M-j}$, $j = 0, \dots, M$. The number of nodes in each time layer grows linearly in M, and the number of nodes in the tree – quadratically in M.

We will skip the solution of the equations and will declare that up to higher order terms,

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

RELATIONSHIP AND CONVERGENCE TO THE BLACK–SCHOLES MODEL

The main advantage of the binomial model is that it is highly applicable for pricing various types of options, including American type and other exotic options. This can be

interpret as an advantage to the Black–Scholes model. Actually, the two models have similar features, because both of them follow the same assumptions. As a result, for European options, the binomial model converges to on the Black–Scholes formula as the number of the time steps increases. In other words, the binomial model provides discrete approximation of to the continuous process underlying the Black–Scholes model.

However, this convergence is not smooth or uniform. That is why it is not recommendable to extrapolate the numerical results for different M to the limit $M \to \infty$. Here we can observe the convergence of the binomial model up to 150 time steps with the data, used in the following sample codes:

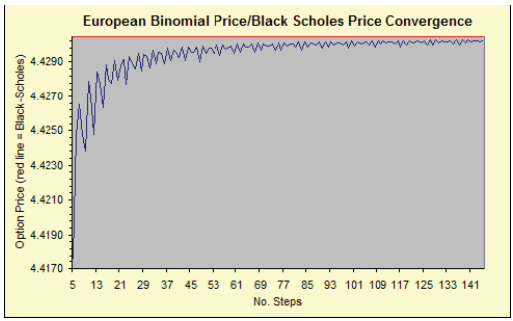

Figure 1. Convergence of binomial model to Black-Scholes model, using the sample data from the below

BINOMIAL MODEL, IMPLEMENTED IN MATLAB

Let us first consider the most intuitive (and equally naive) realization of the binomial method. This is essentially a program translation of the corresponding algorithm. Here we tackle European put option with the following parameters:

```
% Binomial method 1
% Non-optimized version
% Parameters initializing
S0 = 5; K = 10; T = 1; r = 0.06; sigma = 0.3; M = 256; dt = T/M;
u = exp(sigma * sqrt(dt)); d = 1/u; p = (exp(r*dt) - d) / (u - d);
% Asset prices at time T
for j = 1 : M+1
   S(j, M+1) = S0 * d^(M+1-j) * u^(j-1);
end
% Option prices at time T
for j = 1 : M+1
   V(j, M+1) = max(K - S(j, M+1), 0);
end
% Option price at time zero
for i = M : -1 : 1
   for j = 1 : i
      V(j, i) = exp(-r*dt) * (p * V(j+1, i+1) + (1-p) * V(j, i+1));
   end
end
% Option price
V(1, 1)
```

Let us examine the code. First we can improve the memory consumption. In the program we use one $(M+1)$-by-$(M+1)$ matrix for the asset prices and one same-sized matrix for the option prices. Considering an European option, we are interested only in the asset prices at the last time layer, at maturity T. When we trace the tree backwards, at a single step we use only the current and the previous layer. So it is quite possible to use

two single-dimensioned arrays (vectors) instead of a matrix – for the asset prices and the option prices. Right now we reduced the complexity by memory from quadratic to linear. Despite that, some additional enhancements are possible.

As we can notice, the corrections through time layers are made not simultaneously at all nodes but node by node. So it is advisable to use only one vector instead of two. The option price could be obtained only with one vector with $(M + 1)$ elements.

But we can go even further. Note however that the following optimization is applicable only for European-typed options. Later the case of Americans will be discussed. It is noticeable that the only time we operate with both S and V is the calculating of the payoffs at time T and we can integrate this operation into a single vector V.

There is a little bit more work we can do. As the initializing loop is upward, on every iteration the vector grows by one element. Since the memory allocation is comparatively slow operation, we can consider preallocating the memory for the vector. The MathWorks' official suggestion `V = zeros(1, M+1)` seems to be not the most efficient [6]: we use `V(M+1) = 0`. The reason for this is simple – the second variant only allocates the memory, without worrying about the internal values, which we are not interested in now.

The resulting code is presented:

```
% Binomial method 2
% Optimized by memory version
% Parameters initializing
S0 = 5; K = 10; T = 1; r = 0.06; sigma = 0.3; M = 256; dt = T/M;
u = exp(sigma * sqrt(dt)); d = 1/u; p = (exp(r*dt) - d) / (u - d);
% Asset prices at time T
V(M+1) = 0;
for j = 1 : M+1
   V(j) = S0 * d^(M+1-j) * u^(j-1);
end
% Option prices at time T
for j = 1 : M+1
   V(j) = max(K - V(j), 0);
end
% Option price at time zero
for i = M : -1 : 1
   for j = 1 : i
      V(j) = exp(-r*dt) * (p * V(j+1) + (1-p) * V(j));
   end
end
% Option price
V(1)
```

Now we will introduce more significant changes. Due to the interpretative behavior of MATLAB, some optimizations are done by hand as MATLAB does not optimize the intermediate code. Some of them are rather trivial, but the most important actions are to make use of MATLAB's built-in functions as much as possible, because they are fully compiled and designed to run extremely efficiently [3].

To begin with, let us introduce the colon notation. Fortunately, the basic operators work pretty well with vectors and matrices as well as with scalars, so the initializing loop is replaced by `V = S0 * d.^(M:-1:0) .* u.^(0:M)`. What is more, we can optimize the second loop in the same manner, because scalars and vectors are naturally compatible in case of arithmetic operations between them. So it follows that the option values V at time T could be computed in such a way: `V = max(K - S0 * d.^(M:-1:0) .* u.^(0:M), 0)`.

However, in spite of our efforts so far, it appears that we haven't gained much profit. This is because the complexity of the main loop is still $O(M^2)$.

We can speed up the nested loop by removing unnecessary computations. The discounting factor $e^{-r\Delta t}$ at each time step could be deferred until the end, where it accumulates to a single e^{-rT} factor. Furthermore, we can precompute $q = 1 - p$ and get it

out of the loops. But the complexity remains unchanged; therefore we will try to remove the inner loop using the aforementioned colon notation. Note that at every iteration the length of V decreases by one; in the end V is a scalar [3]. The resulting code is here:

```
% Binomial method 3
% Vectorized version
% Parameters initializing
S0 = 5; K = 10; T = 1; r = 0.06; sigma = 0.3; M = 256; dt = T/M;
u = exp(sigma * sqrt(dt)); d = 1/u; p = (exp(r*dt) - d) / (u - d);
% Option prices at time T
V = max(K - S0 * d.^(M:-1:0) .* u.^(0:M), 0);
% Option price at time zero
q = 1 - p;
for i = M : -1 : 1
   V = p * V(2:i+1) + q * V(1:i);
end
V = exp(-r*T) * V;
% Option price
V
```

Here we can see how the execution time changes according to the applied optimizations:

Table 1. **Execution time of the three proposed variants, estimated by averaging over 1000 runs with MATLAB R2014a on a 2.3GHz dual-core x64 3GB 1333MHz DDR3 mobile computer**

Name of code version	M	Execution time (sec.)	Ratio
Binomial model 1 – Non-optimized	512	0.525	-
Binomial model 2 – Partially optimized	512	0.0084	62.5
Binomial model 3 – Fully optimized	512	0.0046	1.8
Binomial model 1 – Non-optimized	1024	4.735	-
Binomial model 2 – Partially optimized	1024	0.0326	145
Binomial model 3 – Fully optimized	1024	0.0128	2.5
Binomial model 1 – Non-optimized	2048	40.25	-
Binomial model 2 – Partially optimized	2048	0.130	308
Binomial model 3 – Fully optimized	2048	0.036	3.6

It can be seen that considering the exposed optimization, the overall progress is expressed in decreasing the computational time about 115 times for $M = 512$, about 370 times for $M = 1024$ and about 1120 times for $M = 2048$. Here the ratio, corresponding to the vectorization, is not dramatically high, but when increasing M, an impressive performance leap will be observed.

FURTHER OPTIMIZATION

Following the mathematical model, the achieved results are far a good deal. If we would like to optimize our program even further, we should change the approach. The algorithm is pretty clear – we have to do some numerical transformation in the main loop. In [3] using a binomial expansion is proposed. We can replace the vector arithmetic with formation of a single sum using binomial coefficients. This can be done with $O(M)$ floating point operations. The binomial formula could not be applied straightforward – some additional numerical experiments are required. In order to avoid overflows and underflows,

logarithms may be used for the computational purposes. Finally, due to the fact that some option values at $t = T$ equal zero, the redundant computations should also be avoided to improve performance [3].

PRICING AN AMERICAN OPTION

So far we have investigated how to find fair price of an European put option. The case of American option is a bit more complicated. An American option differs from an European by the right of exercising it at each time before the maturity $0 \leq t \leq T$. This feature is incorporated into the model by comparing the discounted expectation (the same as European) with the option payoffs at every discrete time layer $t_{0 \leq i \leq M}$ and taking the greater value into account. Because of this speciality, we are not able to apply some of the optimizations discussed before. Our approach however will be to precalculate some expressions in order to reuse them in the main loop. The code is exposed:

```
% Binomial method for American put option
% Optimized version
% Parameters initializing
S0 = 9; K = 10; T = 1; r = 0.06; sigma = 0.3; M = 256; dt = T/M;
u = exp(sigma * sqrt(dt)); d = 1/u; p = (exp(r*dt) - d) / (u - d);
% Precomputations
dpow = d.^(M:-1:0);
upow = u.^(0:M);
% Option prices at time T
V = max(K - S0 * dpow .* upow, 0);
% Option price at time zero
q = 1 - p; e = exp(-r*dt);
ep = e * p; eq = e * q;
for i = M : -1 : 1
   V = ep * V(2:i+1) + eq * V(1:i);
   Si = S0 * dpow(M-i+2:M+1) .* upow(1:i);
   V = max(max(K - Si, 0), V);
end
% Option price
V
```

As we can see, the precomputed power vectors of u and d are used on each loop iteration as well as at the initializing stage. Also some coefficients are prepared in advance, but the most significant change is the way forming the option prices vector V at each time layer. We continue to use the colon notation to access subvectors, hence the storage requirements remain linear in M.

TRINOMIAL OPTION PRICING MODEL

Trinomial tree appears to be a natural generalization of binomial tree [5]. It is used for its better accuracy and speed of convergence. It is seen as a more advanced model [1]. In a trinomial model we consider three stock price developments: in one period the price increases by a factor of u with the probability p_u, decreases by a factor of d with the probability p_d, or remains unchanged with the probability $p_s = 1 - p_u - p_d$. Here we keep the important property $ud = 1$, which leads to a recombining tree and the other parameters are as follows:

$$u = e^{\sigma\sqrt{2\Delta t}}, \quad d = e^{-\sigma\sqrt{2\Delta t}},$$

$$p_u = \left(\frac{e^{r\frac{\Delta t}{2}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}} \right)^2, \qquad p_d = \left(\frac{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{r\frac{\Delta t}{2}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}} \right)^2, \qquad p_s = 1 - p_u - p_d.$$

Here we have $2 * i - 1$ nodes in the i-th time layer and that is why the initialization looks like a bit different. In order to keep the code clear and simple, we consider $d = u^{-1}$

and evaluate the option prices again with a single command, using colon notation. The idea of the algorithm afterwards is quite similar to the discussed so far. The code follows:

```
% Trinomial method for European put option
% Vectorized version
% Parameters initializing
S0 = 5; K = 10; T = 1; r = 0.06; sigma = 0.3; M = 256; dt = T/M;
u = exp(sigma * sqrt(2*dt));
pd = ((exp(sigma*sqrt(dt/2))-exp(r*dt/2))/(exp(sigma*sqrt(dt/2))-exp(-sigma*sqrt(dt/2))))^2;
pu = ((exp(r*dt/2)-exp(-sigma*sqrt(dt/2)))/(exp(sigma*sqrt(dt/2))-exp(-sigma*sqrt(dt/2))))^2;
ps = 1 - pu - pd;
% Option prices at time T
V = max(K - S0 * u.^(-M:M), 0);
% Option price at time zero
for i = M : -1 : 1
   V = pu * V(3:2*i+1) + ps * V(2:2*i) + pd * V(1:2*i-1);
end
V = exp(-r*T) * V;
% Option price
V
```

In the trinomial model implementation we may confidently make use of all results achieved so far. As described before, besides some superficial peculiarities, the program flow stays the same. We use again the colon notation in initializing and computing option prices. The output is equal to the corresponding binomial model realization except that here we benefit an improved convergence.

CONCLUSION

In this paper we have proposed some important and useful optimizations of the MATLAB code implementation. The binomial method, and its powerful extension the trinomial method, are highly applicable for numerous types of options and that is why they are widely used for option valuating. On the other hand, the main drawback of the model is its relatively slow speed. Even for the fastest contemporary computational units the binomial model may be a challenge, so the algorithm optimization is a matter of a crucial importance.

REFERENCES

[1] Clifford, P., O. Zaboronski, *Pricing Options Using Trinomial Trees*, (2008), {http://www2.warwick.ac.uk/fac/sci/maths/people/staff/oleg_zaboronski/fm/, Aug. 2015}

[2] Cox, J., S. Ross, M. Rubinstein, *Option pricing: A Simplified Approach*, Journal of Financial Economics 7, 229-263, (1979)

[3] Higham, D. J., *Nine Ways to Implement the Binomial Method for Option Valuation in MATLAB*, SIAM Review, Vol. 44, No. 4, 661-677, (2002)

[4] Seydel, R. U., *Tools for Computational Finance*, Springer-Verlag, (2009)

[5] Радков, П., *Биномен модел на финансов пазар. Намиране на цената на опция посредством биномни и триномни мрежи*, (2014), {http://video-e.com/math/, Aug. 2015}

[6] http://undocumentedmatlab.com/blog/preallocation-performance, {Aug. 2015}

ABOUT THE AUTHORS

Assoc. Prof. Juri Dimitrov Kandilarov, PhD, Department of Mathematics, University of Rousse, Phone: +359 82 888 634, e-mail: ukandilarov@uni-ruse.bg.

Slavi Georgiev Georgiev, Third year Bachelor student, Financial Mathematics, University of Rousse, e-mail: georgiev.slavi.94@gmail.com.