Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

ON THE ANOMALY OF RAN1() IN MONTE CARLO PRICING OF FINANCIAL DERIVATIVES

Akira Tajima
Syoiti Ninomiya
Shu Tezuka

Tokyo Research Laboratory
IBM Research
1623-14, Shimotsuruma, Yamato-shi, Kanagawa-ken, 242, JAPAN

ABSTRACT

Recently, Paskov reported that, the use of a certain pseudo-random number generator, ran1(), which is given in Numerical Recipes in C First Edition, makes Monte Carlo simulations for pricing financial derivatives converge to wrong values.

In this paper, we trace Paskov's experiment, investigate the characteristics and the generation algorithm of the pseudo-random number generator in question, and explain why the wrong convergences occur. We also present a method for avoiding such wrong convergences. A variance reduction procedure is applied together with the method to obtain more precise value, and the effectiveness is examined.

1 INTRODUCTION

Monte Carlo methods are effective for pricing some financial derivatives, especially when the price cannot be calculated analytically because it depends on the historical movement of the underlying variable or on multiple underlying variables (Hull 1993). For example, the payoffs of look-back options depend on the maximum or minimum prices of underlying assets during the lives of the options. The prices of such options are therefore difficult to calculate analytically, but easy to estimate by Monte Carlo simulation. On the other hand, there are two major drawbacks to using Monte Carlo methods in finance. One is that they cannot be smoothly applied to pricing American options, which can be exercised early, and the simulation therefore requires some optimal strategy for the exercise. This problem has been studied by Tilley (Tilley 1993) and Broadie and Glasserman (Broadie and Glasserman 1995). The other is the amount of computation time. Sometimes hundreds of thousands of simulation paths are necessary to obtain the required accuracy. Further, for hedging purposes, the sensitivities of the result to small changes in the initial values of the underlying variables are important in the financial domain.

In recent years, low-discrepancy sequences have been used in this domain, to improve the convergence speed. Paskov (Paskov 1994) investigated the effectiveness of low-discrepancy sequences in pricing a Collateralized Mortgage Obligation (CMO). He also reported that the use of a certain pseudo-random number generator, – ran1(), which is given in Numerical Recipes in C, First Edition (Press et al. 1988) –, makes Monte Carlo simulations converge to wrong values. Morokoff pointed out that in high dimensions, low-discrepancy sequences are no more uniform than random sequences (Morokoff and Caflisch 1994). Consequently, the Monte Carlo method with "random" sequences is still effective. Further, it is important not only to regard ran1() as a random number generator with a poor performance, but also to investigate why the simulation converged to wrong values in Paskov's experiment.

The objectives of this paper are to identify why the wrong values are obtained with ran1(), and to present a method for avoiding such wrong convergences. In section 2, we trace Paskov's experiment by using a pass-through Mortgage Backed Security (MBS), which is simpler than CMO in the sense that it has no tranches. In section 3, we investigate the cycle and the generation algorithm of ran1() and show why the Monte Carlo simulations converge to wrong values. We also present a method for avoiding the problem, and examine the effectiveness of a variance reduction procedure for obtaining a more precise result. Finally, we summarize this paper in section 4.

2 PRICING MORTGAGE-BACKED SECURITIES (MBS)

In this section, we trace Paskov's experiment (Paskov 1994), in which Monte Carlo simulations for estimating the present value of Collateralized Mortgage Obligation (CMO) converged to wrong values when the random number generator ran1() (Press et al. 1988) was used.

2.1 MBS Specification

Paskov does not give the specification of the tranches of CMO, so we simplify the problem into a pass-through MBS which with no tranches. The underlying pool of mortgages has a maturity of thirty years, and the a cash flow occurs every month; in other words, there are 360 cash flows, and the dimension of the problem is also 360. Let C be the monthly payment on the underlying pool of mortgages.

For $1 \leq k \leq 360$,

r_k	–	the appropriate interest rate;
w_k	–	the percentage pre-paying;
$a_{360-k+1}$	–	the remaining annuity;
u_k	–	the discount factor.

Then, a_k is given by

$$a_k = \sum_{i=0}^{n} (1 + \frac{1}{1+r_0}),$$

where r_0 is the current monthly interest rate.

C and a_k are assumed to be constants. r_k, w_k, and u_k are stochastic variables, defined below. The interest rate follows the lognormal model, and the variable r_k is computed as

$$r_k = K_0 e^{\sigma \xi_k} r_{k-1} = K_0^k e^{\sigma \sum_{i=1}^{k} \xi_i} r_0,$$

where $\xi_k (1 \leq k \leq 360)$ is an independent normal random variable $N(0,1)$. Then, the discount factor u_k is given by

$$u_k = \prod_{i=0}^{k-1} (\frac{1}{1+r_i}).$$

The prepayment model w_k is a function of r_k, and is computed as

$$w_k = K_1 + K_2 arctan(K_3 r_k + K_4).$$

The cash flow in month k is

$$M_k = C(1-w_1)\cdots(1-w_{k-1})(1-w_k + w_k a_{360-k+1}).$$

Thus, the present value of the security is the sum of the present values of the cash flows in all the months,

$$PV = \sum_{k=1}^{360} M_k u_k(\xi_1, \ldots, \xi_{360}).$$

In this experiment, we set the constants as follows:

$K_0 = 1/1.020201$	$r_0 = 0.075/12$
$K_1 = 0.24$	$\sigma = 0.02$
$K_2 = 0.134$	$C = 2000$
$K_3 = -261.17 \times 12$	$a_0 = 0$
$K_4 = 12.72$	$w_0 = 0$

To generate normal random variables $N(0,1)$ from uniform random numbers [0,1), we used the inverse method given in RISK (Moro 1995). Obviously, 360 numbers are necessary for one path of Monte Carlo simulation. When we use pseudo-random generators such as ran1(), we assign the numbers from one generator to 360 months sequentially. On the other hand, in the case with low-discrepancy sequences, we assign the i-th dimension of the sequence to the i-th month ($1 \leq i \leq 360$). Consequently, if we focus on the correlation problem among 360 variables, the serial correlations matters most with pseudo-random generators, whereas the independence among dimensions matters most with low-discrepancy sequences.

2.2 Pricing using Ran1()

Figure 1 shows the result of Monte Carlo simulations. The simulation using Sobol sequence converges to the correct value. But simulations using ran1() converge to three different values, all incorrect. This result is consistent with Paskov's study (Paskov 1994). For comparison, a result using drand48(), a system-supplied random number generator of IBM xlC, is also presented. For details of Sobol sequences, see (Sobol 1967).

3 RAN1() IN NUMERICAL RECIPES IN C, FIRST EDITION

In this section, we investigate the characteristics of ran1() in (Press et al. 1988), and identify why the simulation converged to wrong values.

3.1 The Ran1() Algorithm

The ran1() algorithm is based on three linear congruential generators. Linear congruential generators generate a sequence of integers between 0 and $m-1$ by means of the recurrence relation

$$I_{j+1} = aI_j + c \ (mod \ m).$$

If m, a and c are properly chosen, the period of the sequence will be equal to m.

In ran1(), one generator ix_1 with $m_1 = 259,200$, $a_1 = 7,141$, and $c_1 = 54,773$ is used for the most significant part of the output, another generator ix_2

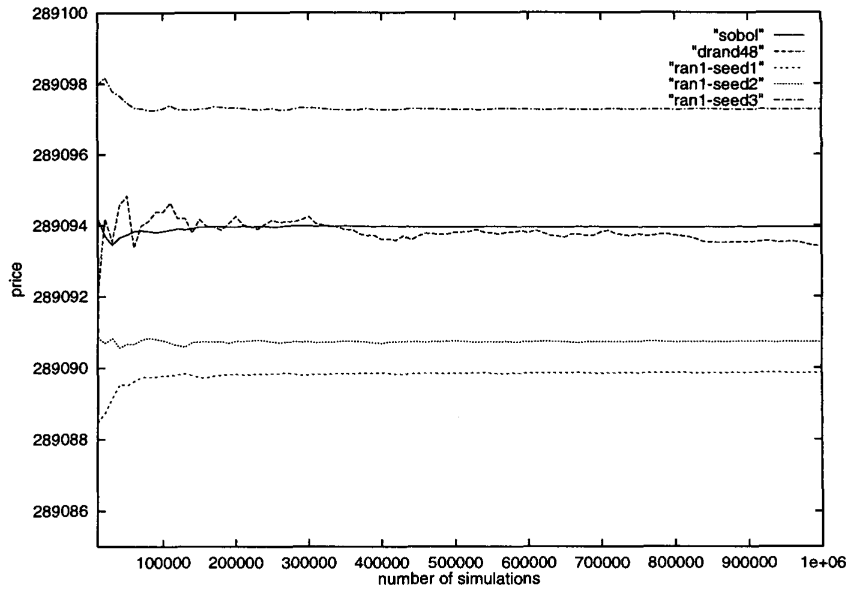

Figure 1: Monte Carlo Simulations using Sobol and ran1() with Three Seeds

with $m_2 = 134,456$, $a_2 = 8,121$, and $c_2 = 28,411$ for the least significant part, and the other generator ix_3 with $m_3 = 243,000$, $a_3 = 4,561$, and $c_3 = 51,349$ for the shuffling, using 97 buckets. The shuffling will prevent serial correlations of the output, which have a negative effect on the result of Monte Carlo method.

The float value $x[0, 1)$ is generated by

$$x = (ix_1 + ix_2 \times \frac{1}{m_2}) \times \frac{1}{m_1}. \quad (1)$$

The ran1() algorithm is as follows:

Initialization: Fill the 97 buckets with float values.

Per Call:

1. Select a bucket by using the value generated by ix_3
2. Return the value in the bucket as output
3. Fill the bucket with the value generated by ix_1 and ix_2

3.2 Discrepancy of Points Generated by Ran1()

We measure the discrepancy of the points generated by ran1(). The discrepancy can be viewed as a quantitative measure for the deviation from uniform distribution, and is related to the error of the numerical integration (Niederreiter 1992). For N points $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{N-1}$ in $[0,1]^k$, $k \geq 1$, and a subinterval $\mathbf{u} = [0, u_1) \times \cdots \times [0, u_k)$, where $0 < u_h \leq 1$ for $1 \leq h \leq k$, we define the L2-discrepancy as

$$T_N^{(k)} = (\int_{[0,1]^k} (\frac{A(\mathbf{u}; N)}{N} - V(\mathbf{u}))^2 d\mathbf{u})^{1/2},$$

where $A(\mathbf{u}; N)$ is the number of values of $n, 0 \leq n < N$, with $\mathbf{v}_n \in \mathbf{u}$, and $V(\mathbf{u})$ is the volume of $\mathbf{u}$.

Let $\mathbf{x}_n$ be a vector of 360 dimensions whose i-th component is used for the i-th dimension of the n-th path of the simulation. We assign values as

$$\mathbf{x}_n = (a_{360(n-1)+1}, a_{360(n-1)+2}, \ldots, a_{360(n-1)+360}),$$

where a_i is the i-th output of ran1(). We calculate the L2-discrepancy of $\mathbf{x}_n$ (Figure 2). The value never falls below 2^{-230}, which is reached after 4,000 points. The move of the value has a pattern with a cycle of about 10,000 or 11,000 points. Consequently, $\mathbf{x}_n$ has a cycle, and the accuracy of the integration does not improve after one cycle.

3.3 Floyd's Algorithm

We have observed that the vector generated from the output of ran1() seems to have a cycle. In this section, we examine whether ran1() has a short cycle.

To obtain the cycle we apply Floyd's algorithm (Knuth 1981). Suppose that we have a sequence of integers $X_0, X_1, X_2, \ldots$, in the range $0 \leq X_n < m$ formed by a rule $X_{n+1} = f(X_n)$. Then X_n is periodic, in the sense that there exist numbers λ and μ

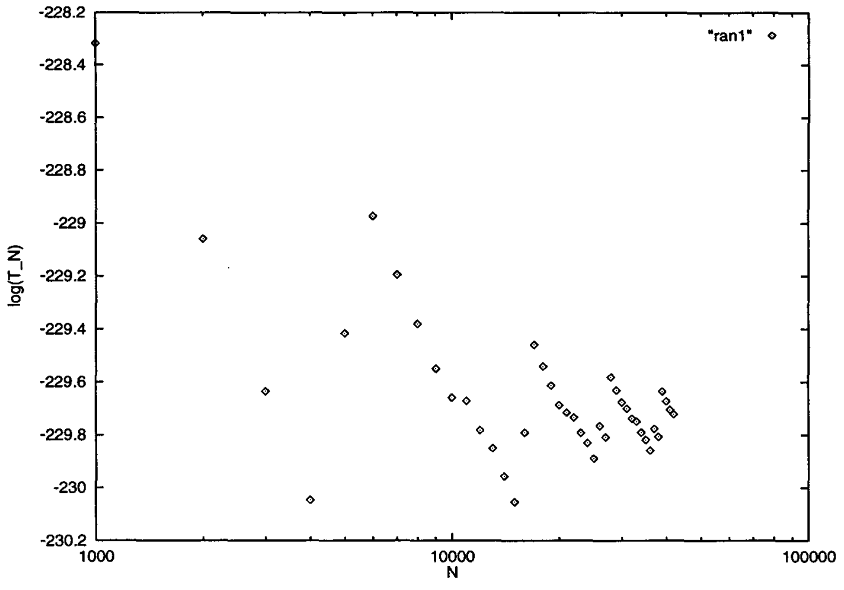

Figure 2: The L2-discrepancy of the Points Generated by ran1()

for which the values $X_0, X_1, \ldots, X_\mu, \ldots, X_{\mu+\lambda-1}$ are distinct, but $X_{n+\lambda} = X_n$ when $n \geq \mu$. That is to say, μ is the length of the transition state and λ is the period of the stationary state. Then there exists a value $n > 0$ such that $X_n = X_{2n}$ and the smallest such value of n lies in the range $\mu \leq n \leq \mu+\lambda$. Once n is found, μ is the smallest i that satisfies $X_i = X_{n+i}$. If none of the values of X_{n+i} for $0 \leq i \leq \mu$ is equal to X_n, $\lambda = n$; otherwise, λ is the smallest such i.

Using the algorithm, we examine the cycle of the sequence generated by ran1(). The algorithm can only be applied to sequences which is formed by a rule $X_{n+1} = f(X_n)$, and therefore satisfies $X_{n+\lambda} = X_n$ for large n. But ran1() does the shuffling using multiple buckets and does not correspond to the condition in the sense that it uses a series of historical values to generate another output, as well as the last output. Here we consider a vector $\mathbf{S}_n$ that consists of all the state variables of ran1(), including ix_1, ix_2, ix_3, and the contents of the buckets, instead of X_n. Then we can say that $\mathbf{S}_{n+1} = f(\mathbf{S}_n)$, and can apply the algorithm.

Unfortunately, the cycle is too large, more than 10^{10}, and we could not obtain it by computer simulations. Anyway, it is very large in comparison with the number necessary for the Monte Carlo simulation, 360×10^6, and does not seem to affect the result of Monte Carlo simulations. We therefore focus on the most significant part of the output of ran1(), that is, we modify ran1() to keep ix_2 equal to zero and apply the same algorithm. This time we obtain the cycle given in Table 1.

Table 1: μ and λ of the Most Significant Part of ran1()

	seed	μ	λ
#1	-386834056	490	3888000
#2	-1947093824	477	3888000
#3	-1505612680	735	3888000
#4	-965150217	444	3888000
#5	-1907586699	527	3888000
#6	-1548523014	650	3888000
#7	-829591941	472	3888000
#8	-1967060894	413	3888000
#9	-1997660300	367	3888000
#10	-158225988	549	3888000
average		512.4	3888000

3.4 The Cycle when Applied to MBS

We continue to examine the cycle of ran1() with the precision $1/m_1$, that is, we ignore the least significant part of the value. From the generation algorithm in Section 3.1, we can see that if all the 97 buckets are visited and the values are all replaced at least once, the effect of the initialization disappears and the generator goes into the stationary state. In the stationary state, the cycle of the sequence C is equal to the least common multiple of m_1 and m_3.

As in Section 3.1, $m_1 = 259,200 = 2^7 \times 3^4 \times 5^2$ and $m_3 = 243,000 = 2^3 \times 3^5 \times 5^3$, and therefore $C = 2^7 \times 3^5 \times 5^3 = 3,888,000$. This coincides with the result in Section 3.3. Here, $C = 3,888,000 = 360 \times 10800$; in other words, the cycle is a multiple of the dimension of the problem. Thus, when applied to the pricing of an MBS with 360 dimensions, there exists a cycle of 10,800 for each dimension. If we consider the vector $\mathbf{x}_n$ in Section 3.2,

$$\mathbf{x}_n = \mathbf{x}_{n+10800}$$

holds for $n > \mu/360$. In other words, a Monte Carlo simulation for an MBS has only 10,800 paths. This coincides with the result obtained in Section 3.2.

We have observed that the number of paths is limited in the stationary state; we therefore estimate the value of μ, the length of the transition state. If we assume that the buckets in ran1() are visited at random instead of through the value of ix_3, the expected number necessary to visit all the buckets, which is equal to the expected μ, can be calculated as the coupon collector's problem (Motwani and Raghavan 1995),

$$E[X] = n \sum_{i=1}^{n} \frac{1}{i} = 97 \times \sum_{i=1}^{97} \frac{1}{i} = 500.23.$$

This theoretical value is consistent with the experimental results given in Table 1. Anyway, one path requires 360 random numbers, and it takes only a few paths to reach the stationary state. Hence the effect of the initial instability is subtle and can be ignored when examining the wrong convergence.

To confirm that the cycle of 10,800 is the cause of the wrong convergence, we compared two Monte Carlo simulations, one a simulation of 1,000,000 paths using ran1(), and the other a simulation of 10,800 paths using the simplified version

$$x = (ix_1 + 0.5) \times \frac{1}{m_1}$$

instead of equation (1). That is to say, we approximate the least significant part by $0.5 \times 1/m_1$.

Figure 3 shows the results with three different seeds. The full simulation converges to the value obtained with the first 10,800 paths which is represented as a horizontal line. The amplitude is caused only by the least significant part.

3.5 Avoiding the Cycle

Here we examine whether we can avoid the incorrect convergence. We have observed that the inaccuracy is caused because the cycle of ran1() is a multiple of the dimension of the problem, 360.

As the cycle of the most significant part of ran1() $C = 3,888,000 = 2^7 \times 3^5 \times 5^3$, Monte Carlo simulation converges to the correct value, if the dimension of the problem is a prime. A simple way to realize this is to skip several random numbers after every path. Namely, for the vector $\mathbf{x}_n$, we assign values as follows:

$$\mathbf{x}_n = \left(a_{(360+\alpha)(n-1)+1}, a_{(360+\alpha)(n-1)+2}, \ldots, a_{(360+\alpha)(n-1)+360}\right)$$

where α is the number of skipped random numbers. Figure 4 shows the result of Monte Carlo simulations using ran1() with the skip numbers 13, 41, and 143, which give the primes 373, 401, and 503 when added to 360. Compared with drand48(), we can say that the anomaly of ran1() is removed.

Further, to check whether they converge to the correct value precisely, we also apply a variance reduction procedure, the antithetic variable technique (Hull 1993). That is to say, after calculating one path by using a normal random number vector $\mathbf{f}_n$, which is generated from the uniform random number vector $\mathbf{x}_n$ as in Section 2.1, we also calculate another path by using $-\mathbf{f}_n$. Figure 5 shows the result obtained by using this antithetic variable technique. In this figure, the X-axis represents the number of paths, and not the number of random number vectors. That is to say, the number of random number vectors generated is the half of the number of paths, and is equal to 500,000.

From these experiments, we conclude that if we apply some appropriate techniques, the correct value can be calculated with a good convergence speed by using ran1().

4 CONCLUSION

We have investigated why the random number generator ran1() in (Press et al. 1988) caused wrong convergences when used for pricing financial derivatives.

The main reason is that the cycle of ran1() is a multiple of the number of dimensions of the problem, 360. Like other multiples of 12, this is a common number of dimensions in the financial domain, because a year has 12 months, and the same phenomenon will occur if the generator is used for pricing other derivative securities.

We have also presented a method for avoiding wrong convergences. Combined with the antithetic variable technique, ran1() can give the correct value with a good convergence speed.

We would like to examine what kinds of tests used to evaluate random number generators detect this problem with ran1().

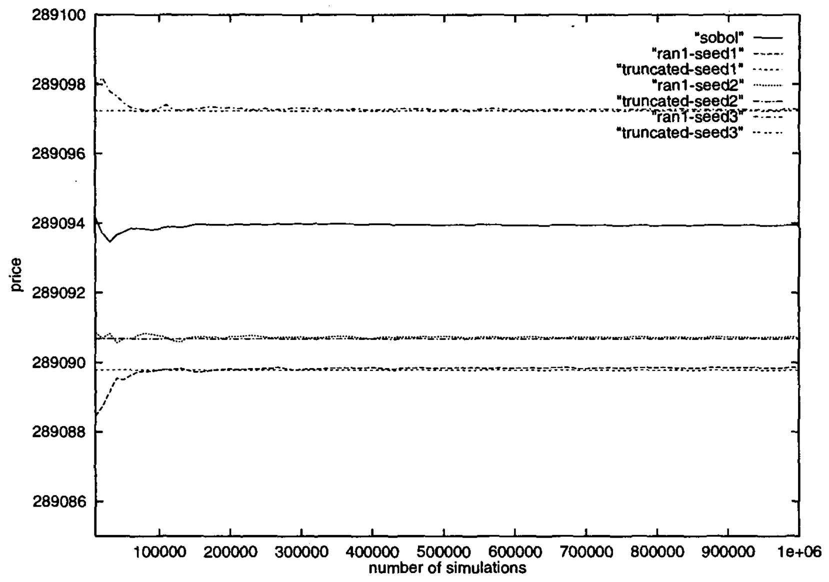

Figure 3: Simulation Almost Converges to the Value of the First 10,800 Trials

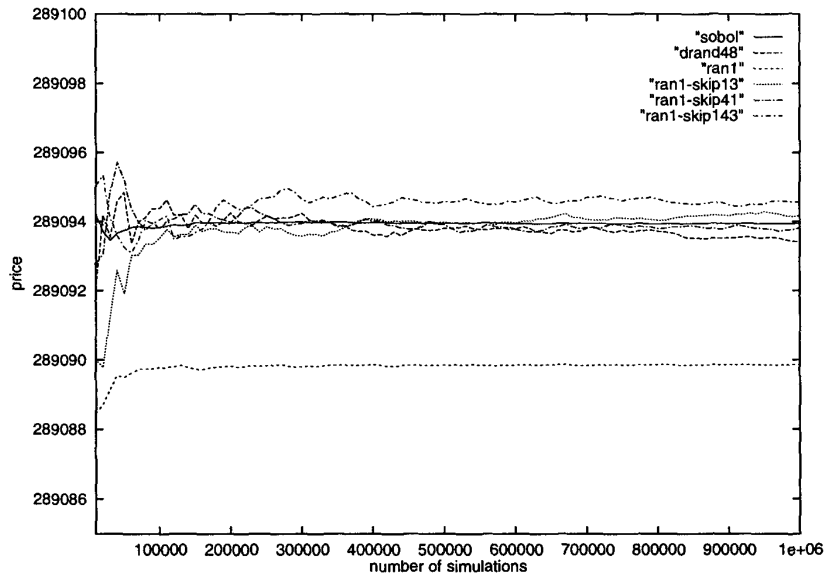

Figure 4: ran1() with Skips after Every 360 Random Numbers

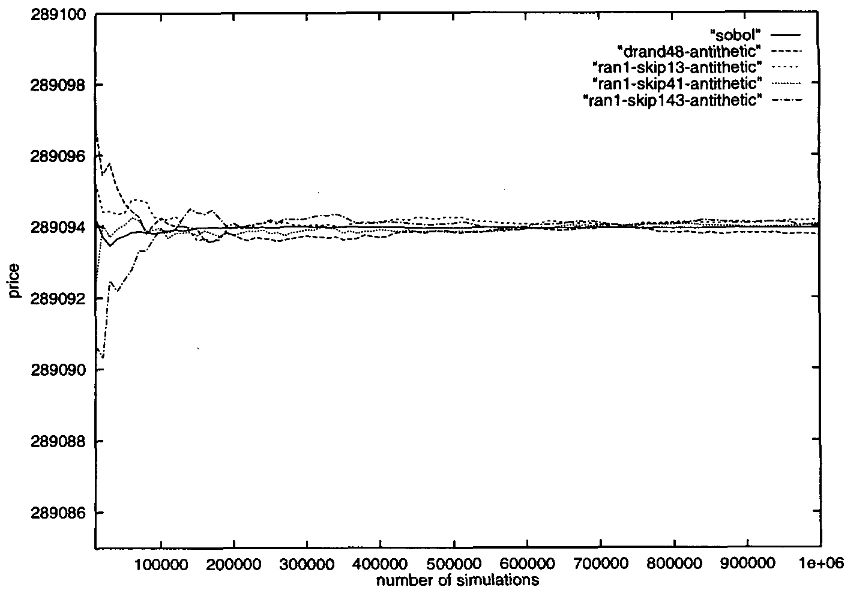

Figure 5: ran1() with Skips and Antithetic Variable Technique

REFERENCES

Broadie, M., and P. Glasserman. 1995. Pricing American-Style Securities Using Simulation. working paper, Columbia University.

Hull, J. C. 1993. *Options, Futures, and Other Derivative Securities, Second Edition.* Prentice Hall International.

Knuth, D. E. 1981. *The Art of Computer Programming, Second Edition, Vol. 2, Seminumerical Algorithms,* Addison-Wesley.

Moro, B. 1995. The Full Monte. *RISK,* Vol. 8, No. 2.

Morokoff, W. J. and R. E. Caflisch. 1994. Quasi-Random Sequences and Their Discrepancies. *Journal on Scientific Computing,* Vol. 15, No. 6.

Motwani, R. and P. Raghavan. 1995. *Randomized Algorithms,* Cambridge University Press.

Niederreiter, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods,* CBMS-NSF Regional Conference Series in Applied Math., No. 63, SIAM.

Paskov,S. H. 1994. Computing High Dimensional Integrals with Applications to Finance. Technical Report CUCS-023-94, Columbia University.

Press, W. H., S. Teukolsky, W. Vetterling, and B. Flannery. 1988. *Numerical Recipes in C, First Edition,* Cambridge University Press.

Sobol, I. M. 1967. The Distribution of Points in a Cube and the Approximate Evaluation of Integrals. *USSR Computational Mathematics and Mathematical Physics,* 7.

Tilley, J. A. 1993. Valuing American Options in a Path Simulation Model. *Transactions of the Society of Actuaries,* Vol. 45.

AUTHOR BIOGRAPHIES

AKIRA TAJIMA is a researcher in IBM Research Tokyo Research Laboratory. His research interests include low-discrepancy sequences, numerical integration, and their applications in financial engineering.

SYOITI NINOMIYA is a researcher in IBM Research Tokyo Research Laboratory. His research interests include pseudo random number generation, low-discrepancy sequences, and numerical integration.

SHU TEZUKA is manager of Exploratory Technologies and Applications group at IBM Tokyo Research Laboratory. He received a doctoral degree in Engineering from University of Tokyo in 1986. His research interests include random number generation, quasi-Monte Carlo methods, combinatorial optimization, discrete event simulation, and financial engineering. He is the author of the monograph, *Uniform Random Numbers: Theory and Practice,* Kluwer Academic Publishers, 1995. In 1995, he was awarded the *12th Inoue Prize for Science* on his significant contributions to uniform random number generation.