J. Statist. Comput. Simul., 2003, Vol. 00, pp. 1–11

Taylor & Francis
Taylor & Francis Group

A MEMORY REDUCTION METHOD IN PRICING AMERICAN OPTIONS

RAYMOND H. CHAN*, YONG CHEN† and K. M. YEUNG‡

Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong

(Received 16 September 2002; In final form 15 June 2003)

This paper concerns with the pricing of American options by simulation methods. In the traditional methods, in order to determine when to exercise, we have to store the simulated asset prices at all time steps on all paths. If N time steps and M paths are used, then the storage requirement is O(MN). In this paper, we present a simulation method for pricing American options where the number of storage required only grows like O(M). The only additional computational cost is that we have to generate each random number twice instead of once. For machines with limited memory, we can now use a larger N to improve the accuracy in pricing the options.

Keywords: Option pricing; Monte Carlo method; Random number generator

1 INTRODUCTION

Pricing options by Monte Carlo simulations was first introduced in Boyle (1977). An advantage of Monte Carlo simulations is that the convergence rate is independent of the number of state variables. Path dependency can also be handled easily by Monte Carlo simulations. However there is a general belief that Monte Carlo simulations can be used efficiently only for European-style options. The apparent difficulties in using Monte Carlo simulations to price American-style options come from the backward nature of the early exercise feature. There is no way of knowing whether early exercise is optimal when a particular asset price is reached at a given time.

The first work on pricing American-style options by Monte Carlo simulations is due to Tilley (1993). Tilley's algorithm is computationally inefficient because it requires the storage of all asset prices at all simulation times for all simulated paths. Thus the total storage requirement grows like O(MN) where M is the number of simulated paths and N is the number of time steps. The accuracy of the simulation is hence severely limited by the storage requirement. Moreover, in Monte Carlo methods, continuous exercising is replaced by discrete exercising, and that produces biases. To get higher accuracy, one can increase the number of time steps N. But then larger N means more memory.

* Corresponding author. E-mail: rchan@math.cuhk.edu.hk
† E-mail: ychen@math.cuhk.edu.hk
‡ E-mail: kmyeung@math.cuhk.edu.hk

ISSN 0094-9655 print; ISSN 0000-0000 online
DOI: 10.1080/00949650310001597876

The main problem of traditional Monte Carlo methods in pricing American options is that the simulated paths are all generated in the time-increasing direction, *i.e.*, they start from the initial asset price S_0 and march to the final time according to the given geometric Brownian motion. Since the pricing of American options is a backward process, one has to save all the intermediate asset prices. In this paper, we propose a simulation method that does not require storing of all the intermediate asset prices. The storage is therefore independent of N and grows only like O(M), where M is the number of paths. Thus the storage is significantly reduced, and we can increase N to improve the accuracy of our results. Our main idea is to generate the paths in the time-decreasing direction that still follow the given geometric Brownian motion starting from S_0. The only additional cost in our method is that we have to generate each random number twice instead of once. We will see that the resulting computational cost is at most four times of that of the traditional methods.

The remainder of this paper is organized as follows. Section 2 presents our method of simulating the sample paths. We give a detailed discussion of our memory reduction method in Section 3. Section 4 gives some numerical results to illustrate the effectiveness of our method. Finally, some conclusions are given in Section 5.

2 THE SIMULATION METHOD

In traditional simulation methods for pricing options, the simulated paths are generated in the forward direction, *i.e.* follow the direction of the time. The main idea of our method is to generate the simulated paths in the backward (*i.e.* time-decreasing) direction. In this section, we first recall how forward paths are generated. Then we introduce our method of generating backward simulated paths. These paths still follow the given geometric Brownian motion. For simplicity, we will use the MATLAB language to illustrate how the codes are to be written. The corresponding codes for FORTRAN 90 are given in the Appendix.

As usual, we let the asset price S follow a geometric Brownian motion

$$\frac{\mathrm{d}S}{S} = r\,\mathrm{d}t + \sigma\,\mathrm{d}X \tag{1}$$

with drift r and variance σ in the risk-neutral world. By Ito's Lemma, we have

$$S(t+\mathrm{d}t) = S(t)e^{(r-(1/2)\sigma^2)\,\mathrm{d}t+\sigma\,\mathrm{d}X} = S(t)e^{(r-(1/2)\sigma^2)\,\mathrm{d}t+\sigma\varepsilon\sqrt{\mathrm{d}t}}, \tag{2}$$

where ε is a standard normal random variable, see for instance Kwok (1998, p. 225). In the path simulation, we divide the time horizon into N time steps with each having the length

$$\Delta t = \frac{T-t_0}{N},$$

where t_0 is the current time and T is the final time. Thus the time horizon is discretized as $t_0 < t_1 < \cdots < t_N = T$. Let the asset price at time t_0 be $S(t_0) = S_0$.

2.1 The Forward Paths

By (2), we can simulate a path as follows:

$$\begin{aligned} S_i &:= S(t_i) = S(t_{i-1} + \Delta t) \\ &= S(t_{i-1}) e^{(r-(1/2)\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\varepsilon_i} \\ &= S_{i-1} e^{(r-(1/2)\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\varepsilon_i}, \quad i = 1, 2, \ldots, N, \end{aligned} \tag{3}$$

where $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N$ are independent identically distributed standard normal random numbers. We will call such a path $S_0 \to S_1 \to \cdots \to S_n$ a *forward path* since it is generated in the time-increasing direction.

To generate a forward path in the computer, we need to generate N random numbers. Most computer languages already have built-in functions to generate them. (In MATLAB, it is "randn".) Moreover, by using the concept of a *seed*, one has the flexibility to change or fix the sequence of random numbers each time they are generated. For example, in MATLAB, the commands

```
randn ('seed',d);
e = randn;
```

give a different random number e each time the seed d is changed, but give the same random number if d is fixed.

Notice that from (3), the intermediate asset prices can be written as

$$S_i = S_0 e^{i(r-(1/2)\sigma^2)\Delta t + \sigma\sqrt{\Delta t}(\varepsilon_1 + \varepsilon_2 + \cdots + \varepsilon_i)}, \quad i = 1, 2, \ldots, N. \tag{4}$$

2.2 The Backward Paths

The evaluation of American-style options is a backward process starting from the expiry date T back to the current time t_0. It is thus natural to ask if we can generate a path in the time-decreasing direction which still follows the geometric Brownian motion (1) starting from S_0. The answer is yes. Given S_0, define

$$\begin{aligned} S_1 &= S_0 e^{(r-(1/2)\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\varepsilon_N}, \\ &\vdots \\ S_i &= S_0 e^{i(r-(1/2)\sigma^2)\Delta t + \sigma\sqrt{\Delta t}(\varepsilon_N + \varepsilon_{N-1} + \cdots + \varepsilon_{N-i+1})}, \\ &\vdots \\ S_N &= S_0 e^{N(r-(1/2)\sigma^2)\Delta t + \sigma\sqrt{\Delta t}(\varepsilon_N + \varepsilon_{N-1} + \cdots + \varepsilon_1)}. \end{aligned} \tag{5}$$

Here $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N$ are independent identically distributed standard normal random numbers. Comparing (4) with (5), and notice that $\varepsilon_N, \varepsilon_{N-1}, \ldots, \varepsilon_1$ are also independent identically distributed standard normal random numbers, the sequence $S_0, S_1, \ldots, S_N$ therefore still represents a path that follows the geometric Brownian motion (1) starting from S_0. From (5), we can write

$$S_i = f(i, \omega_i), \quad i = 1, 2, \ldots, N, \tag{6}$$

where f is a fixed function depending only on the parameters S_0, σ, r, and Δt, and $\omega_i = \varepsilon_N + \varepsilon_{N-1} + \cdots + \varepsilon_{N-i+1}$.

In this setting, it is easy to generate the path $S_0 \to S_N$ in the backward direction, *i.e.* from S_N to S_0. Again, we use MATLAB commands to illustrate this. To emphasize that we need not store the intermediate values $S_N, \ldots, S_i, \ldots, S_1$, we drop the subscript i in the MATLAB codes.

ALGORITHM 1

1. Initialization:
   (a) Set the seed of the path to any given positive integer, say d.
   randn ('seed', d);
   (b) Generate the random numbers $\{\varepsilon_i\}_{i=1}^N$ and compute their sum ω_N.
   omega = 0;
   for i = 1:N,
   omega = omega + randn;
   end;
   (c) Compute the asset price S_N at the expiry date T using (6).
   S = f(N, omega);
2. For $i = N, \ldots, 1$, generate S_{i-1} from S_i:
   (a) Set the seed to d.
   randn ('seed', d);
   (b) Extract ε_{N-i+1} and compute $\omega_{i-1} = \omega_i - \varepsilon_{N-i+1}$.
   omega = omega − randn;
   (c) Compute S_{i-1} by using (6).
   S = f(i − 1, omega);
   (d) Extract the new seed and set it to d (see the explanation below).
   d = randn ('seed');

We note that the most important step in the whole algorithm is Step 2(d). It is a command that extracts the seed corresponding to the next random number in the sequence $\{\varepsilon_i\}_{i=1}^N$. With the command, we can generate ε_i without having to generate the whole sequence $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{i-1}$. In essense, we generate $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_i$ as follows:

$$\mathrm{d} = d_1 \to \varepsilon_1 \to d_2 \to \varepsilon_2 \to \cdots \to \varepsilon_{i-1} \to d_i \to \varepsilon_i, \tag{7}$$

where d_i is the *current seed* corresponding to ε_{i-1} and is obtained from Step 2(d). We remark that there is also such a command to compute the current seed in FORTRAN 90, see Appendix. We will call the path generated by our method a *backward path*, as it is generated in the time-decreasing direction.

Figure 1 shows a forward path with its associated backward path. Figure 2 shows 10 forward paths and 10 backward paths. Since both forward and backward paths follow the same geometric Brownian motion (1), we can use either one to price options. The advantage of using backward paths is that the direction of the paths is the same as the direction of pricing the options, and hence storage can be reduced as we will see in the next section.

We emphasize again that in Algorithm 1, we do not need to store the whole random number sequence $\{\varepsilon_i\}_{i=1}^N$ or the intermediate asset prices $\{S_i\}_{i=1}^N$. All we need is to store the current seed d, the sum omega ($=\omega_i$), and the current asset price S($=S_i$). Thus the total storage is 3 variables per path.

Regarding the computational cost, we need to generate each ε_i twice: one in Step 1(b) and one in Step 2(b). For this, a total of $4N$ calls involving "randn" is needed in Steps 1(b), 2(a),

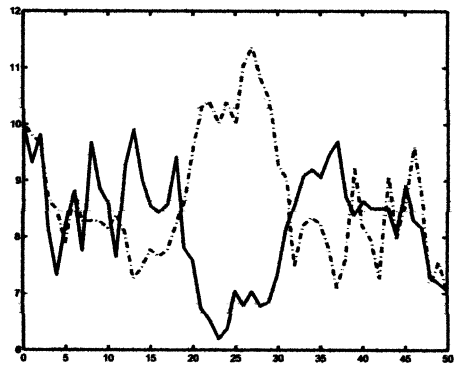

FIGURE 1 A forward path (solid) and its associated backward path (dash).

2(b), and 2(d). To see the timing of these function calls, we tried on our Pentium II PC one million calls of

$$\text{omega} = \text{omega} + \text{randn}, \quad \text{randn('seed', i)} \quad \text{and} \quad \text{d} = \text{randn('seed')}.$$

They require 25.23, 25.19 and 24.88 CPU seconds respectively. Thus the cost of all four steps are roughly the same.

Recall that in generating a forward path, we need to generate N random numbers (see (3)), a cost comparable to Step 1(b). Thus the computational cost of generating a backward path is at most four times that of a forward path. It is an overestimate as we have not counted the cost of computing S_i in (3) and (5). In Table I, we give the CPU times in generating M paths where each path has N time steps. We see that the cost of generating a backward path is about 3.3 times of that of a forward path.

We end this section by pointing out that like the generation of forward paths, the backward path generation is well-adapted to parallel computations. There is no communication between the paths when they are generated. Moreover, the method can be applied to any pseudo random number generators (including those in Quasi Monte Carlo methods) provided that the generators have the following properties: (i) the same pseudo random number sequence $\{\varepsilon_i\}_{i=1}^N$ can be generated if the seed d is fixed, and (ii) we can generate ε_{i+1} using ε_i only, and there is no need to store d and all the preceding random numbers $\{\varepsilon_j\}_{j=1}^{i-1}$ (see (7)).

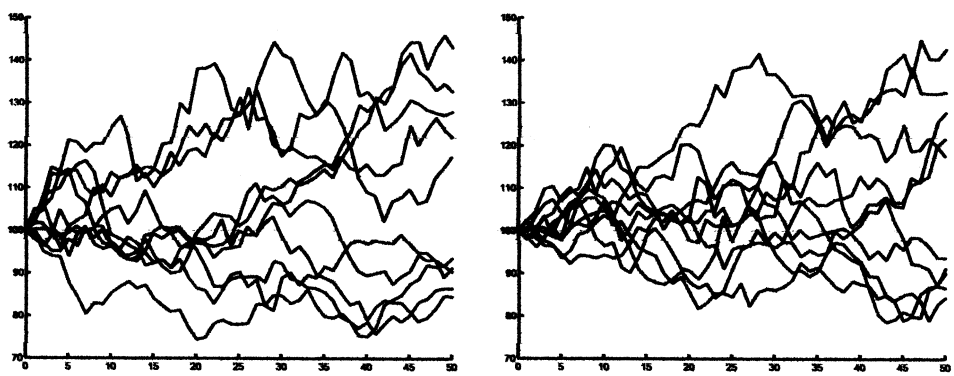

FIGURE 2 Simulations of forward paths (left) and backward paths (right).

TABLE I CPU Time in Seconds for Generating Backward/Forward Paths.

	N		
M	*10*	*100*	*1000*
100	0.11/0.05	1.10/0.38	11.20/3.46
1000	1.21/0.39	11.37/3.46	112.55/34.22
10,000	12.58/3.79	113.69/34.55	1139.65/342.95

3 THE MEMORY REDUCTION METHOD

In this section, we present our memory reduction method. It is a method that can reduce the memory requirement of simulation methods for pricing American options. The main idea is to replace the forward path simulation in a given method by the backward path simulation. For simplicity, we explain our method by using the simple but powerful method given in Longstaff and Schwartz (2001).

At the final exercise date, the optimal exercise strategy for an American option is to exercise the option if it is in the money. Prior to the final date, however, the optimal strategy is to compare the immediate exercise value with the expected cash flow from continuing, and then exercise if immediate exercise is more valuable. Thus, the key to optimally exercising an American option is to identify the conditional expected value of continuation. In Longstaff and Schwartz (2001), the cross-sectional information in the simulated paths is used to identify the conditional expectation function. This is done by regressing the cash flows from continuation on a set of basis functions depending on the current asset prices. The fitted function from this regression is an efficient unbiased estimate of the conditional expectation function, and by which, one can estimate an optimal stopping rule for the option.

The Longstaff–Schwartz method uses the forward paths and hence requires the storage of all the intermediate asset prices on every path. Thus the storage requirement is O(NM), where M is the number of paths, and N is the number of time steps. Moreover, like other methods for pricing American options, the method has biases, and it is difficult to obtain high accuracy without using large M and N. But by replacing the forward paths with our backward paths, we now show that we can reduce the memory requirement to $7M$.

As in Longstaff and Schwartz (2001), we illustrate our method by using a numerical example. Consider an American put option on a share of non-dividend-paying stock. The strike price E and the underlying asset price S_0 are both equal to \$10, the riskless rate r is 0.1, the volatility σ is 0.4, and the time to expiration T is 0.5 year. We set the number of time step $N = 5$, *i.e.* the option is exercisable at time $i = 1, 2, 3, 4$ and 5. We illustrate the algorithm by using ten paths, *i.e.* $M = 10$.

ALGORITHM 2

1. Initialization:

   (a) Using Step 1 of Algorithm 1 in Section 2, compute S(j) ($=S_5(j)$) and omega(j) ($=\omega_5(j)$) for each path j, $1 \le j \le M$. The seed d(j) for each path j can be set arbitrarily. For example, we can set d(1) to be the internal seed of the computer, and d(j), $2 \le j \le M$, to be the current seed corresponding to the last random number of the previous path, *i.e.* d(j) $= d_{N+1}(j-1)$ in (7).

(b) Compute the cash flow $P_5(j) = \max\{E - S_5(j), 0\}$ for each path j. It is the cash flow realized by the option holder conditional on not exercising the option before the final expiration date $i = 5$.

Path j	$S_5(j)$	$P_5(j) = max\{E - S_5(j), 0\}$
1	9.21246571	0.78753429
2	12.96684106	0.00000000
3	8.74786037	1.25213963
4	10.00266689	0.00000000
5	14.02305154	0.00000000
6	8.30637541	1.69362459
7	11.05985986	0.00000000
8	9.73994528	0.26005472
9	9.55956729	0.44043271
10	8.40500787	1.59499213

2. Backward time-marching to $i = 4$:
   (a) Using Step 2 in Algorithm 1, backward time-marching omega(j), S(j), and d(j) to time $i = 4$ for each path j. Notice that the variables omega(j), S(j), and d(j) will overwrite themselves. For example, memory location of $S_5(j)$ will be overwritten by $S_4(j)$.
   (b) Compute if the option is in the money.

Path j	$S_4(j)$	$max\{E - S_4(j), 0\}$
1	9.16327983	0.83672017
2	13.45449187	0.00000000
3	8.99170659	1.00829341
4	10.13840157	0.00000000
5	12.80505684	0.00000000
6	8.36260832	1.63739168
7	9.91977944	0.08022056
8	9.73050211	0.26949789
9	11.42108139	0.00000000
10	9.30440362	0.69559638

   (c) If the option is in the money, decide whether to exercise it immediately or to continue to hold it until the next exercisable time, *i.e.* hold on till $i = 5$. There are six paths ($j = 1, 3, 6, 7, 8, 10$) for which the option is in the money. Let X be the vector containing these asset prices $S_4(j)$, and Y be the vector containing the cash flows received at time $i = 5$, but discounted back to $i = 4$, *i.e.* $Y(j) = \rho P_5(j)$, where ρ is the discounted factor given by $\rho = e^{-r\Delta t} = 0.99004983374917$.

Path j	$X = S_4(j)$	$Y = \rho P_5(j)$
1	9.16327983	0.77969819288627
3	8.99170659	1.23968063251225
6	8.36260832	1.67677274376300
7	9.91977944	0.00000000000000
8	9.73050211	0.25746713230169
10	9.30440362	1.57912169313773

   (d) Use the least squares approach in Longstaff and Schwartz (2001) to estimate the expected cash flow from continuing to hold the option conditional on the asset

price $S_4(j)$. More precisely, regress Y on 1, X, and X^2. The resulting conditional expectation function is

$$E[Y|X] = -41.89780752481383 + 10.47643636927008X - 0.63030372995672X^2.$$

(e) Using the function $E[Y|X]$, compute the value of immediate exercising and the value from continuation at time $i = 4$.

Path j	$X = S_4(j)$	*Exercising* $\max\{E - S_4(j), 0\}$	*Continuation* $E[Y\|S_4(j)]$
1	9.16327983	0.83672017	1.17681838738995
3	8.99170659	1.00829341	1.34268154790032
6	8.36260832	1.63739168	1.63335832189242
7	9.91977944	0.08022056	0.00296772591145
8	9.73050211	0.26949789	0.36433778097917
10	9.30440362	0.69559638	1.01257663052763

(f) From the table, decide whether to exercise the option immediately ($j = 6, 7$) or continue to hold it ($j = 1, 3, 8, 10$). Then determine the current cash flow $C_4(j)$ conditional on not exercising prior to the time $i = 4$, *i.e.*

$$C_4(j) = \begin{cases} \max\{E - S_4(j), 0\}, & \max\{E - S_4(j), 0\} \geq E[Y|S_4(j)], \\ 0, & \text{otherwise.} \end{cases}$$

Because the option can be exercised only one time during its life, non-zero cash flow appears at most one time for each path. So, the cash flows $P_5(6)$ and $P_5(7)$ are reset to zero. See the table below.

(g) Finally, compute the present value of the cash flow $P_4(j)$ at time $i = 4$ for all paths. It is given by $P_4(j) = C_4(j) + \rho P_5(j)$.

Path j	$C_4(j)$	$P_5(j)$	$P_4(j) = C_4(j) + \rho P_5(j)$
1	0.00000000	0.78753429	0.77969819288627
2	0.00000000	0.00000000	0.00000000000000
3	0.00000000	1.25213963	1.23968063251225
4	0.00000000	0.00000000	0.00000000000000
5	0.00000000	0.00000000	0.00000000000000
6	1.63739168	0.00000000	1.63739168000000
7	0.08022056	0.00000000	0.08022056000000
8	0.00000000	0.26005472	0.25746713230169
9	0.00000000	0.44043271	0.43605033131320
10	0.00000000	1.59499213	1.57912169313773

(h) Go back to Step 2(a) and backward time-marching to $i = 3$ etc.

In essense, given $S_{i+1}(j)$ and $P_{i+1}(j)$, the algorithm first computes $S_i(j)$ using Step 2 in Algorithm 1. Then it computes $E[Y|X]$ by regressing $\rho P_{i+1}(j)$ on $S_i(j)$. With $E[Y|X]$, it computes the current cash flow $C_i(j)$, and finally the present value of the cash flow $P_i(j) = C_i(j) + \rho P_{i+1}(j)$.

To complete the numerical example, we give $P_3(j)$, $P_2(j)$ and $C_1(j)$ in the following table.

Path j	$P_3(j)$	$P_2(j)$	$C_1(j)$
1	0.6934161465	0.6865165406	0.0000000000
2	0.0000000000	0.0000000000	1.2827721948
3	0.0000000000	0.0000000000	1.8900363784
4	0.0000000000	0.0000000000	0.1908549358
5	0.0000000000	0.0000000000	0.7185000764
6	1.6210993631	1.6049691550	0.0000000000
7	0.0794223516	0.0786320860	0.0000000000
8	1.0048647352	0.9948661640	0.0000000000
9	0.4317115566	0.4274159549	0.0000000000
10	1.0857774653	1.0749737990	0.0000000000

From the table, we can compute the present value of the cash flow at $i = 1$: $P_1(j) = C_1(j) + \rho P_2(j)$. Since we do not exercise at $i = 0$, the option can now be valued by averaging the present value of the cash flow at time 0 on all paths, *i.e.* by averaging $P_0(j) = \rho P_1(j)$ on all paths j. For this example, it will be \$0.8813.

Let us now calculate the memory requirement of our method. Let the time to expiration be divided into N time steps, and M paths are taken. In traditional methods such as the Longstaff–Schwartz method where forward paths are used, at least MN memory are needed to store $S_i(j)$ for $i = 1, \ldots, N$ and $j = 1, \ldots, M$. (In fact, in Longstaff and Schwartz (2001), instead of computing $P_i(j)$, all current cash flows $C_i(j)$ are stored. Hence the memory requirement is $2MN$.) However, in our method, we just need to store 7 vectors with M entries: d(j), $\omega_i(j)$, $S_i(j)$, $P_i(j)$, X, Y, and $C_{i-1}(j)$, for $j = 1, \ldots, M$. All these vectors can be overwritten by themselves at the next time step, *i.e.* at time $i - 1$. Thus the memory requirement is independent on N and grows only like $7M$.

Regarding the computational cost, we see that the additional cost in our method is the generation of the backward paths in Step 2(a) of Algorithm 2. As seen in Section 2, this is at most four times the cost of generating the forward paths. Therefore, the computational cost of our memory reduction method is at most four times that of the Longstaff–Schwartz method. Again it is an overestimate as we have not counted the cost of all other computations in the algorithm, such as Steps 2(b)–(g). The timing should even be more favorable for our method if M and N are so large that out-of-core memory has to be used in storing the intermediate asset prices in the traditional methods.

4 NUMERICAL RESULTS

In this section, we test our memory reduction method on an example given in Wilmott *et al.* (1998, p. 176). It is an American put option with strike price $E = \$10$, the riskless rate $r = 0.1$, the volatility $\sigma = 0.4$, and the expiration date $T = 6$ months. We emphasize that since the forward and backward paths both satisfy the same geometric Brownian motion (1), the results obtained by our method should statistically be the same as those obtained by the method given in Longstaff and Schwartz (2001). However, here we would like to illustrate that with less stringent requirement on the memory, our method can provide better accuracy. All our computations were done by FORTRAN 90 on a SGI Origin 3200 machine with 16 Gigabyte RAM.

Again we use M and N to denote the number of paths and the number of time steps respectively. In pricing American options by simulations, there are two causes for the errors: (a) the

TABLE II Memory Reduction Method with $M = 10^4$ and $N = 100$.

S_0	*CNM*	*Value*	*Mean*	*STD*	*Error*
2	8.0000	7.9954	7.9951	0.0005	0.0049
4	6.0000	5.9970	5.9949	0.0012	0.0051
6	4.0000	3.9946	3.9950	0.0019	0.0050
8	2.0951	2.1104	2.0944	0.0112	0.0007
10	0.9211	0.9167	0.9212	0.0116	−0.0001
12	0.3622	0.3679	0.3654	0.0078	−0.0032
14	0.1320	0.1356	0.1342	0.0046	−0.0022
16	0.0460	0.0410	0.0476	0.0021	−0.0016

number of sample paths M is finite, and (b) continuous exercisable strategy is replaced by discrete one, *i.e.* N is finite. To improve the accuracy, we can increase either M or N.

Tables II and III show the effect on the erorrs by increasing M when N is fixed. In the tables, the data under the column "CNM" are results computed by the Crank–Nicolson method and are given in Wilmott *et al.* (1998, p. 176). Those under "Value" are the values obtained by one trial of our method for the given M and N. The results under the "Mean" and "STD" are the means and standard deviations obtained after 100 trials. The final column "Error" is the difference between "CNM" and the "Mean". We note that when M increases, the standard deviations decrease for all S_0 but the errors do not decrease for some S_0, especially for those close to E. Thus increasing M is not very efficient in this case.

Next we compare Table III with Table IV to see the effect on the errors by increasing N when M is fixed. When N changes from 10 (Tab. IV) to 100 (Tab. III), we see that the errors decrease rapidly for all S_0, even though the standard deviations do not decrease by much. These results indicate that in order to reduce the errors, it is worthwhile to increase N

TABLE III Memory Reduction Method with $M = 10^5$ and $N = 100$.

S_0	*CNM*	*Value*	*Mean*	*STD*	*Error*
2	8.0000	7.9952	7.9950	0.0002	0.0050
4	6.0000	5.9946	5.9950	0.0003	0.0050
6	4.0000	3.9945	3.9950	0.0005	0.0050
8	2.0951	2.0902	2.0911	0.0030	0.0040
10	0.9211	0.9167	0.9183	0.0031	0.0028
12	0.3622	0.3639	0.3612	0.0023	0.0010
14	0.1320	0.1335	0.1329	0.0015	0.0001
16	0.0460	0.0474	0.0461	0.0007	−0.0001

TABLE IV Memory Reduction Method with $M = 10^5$ and $N = 10$.

S_0	*CNM*	*Value*	*Mean*	*STD*	*Error*
2	8.0000	7.9509	7.9500	0.0005	0.0491
4	6.0000	5.9487	5.9500	0.0010	0.0500
6	4.0000	3.9476	3.9450	0.0017	0.0550
8	2.0951	2.0851	2.0827	0.0035	0.0124
10	0.9211	0.9185	0.9157	0.0034	0.0054
12	0.3622	0.3623	0.3595	0.0025	0.0027
14	0.1320	0.1295	0.1311	0.0015	0.0009
16	0.0460	0.0438	0.0457	0.0008	0.0003

than M especially if N is not large. The advantage of our method lies in the fact that we can reduce the errors by increasing N and yet do not incur any penalty on memory.

5 CONCLUSION

We have presented a new simulation technique for pricing American-style options without storing all the intermediate asset prices. The main idea is to simulate the paths in the time-decreasing direction – the backward paths. We have illustrated our method by using the Longstaff–Schwartz method to price an American put option. However, by replacing forward paths with backward paths, our method can in fact be applied to other Monte Carlo methods for other kinds of options. Moreover, as noted at the end of Section 2, our method of generating the backward paths is well-adapted to parallel computations and can be extended to some Quasi Monte Carlo methods too.

Acknowledgements

The research was partially supported by the Hong Kong Research Grant Council grant CUHK4243/01P and CUHK DAG 2060220.

References

Boyle, P. (1977). Options: A Monte Carlo approach. *Journal of Financial Economics*, **4**, 323–338.
Kwok, Y. (1998). *Mathematical Models of Financial Derivatives.* Springer–Verlag, Singapore.
Longstaff, F. and Schwartz, E. (2001). Valuing American option by simulation: A simple least squares approach. *The Review of Financial Studies*, **14**, 113–147.
Ross, S. (1997). *Simulation*, 2nd Ed. Academic Press, San Diego, CA.
Tilley, J. (1993). Valuing American options in a path-simulation model. *Transactions of the Society of Actuaries*, **45**, 563–577.
Wilmott, P., Howison, S. and Dewynne, J. (1998). *The Mathematics of Financial Derivatives.* Cambridge University Press, Cambridge.

APPENDIX

In FORTRAN 90, the commands to set the seed to d are:

```
call random_seed (size = 1)
seed(1) = d
call random_seed(put = seed(1:1))
```

The commands to extract the current seed d are:

```
call random_seed(get = current(1:1))
d = current(1)
```

We remark that our FORTRAN 90 only provides uniformly distributed random numbers and we have used the Box–Muller transform to produce normal distributed random numbers, see Ross (1997, p. 73).

Taylor & Francis
Taylor & Francis Group

Journal… Statist. Comput.Simul, Article ID… GSCS 031056

TO: CORRESPONDING AUTHOR

AUTHOR QUERIES - TO BE ANSWERED BY THE AUTHOR

The following queries have arisen during the typesetting of your manuscript. Please answer the queries.

No Quries	

Production Editorial Department, Taylor & Francis Ltd.
4 Park Square, Milton Park, Abingdon OX14 4RN

Telephone: +44 (0) 1235 828600
Facsimile: +44 (0) 1235 829000