LOW-DISCREPANCY SEQUENCES: MONTE CARLO SIMULATION OF OPTION PRICES

Silvio Galanti
is with Integral Development Corporation in Palo Alto, California.

Alan Jung
is an associate professor at San Francisco State University's College of Business.

This article examines the use of Monte Carlo simulation with low-discrepancy sequences (or quasi-Monte Carlo) for valuing complex derivatives contracts versus the more traditional Monte Carlo method using random sequences. Unlike the latter, low-discrepancy (or quasi-random) sequences are deterministic.

Some research has hinted that low-discrepancy sequences improve the rate of convergence of Monte Carlo simulations. Depending on the number of discrete time intervals involved and the smoothness of the problem, the results to date are inconclusive and even contradictory, however.

This article attempts to shed some light on the issue. Low-discrepancy sequences are implemented in several examples using complex path-dependent options, including barrier, Asian, and look-back options.

Because of its simplicity, traditional Monte Carlo simulation has become a popular method for valuing derivatives contracts. It is computationally easy and more efficient than lattice or tree methods at valuing certain path-dependent processes (such as in Asian, barrier, forward start, and look-back options and mortgage-backed securities). This is especially true when the processes depend on several stochastic variables.

Monte Carlo simulation does have a number of disadvantages. In particular, it has difficulty handling the early exercise feature of American options, and it can be computationally onerous to achieve a high level of accuracy. Low-discrepancy sequences, by contrast, use deterministic sampling, which may provide shorter computational times and higher accuracy.

We review both the Monte Carlo method and the most common low-discrepancy sequences. We discuss the problems that may be encountered in implementing these sequences and compare their performance to the more traditional random sequences.

We use both random and low-discrepancy sequences with Monte Carlo simulation to value various derivatives contracts, including not only plain vanilla options, but also such path-dependent derivatives as barrier, Asian, and look-back options. We show that in some cases low-discrepancy sequences are a viable alternative to random sequences.

I. MONTE CARLO SIMULATION

Because there is a voluminous literature on Monte Carlo simulation, we provide only a brief overview here, limiting the discussion to the pricing of derivatives.[1]

In a risk-neutral environment, the value of any derivative security is the discounted value of its expected terminal date cash flow. The expectation is under the risk-neutral measure, where the original probability measure is transformed into an equivalent martingale measure. Interest rates are not stochastic, and discounting occurs at the risk-free rate r. The current price of a derivative security is given by

$$\text{Price} = e^{-rT}E[f(S_0, ..., S_T)] \quad (1)$$

where T is the maturity date of the derivative; E[•] is the expectations operator under the risk-neutral measure; $f(S_0, ..., S_T)$ is the derivative's terminal date cash flow, which may be dependent on the entire price history of the underlying asset; and $S_0, ..., S_T$ is the history of prices for the underlying asset from t = 0 to T.

In its crudest form, Monte Carlo simulation approximates the expectation of the derivative's terminal date cash flows with a simple arithmetic average of the cash flows taken over a finite number of simulated price paths:

$$\text{Price} \approx e^{-rT}\left[\frac{1}{N}\sum_{n=1}^{N} f(S_0^n, ..., S_T^n)\right] \quad (2)$$

where $S_0^n, ..., S_T^n$ is the n-th (n = 1, 2, ..., N) simulated price path of the underlying asset over the life of the derivative, and $f(S_0^n, ..., S_T^n)$ is the derivative's terminal date cash flow from this path.

Convergence Properties of Random Sequences

For a large sample of simulated price paths, the mean of the sample will closely approximate the derivative's true price [i.e., Equation (1)]. The rate of convergence is $\sigma/\sqrt{N}$, where σ is the standard deviation of the population, and N is the number of simulations (price paths).

Unfortunately, the rate of convergence is remarkably slow. For example, to reduce the error by a factor of 10, the number of simulations would have to increase by a factor of 100. Hence, high accuracy requirements may lead to long computation times. Alternatively, reducing σ by a factor of 10 would achieve the same result without the need for additional sampling. Numerous variance reduction techniques are available to reduce the size of σ, such as antithetic, control variate, or importance sampling.[2]

Random Price Paths and Monte Carlo Simulation

To simulate a price path of the underlying asset, we assume that the asset pays no dividends and follows a geometric Brownian process. The n-th price path $S_{i=0}^n, ..., S_{i=s}^n$, for n = 1, 2, ..., N, follows recursively from

$$S_i^n = S_{i-1}^n \exp\left[\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\varepsilon_i^n\sqrt{\Delta t}\right] \quad (3)$$

where the time to maturity has been divided into "s" time intervals of length Δt (i = 1, 2, ..., s = $T/\Delta t$); r is the risk-free drift rate; σ is the underlying asset's volatility; and for each time interval, ε_i^n is a random sample independently drawn from a standardized normal distribution N(0, 1).

After the entire price path $S_{i=0}^n, ..., S_{i=s}^n$ has been simulated, we calculate the derivative's terminal date cash flow $f(S_0^n, ..., S_s^n)$. We repeat the procedure for each additional price path. Often we will find it convenient to refer to each repetition of this procedure as "a simulation."

After a large number of independent simulations, we compute a simple arithmetic average of the resulting terminal values. This gives us the expected terminal value of the derivative, which we then discount to the present at the risk-free rate to obtain the derivative's current price [i.e., Equation (2)].

Exhibit 1 illustrates three price paths, each with ten time intervals, for an asset with r = 10%, σ = 20%, and Δt = 0.10 year.

Simulating the Gaussian Deviates: ε

The key to Monte Carlo simulation is generat-

EXHIBIT 1
THREE PRICE PATHS
(TEN TIME INTERVALS, r = 10%, σ = 20%)

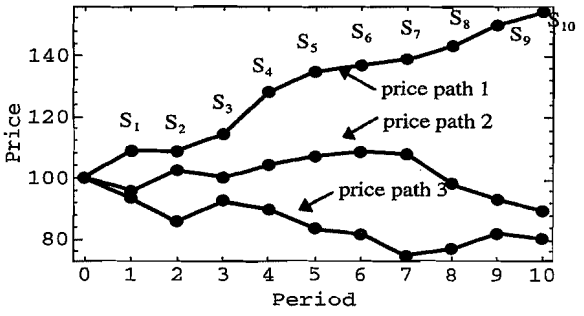
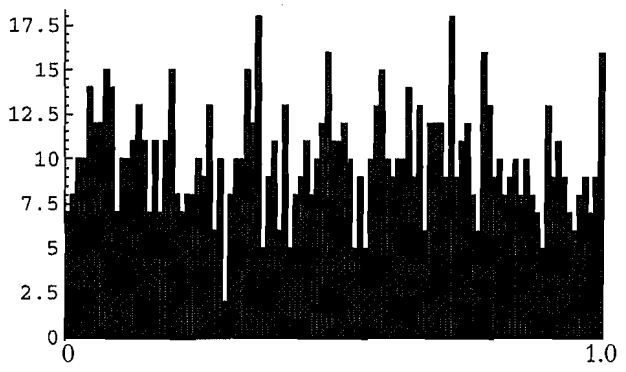


EXHIBIT 2
1,000 RANDOM UNIFORM NUMBERS (0, 1)



ing ε, the N(0, 1) random variables. In practice, the εs are calculated from an algorithm that transforms a random number in the interval (0, 1) into a corresponding Gaussian deviate. Therefore, the problem of generating "good" εs becomes a problem of generating "good" random numbers in the interval.

By "good," we mean that for *each time step*, the distribution of simulated εs across all price paths should closely approximate N(0, 1). Correspondingly, "good" random numbers in the interval (0, 1) should be uniformly distributed over the interval (0, 1).

We illustrate this point by simulating 1,000 price paths (Exhibit 1 depicts three of them). Concentrating only on the first time interval of each price path, Exhibit 2 displays a bar graph of 1,000 uniform random numbers.[3] The horizontal axis is divided into 100 bins, each of size 0.01.

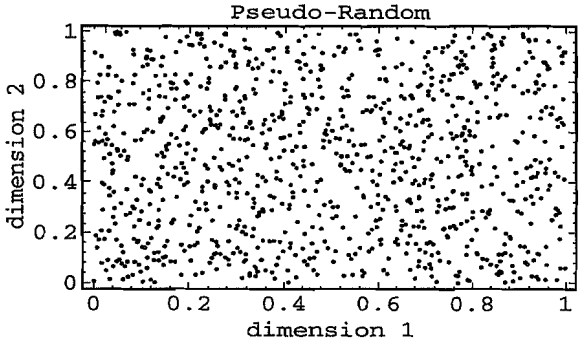
For uniform coverage, we would expect to see ten points per bin. In fact, this is not the case; some bins have as many as eighteen points, while others have as few as two. A high bin count indicates that some points are clustered together, which also implies that some bins will have too few points. As the number of points (i.e., price paths) increases, however, the distribution becomes more uniform.

Therefore, accurately simulating the N(0, 1) distribution requires a large number of price paths. If the number is not large enough, the distribution of simulated ε_is may not be close to N(0, 1). That is, the simulated ε_is may cluster together and not uniformly cover the domain of N(0, 1).

Not only do we want uniform coverage for the first time interval [that is, in the interval (0, 1)], but we also want it across all time intervals. Consequently, the distribution across all time intervals must uniformly fill the unit hypercube. Exhibit 3 illustrates this for the case of two time intervals. The figure plots 1,000 two-dimensional uniformly distributed (0, 1) random numbers, with the first time interval on the horizontal axis and the second one on the vertical axis. Clustering gives rise to several areas that are empty of points. Increasing the number of points will fill in the plot more uniformly.

In general, the more evenly the points are distributed throughout the domain, the more accurate the simulation. This is the approach of low-discrepancy sequences. The points are "deterministically"

EXHIBIT 3
RANDOM SEQUENCE — 1,000 SIMULATIONS



rather than randomly chosen to fill the interval (0, 1) uniformly, thereby minimizing clustering and improving accuracy. Also, by uniformly picking the points, higher accuracy may be achieved with a smaller number of simulations (price paths).

Exhibit 4 demonstrates that low-discrepancy sequences (1,000 two-dimensional points) are more uniformly dispersed throughout the unit square than are random sequences. Unlike the 1,000 random points in Exhibit 3, 1,000 deterministic points exhibit minimal clustering.

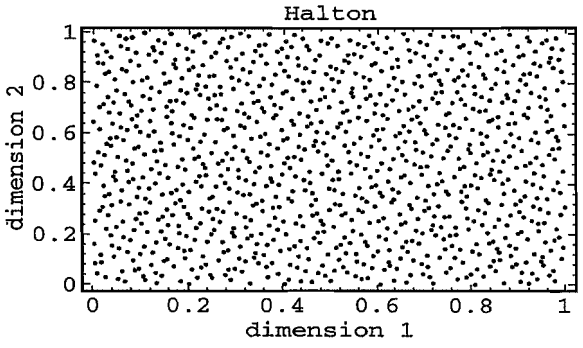
II. LOW-DISCREPANCY SEQUENCES

Low-Discrepancy Price Paths and Monte Carlo Simulation

To simulate a price path using low-discrepancy sequences, we first generate a number in the interval [0, 1), and then transform it into a normally distributed number, $\varepsilon \sim N(0, 1)$, by applying the inverse normal function. For each time interval i, a new ε_i is generated, and the corresponding asset price path follows from Equation (3). For each additional price path, the procedure is repeated.

In the terminology of low-discrepancy sequences, the number of time intervals is referred to as the number of dimensions of the simulation (e.g., number of time intervals = number of dimensions). We use this terminology throughout. Additionally, we frequently interchange the terms "simulations (number of)" with "price paths (number of)."

EXHIBIT 4
LOW-DISCREPANCY SEQUENCE — 1,000 SIMULATIONS



Convergence Properties of Low-Discrepancy Sequences

Recall that crude Monte Carlo simulation using random sequences generates "probabilistic" error bounds that are of the order of $1/\sqrt{N}$, where N is the number of simulations. Also, the error bounds are independent of dimensionality. Low-discrepancy sequences, on the other hand, generate a "deterministic" upper bound to the error that is of the order of $(\log N)^s/N$, where s is the number of dimensions (see Niederreiter [1992]). Boyle et al. [1995] and Joy, Boyle, and Tau [1995] have indicated that in practice the upper bound significantly overestimates the actual error.

Three low-discrepancy sequences for generating uniformly distributed numbers in the interval (0, 1) are the Halton, Faure, and Sobol sequences.

Halton Sequences

We examine the simplest of the low-discrepancy sequences introduced by Halton [1960]. The Halton sequence is a class of multidimensional *infinite* sequences that fill the interval [0, 1). Recall that, along a price path, one time interval is equal to one dimension. The Halton sequence takes any non-negative integer, and transforms it into a number in the interval [0, 1). While tedious, the transformation is purely mechanical and is easily programmed for computer.

To generate a multidimensional Halton sequence, we follow a two-step procedure. Typically, we begin with a consecutive sequence of non-negative integers, for example n = 0, 1, 2, 3, ..., N – 1, where N is the number of simulations (price paths).

Step 1: Each integer n is expanded in an arbitrary base p, where p is any prime number greater than or equal to 2. That is, each integer is converted to its representation in the base p number system (e.g., 011 is the base 2 representation of the integer 3).

Step 2: The base p number is transformed into a number in the interval [0, 1) by reflection about the decimal point. This step is best illustrated with an example. The other low-discrepancy sequences use a variation of this procedure.

We illustrate the two-step procedure for the

one-dimensional Halton sequence in base 2. We start with a sequence of non-negative integers, and then transform each integer into its Halton representation. From the perspective of a one-dimensional Monte Carlo simulation (e.g., simulating only the terminal date cash flows to a European option), 10,000 Monte Carlo simulations typically would use n = 0, 1, 2, 3, ..., 9,999, with the n-th integer corresponding to the (n + 1)-th simulation .

We limit our example to the first six integers, n = 0, 1, 2, ..., 5. Because the procedure is identical for all n, we illustrate it only for n = 4.

- Step 1: Expand the integer 4 in powers of 2; i.e., convert 4 to the base 2 number system. The expansion is

  $$4 = 1(2^2) + 0(2^1) + 0(2^0)$$

  The coefficients of the expansion (i.e., 1, 0, and 0) become the unique digit expansion of 4 in the base 2 number system. Thus, "100" is the base 2 representation of 4.
- Step 2: Convert "100" into a number in the interval [0, 1) by reflecting it about the decimal point (i.e., reverse the digits of "100," and shift the binary decimal point all the way to the left, making 0.001). In the base 2 number system, the binary fraction "0.001," i.e.,

  $$0.001 = 0\left(\frac{1}{2^1}\right) + 0\left(\frac{1}{2^2}\right) + 1\left(\frac{1}{2^3}\right)$$

  represents 1/8. Thus the corresponding Halton number for the integer is 1/8, which clearly falls within the interval [0, 1).

Exhibit 5 summarizes the two-step procedure for generating the first six one-dimensional Halton numbers from n = 0, 1, 2, ..., 5.

The multidimensional Halton sequence is identical to the one-dimensional Halton sequence, except that a different base (i.e., prime number) is used for each of the dimensions. Successive dimensions are generated sequentially following the two-step process. Typically, for each additional dimension,

EXHIBIT 5
TWO-STEP PROCEDURE: FIRST-DIMENSION HALTON NUMBERS

n	Step 1 Base 2 Expansion	Step 2 Reflection about the Decimal Point Base 2	
0	0	$\frac{0}{2^1} + \frac{0}{2^2} + \frac{0}{2^3} = 0$	(0.0 in base 2)
1	1	$\frac{1}{2^1} + \frac{0}{2^2} + \frac{0}{2^3} = \frac{1}{2}$	(0.1 in base 2)
2	10	$\frac{0}{2^1} + \frac{1}{2^2} + \frac{0}{2^3} = \frac{1}{4}$	(0.01 in base 2)
3	11	$\frac{1}{2^1} + \frac{1}{2^2} + \frac{0}{2^3} = \frac{3}{4}$	(0.11 in base 2)
4	100	$\frac{0}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} = \frac{1}{8}$	(0.001 in base 2)
5	101	$\frac{1}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} = \frac{5}{8}$	(0.101 in base 2)

the prime numbers are selected in successive order (e.g., 2, 3, 5, 7, ...,), where, for example, the k-th prime number corresponds to the k-th dimension.

The Halton Algorithm. The two-step procedure for generating an s-dimensional Halton sequence (i.e., s time intervals per price path) is:

- Step 1: For each of the s dimensions, say, the k-th one, convert a consecutive sequence of non-negative integers (i.e., n = 0, 1, 2, 3, ..., N − 1, where N = number of price paths) into their representation in the base p_k number system, where p_k denotes the k-th prime number greater than or equal to 2. Formally, we have

  $$n = \sum_{j=0}^{\infty} a_j^k(n)(p_k)^j \quad \text{for any integer } n \geq 0 \qquad (4)$$

  $a_j^k(n)$ is the j-th digit of the unique digit expansion of n in base p_k (e.g., the expansion of n = 3 in base $p_1 = 2$ is "11," where the digits "1" and "1" are the 1st and 0th digit of the base 2 expan-

EXHIBIT 6
HALTON SEQUENCE — FIRST DIMENSION $[\phi_2^1(n)]$

Cycle	1		2		3		4		5		6		7		8	
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi_2^1(n)$	0	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{8}$	$\frac{5}{8}$	$\frac{3}{8}$	$\frac{7}{8}$	$\frac{1}{16}$	$\frac{9}{16}$	$\frac{5}{16}$	$\frac{13}{16}$	$\frac{3}{16}$	$\frac{11}{16}$	$\frac{7}{16}$	$\frac{15}{16}$

sion, implying $[...a_2^1(3), a_1^1(3), a_0^1(3)] = [...0, 1, 1])$.

- Step 2: To obtain a number in the interval [0, 1), reflect the digit expansion about the decimal point using

$$\phi_{p_k}^k(n) = \sum_{j=0}^{\infty} a_j^k(n)(p_k)^{-j-1} \quad (5)$$

The result is the n-th Halton number $\phi_{p_k}^k(n)$ for the k-th dimension. For example, using the previous example of n = 3, we have $\phi_2^1(3) = 3/4 = (1)(2^{-1}) + (1)(2^{-2}) + (0)(2^{-3})$ or equivalently 0.11 in base 2 from reflecting 11 about the decimal point.

The two steps are repeated for each of the remaining s dimensions, with each successive dimension using the next largest prime number as its base. From Equations (4) and (5), we see that if it were not for the different base, the Halton sequences for each dimension would be identical.

Exhibits 6 and 7 provide the first sixteen Halton numbers for dimensions 1 and 2, respectively. The tables demonstrate that each successive Halton number (n = 0, 1, 2, 3, ..., 15) knows how to "fill in the gaps" in the preceding numbers. As n increases beyond fifteen, the resulting Halton numbers for each dimension begin to cover the interval [0, 1) uniformly.

Because each successive Halton number "fills in the gaps" of the existing sequence, we do not need to know in advance how many integers are required to achieve a specific level of accuracy. This implies that we can easily implement a stopping rule for the simulation.

Halton Sequence: Uniform Coverage. Looking at Exhibit 6 for dimension 1, we see that the sequence of sixteen Halton numbers is grouped in cycles of length 2 (i.e., eight distinct pairs). The length of the cycle equals the base of dimension 1, i.e., $p_1 = 2$.

Within each cycle, the numbers are increasing in magnitude and proportional to powers of 1/2. The first cycle begins with terms proportional to 1/2, implying the cycle has a grid size of 1/2. Moving from one cycle to the next, the Halton sequence in general moves to smaller and smaller grid sizes (e.g., 1/2, 1/4, 1/8, 1/16, ...,), with each succeeding Halton number filling in the gaps left by the numbers of the preceding cycles.

The second dimension "fills in the gaps" in an identical fashion (Exhibit 7), but the base is the next prime number, 3, and the terms are in cycles of

EXHIBIT 7
HALTON SEQUENCE — SECOND DIMENSION $[\phi_3^2(n)]$

Cycle	1			2			3			4			5			
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi_3^2(n)$	0	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{9}$	$\frac{4}{9}$	$\frac{7}{9}$	$\frac{2}{9}$	$\frac{5}{9}$	$\frac{8}{9}$	$\frac{1}{27}$	$\frac{10}{27}$	$\frac{19}{27}$	$\frac{4}{27}$	$\frac{13}{27}$	$\frac{22}{27}$	$\frac{7}{27}$

length 3 with grid sizes that are powers of 1/3, (e.g., 1/3, 1/9, 1/27, ...,).

Because the base (i.e., prime number) of each Halton sequence increases with increasing dimensions, high-dimensional Halton sequences exhibit long cycle lengths. For example, with twenty-six dimensions, the corresponding prime number is $p_{26} = 101$ (i.e., the twenty-sixth prime number). The resulting cycle has 101 terms. This implies that the speed at which increasingly finer grid points are generated decreases with increasing dimensions.[4]

Several authors indicate that for high dimensions, low-discrepancy sequences tend to be overly sensitive to the starting values of "n," because reflection about the decimal point generates values that cluster near zero, e.g., $1/p \approx 0$ for large p.[5] To minimize the effect, typically the first 10 to 200 integers are discarded, implying that the initial sequence n = 0, 1, 2, ..., becomes n = 10, 11, 12, ..., or n = 200, 201, 201, ..., or anything in between.

Faure Sequences

The Faure sequence essentially is a permutation of the Halton sequence. Unlike the Halton sequence, it uses the *same* base for each dimension (i.e., time interval). That base is the smallest prime number that is greater than or equal to the number of dimensions in the problem and not smaller than 2.

As previously mentioned, using the same base for each dimension is problematical for the Halton sequence, because the resulting sequences are identical across all dimensions. The Faure sequence overcomes this difficulty by reordering the Faure numbers within each dimension, so that the sequences are different for each dimension.

We illustrate the procedure for generating a two-dimensional Faure sequence. The procedure for creating higher-dimensional Faure sequences is identical.

For the two-dimensional Faure sequence, the base is 2. We generate the Faure numbers for each dimension sequentially, beginning with the first dimension. The construction of the first dimension is identical to the two-step procedure for the Halton sequence.

Exhibit 8 shows the first sixteen Faure numbers of dimension 1 that are generated from the integers n = 0, 1, 2, 3, ..., 15 (i.e., sixteen price paths or simulations).

After calculating the Faure sequence for dimension 1, we proceed to the second dimension. For ease of discussion, we first state the general procedure for constructing the Faure sequence, and then illustrate it with several examples. For this as well as all higher dimensions, we follow a three-step procedure. For each of the integers n = 0, 1, 2, 3, ..., 15, the three steps are:

- Step 1: Convert each integer n into its representation in the base p number system:

$$n = \sum_{j=0}^{\infty} a_j(n) p^j \tag{6}$$

where $a_j(n)$ is the j-th digit of the unique digit expansion of n in base p. Because the Faure sequence uses the same base for each dimension, this step needs to be done only once (e.g.,

EXHIBIT 8
FAURE SEQUENCE — FIRST AND SECOND DIMENSIONS $[\phi_2^1(n), \phi_2^2(n)]$

Cycle	1		2		3		4		5		6		7		8	
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi_2^1(n)$	0	1/2	1/4	3/4	1/8	5/8	3/8	7/8	1/16	9/16	5/16	13/16	3/16	11/16	7/16	15/16
$\phi_2^2(n)$	0	1/2	3/4	1/4	5/8	1/8	3/8	7/8	15/16	7/16	3/16	11/16	5/16	13/16	9/16	1/16

the first dimension).

- Step 2: Reorder the digit expansion ($a_j(n)$ from step 1), and obtain a new sequence of ordered digits, denoted by $b_i(n)$. The reordering follows from

$$b_i(n) = \left[\sum_{j\geq i}^{\infty}(k-1)^{j-i}\,{}^{j}C_i a_j(n)\right] \bmod p \qquad (7)$$

where ${}^{j}C_i = \dfrac{j!}{i!(j-i)!}$ is the binomial coefficient; k = the k-th dimension; and mod p denotes modular p arithmetic (i.e., ([•] mod p) is the remainder from dividing [•] by p).

- Step 3: Obtain the n-th Faure number $\phi(n)$ by reflecting the $b_i(n)$ from step 2 about the decimal point:

$$\phi(n) = \sum_{i=0}^{\infty} b_i(n)p^{-i-1} \qquad (8)$$

We illustrate the three-step procedure below, but for ease of discussion we express Equation (7) in matrix notation:

$$\begin{bmatrix} b_0(n) \\ b_1(n) \\ b_2(n) \\ . \\ . \\ . \end{bmatrix} = \begin{bmatrix} {}^{0}C_0 & (k-1)\,{}^{1}C_0 & (k-1)^2\,{}^{2}C_0 & \cdots \\ 0 & {}^{1}C_1 & (k-1)^1\,{}^{2}C_1 & \cdots \\ 0 & 0 & {}^{2}C_2 & \cdots \\ . & . & . & \\ . & . & . & \\ . & . & . & \end{bmatrix} \times \begin{bmatrix} a_0(n) \\ a_1(n) \\ a_2(n) \\ . \\ . \\ . \end{bmatrix} \bmod p \qquad (9)$$

For the second dimension, we have k = 2. Recall the base of the two-dimensional Faure sequence is p = 2. Because the procedure is identical for all n, we illustrate it only for n = 2.

- Step 1: From Equation (6), the base 2 expansion of n = 2 is [..., $a_2(2)$, $a_1(2)$, $a_0(2)$] = [..., 0, 1, 0].
- Step 2: Using Equation (9), reorder the expansion to obtain ...$b_2(2)$, $b_1(2)$, $b_0(2)$:

$$\begin{bmatrix} b_0(2) \\ b_1(2) \end{bmatrix} = \begin{bmatrix} {}^{0}C_0 & {}^{1}C_0 \\ 0 & {}^{1}C_1 \end{bmatrix}\begin{bmatrix} a_0(n) \\ a_1(n) \end{bmatrix} \bmod 2$$

$$= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \bmod 2$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \bmod 2$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The reordered digit expansion is [..., $b_2(2)$, $b_1(2)$, $b_0(2)$] = [..., 0, 1, 1].

- Step 3: Using Equation (8), reflect $b_i(2)$ about the decimal point. The third Faure number $\phi(2)$ is

$$\phi(2) = \sum_{i=0}^{\infty} b_i(2)2^{-i-1}$$

$$= \frac{1}{2} + \frac{1}{4}$$

$$= \frac{3}{4}$$

The remaining Faure numbers for dimension 2 are generated identically (see Exhibit 8).

The Faure Algorithm. To generate an s-dimensional Faure sequence (i.e., s time intervals per price path), beginning with the first dimension, typically each successive dimension is generated sequentially. We start with a consecutive sequence of non-negative integers (i.e., n = 0, 1, 2, ..., N − 1, where N is the number of price paths).

- Step 1: Using Equation (6), convert each integer n into its representation in the base p number system, where p is the smallest prime number satisfying $p \geq s$ and $p \geq 2$.

  For dimension 1 only:

- Step 2: For each integer n, obtain the n-th Faure number by reflecting its digit expansion from step 1 about the decimal point; see Equation (5).

  For all higher dimensions $2 \leq k \leq s$:

- Step 2: Using Equation (7), obtain the reordered digit expansion [i.e., $b_i(n)$] for each integer n.
- Step 3: Using Equation (8), obtain the n-th Faure number by reflecting the reordered digit expansion $b_i(n)$ about the decimal point.

For each additional dimension, repeat steps 2 and 3.

Faure Sequence: Uniform Coverage. The multidimensional Faure sequence fills in the unit hypercube in cycles of length p. But unlike the Halton sequence, the Faure sequence no longer lies in cycles of increasing terms (with the exception of the first dimension). Also, unlike the Halton sequence, which uses a different grid size for each dimension, the Faure sequence uses the same initial grid size of 1/p across all dimensions. This is illustrated in Exhibit 8 for two dimensions using the first sixteen terms of the sequence. The base is $p = 2$, which also is the cycle length.

Recall that the speed at which the Halton sequence generates increasingly finer grid points is inversely related to the size of its base (cycle length). This also holds for the Faure sequence, but for the same number of dimensions the Faure sequence generally uses a significantly smaller base.

For example, with twenty-six dimensions, the Faure base is $p = 29$. Compare this to the Halton sequence, which uses each of the first twenty-six prime numbers ($p_1 = 2$, $p_2 = 3, 5, \ldots, p_{26} = 101$). Therefore, for the same number of points, the Faure sequence should "fill in the gaps" in the existing sequence at a faster rate than the Halton sequence.

Because the Faure sequence is essentially a reordering of the Halton sequence, it suffers from the same start-up maladies. In particular, for high dimensions and low values of "n," the Faure points tend to cluster about zero. Additionally, the points become repetitive at low values of "n." To minimize these effects, Faure suggests discarding the first $n = (p^4 - 1)$ points, where p is the base of the Faure sequence (see Bratley and Fox [1988]).

For example, for a one-dimensional Monte Carlo simulation using 10,000 simulations ($p = 2$), rather than initiate the Faure sequence with the consecutive integers $n = 0, 1, 2, \ldots, 9{,}999$, we discard the first fifteen ($= 2^4 - 1$) of these, and initiate the Faure sequence with $n = 15, 16, 18, \ldots, 10{,}014$. As the dimensionality of the Monte Carlo simulation increases, so does the number of discarded points.

Sobol Sequences

The Sobol sequence, like the Faure sequence, is a reordering of the Halton sequence. Unlike the two other sequences, the multidimensional Sobol sequence eliminates the problems caused by large prime numbers by using only the single base $p = 2$. The low base implies a short cycle length, which optimizes the speed at which increasingly finer grid points are generated.

Bratley and Fox [1988] discuss two algorithms for generating Sobol sequences: 1) Sobol's original method, and 2) Antonov and Saleev's [1979] variation. Antonov and Saleev demonstrate that their variation is faster than Sobol's original method. Consistent with industry experience, we find their variation to be approximately 20% faster. We use their variation throughout.

The discussion follows from Bratley and Fox [1988], of which we summarize only the salient points. The construction of a multidimensional Sobol sequence follows a four-step procedure. Because the four steps are identical for each dimension, we illustrate the procedure for only a single dimension.

For each of the four steps, we state the procedure for constructing a Sobol sequence, and then illustrate it with several examples.

Step 1. Generate a set of odd integers m_i, for $i = 1, 2, \ldots, [\log_2 N]$, that satisfy the condition: $0 < m_i < 2^i$, where N is the number of price paths, and $[\log_2 N]$ is the smallest integer larger than $\log_2 N$. $[\log_2 N]$ is the maximum number of digits in the

expansion of N in base 2. This entails a complicated recursive procedure.

We start with a series of integers h_1, h_2, h_3, ..., where h_j is either "0" or "1." The h_j are the coefficients of a primitive polynomial of modulo 2. That is, the polynomials have coefficients that are either 0 or 1.[6] A primitive polynomial of degree d is

$$P = x^d + h_1x^{d-1} + h_2x^{d-2} + ... + h_{d-1}x + 1 \quad (10)$$

Our only interest in the primitive polynomials is in their coefficients.

Next, we generate the set of m_i for all i using the coefficients of the primitive polynomial and a recursive relationship for i > d:

$$m_i = 2h_1m_{i-1} \oplus 2^2h_2m_{i-2} \oplus ... \oplus 2^{d-1}h_{d-1}m_{i-d+1} \oplus 2^dm_{i-d} \oplus m_{i-d} \quad (11)$$

where h_1, h_2, ..., h_{d-1} are the coefficients of the primitive polynomial of degree d, and $\oplus$ is the bit-by-bit exclusive-or (EOR) operator:

$$1 \oplus 0 = 0 \oplus 1 \equiv 1$$

$$1 \oplus 1 = 0 \oplus 0 \equiv 0$$

Because Equation (11) generates only m_i for i > d, the first "d" odd integers (i.e., m_1, m_2, ..., m_d) must be supplied. Fortunately, they can be chosen freely provided they satisfy that m_i is odd and $0 < m_i < 2^i$.

We illustrate this procedure using a third-degree primitive polynomial:

$$P = x^3 + x^2 + 1$$

$$= 1x^3 + \underbrace{1}_{h_1}x^2 + \underbrace{0}_{h_2}x + 1 \quad (12)$$

with coefficients $h_1 = 1$ and $h_2 = 0$. The recurrence relationship [Equation (11)] becomes

$$m_i = 2m_{i-1} \oplus 2^3m_{i-3} \oplus m_{i-3} \quad (13)$$

We arbitrarily choose the first three m_i, i.e., $m_1 = 1$, $m_2 = 3$, and $m_3 = 7$. Beginning with i = 4, we have from Equation (13):

$$m_4 = 2m_3 \oplus 8m_1 \oplus m_1$$

$$= 14 \oplus 8 \oplus 1$$

$$= 1110 \oplus 1000 \oplus 0001 \quad \text{expanding in base 2}$$

$$= 0111$$

$$= 7 \quad \text{in base 10}$$

For i = 5, we have from Equation (13):

$$m_5 = 2m_4 \oplus 8m_2 \oplus m_2$$

$$= 14 \oplus 24 \oplus 3$$

$$= 01110 \oplus 11000 \oplus 00001 \quad \text{expanding in base 2}$$

$$= 10{,}111$$

$$= 23 \quad \text{in base 10}$$

and so on.

We see that each m_i is an odd integer satisfying, $0 < m_i < 2^i$. Using the third-degree primitive polynomial in Equation (12), Exhibit 9 summarizes the results for the first six values of m_i.

Step 2. Calculate a set of "direction numbers" by converting m_i into a binary fraction in the base 2 number system. The i-th direction number v(i) for i = 1, 2, ..., $[\log_2 N]$ is given by

$$v(i) = \frac{m_i}{2^i} \quad \text{in base 2} \quad (14)$$

Because the procedure is identical for all direction numbers, we illustrate it only for i = 4. From Equation (14) and using the fourth odd integer, $m_{i=4} = 7$ from Exhibit 9, we have

$$v(4) = \frac{7}{2^4} = \frac{7}{16}$$

Expanding 7/16 in base 2, we have

$$\frac{7}{16} = \frac{0}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{0}{2^5} + \ldots +$$

where the coefficients of the expansion are 0, 1, 1, and 1. Therefore, v(4) = 0.0111 in the base 2 number system. Equation (14) is equivalent to right-shifting the binary expansion of $m_{i=4} = 7$ by i = 4 digits (adding preceding "0s" if necessary). That is, the base 2 expansion of 7 is "111," and shifting the binary decimal point to the left four digits by adding a preceding 0, we have v(4) = 0.0111. Exhibit 9 summarizes the results for the first six direction numbers.

Step 3. Convert a consecutive sequence of non-negative integers (i.e., n = 0, 1, 2, ..., N – 1) into their representation in the base 2 number system. Because the Sobol sequence uses the same base 2 for each dimension, this step needs to be done only once.

Step 4. Calculate the n-th Sobol number $\phi(n)$ for n = 0, 1, ..., N – 1 using the Antonov and Saleev recursive algorithm:

$$\phi(n + 1) = \phi(n) \oplus v(c) \qquad (15)$$

where $\phi^k(0) = 0$; v(c) is the c-th direction number; and c is the rightmost zero-bit in the base 2 expansion of n (e.g., the rightmost zero-bit of "010001" (n = 17) is the second digit from the right; hence c = 2).

Using the direction numbers from Exhibit 9, we illustrate step 4 for generating the first six Sobol numbers based on the primitive polynomial in Equation (12). For n = 0, the binary expansion of "0" is [..., $a_1(0)$, $a_0(0)$] = [..., 0, 0]. Because the rightmost zero-bit is c = 1, we use the first direction number, v(1) = 0.1, from Exhibit 9. From Equation (15) with $\phi^k(0) = 0$, the first Sobol number is

$$\begin{aligned} \phi(1) &= \phi(0) \oplus v(1) \\ &= 0.0 \oplus 0.1 && \text{in binary} \\ &= 0.1 && \text{in binary} \\ &= \frac{1}{2} && \text{in base 10} \end{aligned}$$

For n = 1, its binary expansion is [..., $a_1(1)$, $a_0(1)$] = [..., 0, 1]. Because the rightmost zero-bit of "01" is c = 2, we use the second direction number, v(2) = 0.11, from Exhibit 9. From Equation (15), the second Sobol number is

$$\begin{aligned} \phi(2) &= \phi(1) \oplus v(2) \\ &= 0.10 \oplus 0.11 && \text{in binary} \\ &= 0.01 && \text{in binary} \\ &= \frac{1}{4} && \text{in base 10} \end{aligned}$$

For n = 2, its binary expansion is [..., $a_1(2)$, $a_0(2)$] = [..., 1, 0]. The rightmost zero-bit of "10" is c = 1, and we use the first direction number, v(1) = 0.1, from Exhibit 9. From Equation (15), the third Sobol number is

$$\begin{aligned} \phi(3) &= \phi(2) \oplus v(1) \\ &= 0.01 \oplus 0.10 && \text{in binary} \\ &= 0.11 && \text{in binary} \\ &= \frac{3}{4} && \text{in base 10} \end{aligned}$$

and so on. The last row of Exhibit 9 shows the first six Sobol numbers (excluding the 0th).

The Sobol Algorithm. To generate an s-dimensional Sobol sequence (i.e., s time intervals per price path), beginning with the first dimension, each successive dimension is generated sequentially using a

EXHIBIT 9
SOBOL: ODD INTEGERS (m_i), DIRECTION NUMBERS, AND SOBOL [$\phi(n)$]

i (or n)	1	2	3	4	5	6
m_i	1	3	7	7	23	17
m_i base 2	1	11	111	111	10,111	10,001
v(i) base 10	1/2	$3/2^2$	$7/2^3$	$7/2^4$	$23/2^5$	$17/2^5$
v(i) base 2	0.1	0.11	0.111	0.0111	0.10111	0.010001
Sobol $\phi(n)$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{8}$	$\frac{5}{8}$	$\frac{3}{8}$

different primitive polynomial. Typically, the procedure begins with the primitive polynomial of lowest degree, and moves to higher degrees as s increases. For each dimension:

- Step 1: Using the primitive polynomial Equation (10) and the recursive relationship Equation (11), generate a set of *odd* integers m_i for i = 1, 2, ..., $[\log_2 N]$ that satisfy the condition: $0 < m_i < 2^i$, where N is the number of price paths, and $[\log_2 N]$ is the smallest integer larger than $\log_2 N$.
- Step 2: Using Equation (14), calculate a set of direction numbers v(i) by converting m_i into a binary fraction in the base 2 number system (i.e., by right-shifting the binary expansion of m_i by i digits).
- Step 3: Using Equation (6) with p = 2, convert a consecutive sequence of non-negative integers (i.e., n = 0, 1, 2, ..., N − 1), into their representation in the base 2 number system. This step needs to be done only once.
- Step 4: Beginning with n = 0, and moving consecutively to the next higher integer, locate the rightmost zero-bit of the binary expansion of n from step 3. Label this bit as the c-th digit (counting from the rightmost digit of the expansion). Use the c-th direction number v(c) that is calculated in step 2 along with Equation (15) to calculate the n-th Sobol number. Repeat the procedure for each n.

Repeat steps 1, 2, and 4 for each of the s dimensions, using a different primitive polynomial for each dimension.

Sobol Sequence: Uniform Coverage. The s-dimensional Sobol sequence fills in the unit hypercube in cycles of length 2. Within each cycle, the points are not necessarily increasing in magnitude. The initial grid size is 1/2 across all dimensions, and each succeeding grid size is a factor of 1/2 smaller.

For high dimensions, the Sobol sequence uses a shorter cycle length than either the Halton or Faure sequence. Therefore, for the same number of points, the Sobol sequence should "fill in the gaps" in the existing sequence at a faster rate than either of the two other sequences.

For three dimensions, the first sixteen terms are shown in Exhibit 10. The first row of the table is the base 2 Halton sequence, and each succeeding row (i.e., dimension) is a reordering of the first row. The low base of the Sobol sequence eliminates the low values of n clustering that the two other sequences experience, while it introduces another low values of n problem; namely, the points tend to repeat themselves across dimensions. This results in additional clustering.

Again, the solution is to discard the first m points (i.e., initiate the sequence with n = m, m + 1, m + 2, ..., instead of n = 0, 1, 2, ...,). Boyle, Broadie, and Glasserman [1995] use m = 64, but the choice appears to be arbitrary.

EXHIBIT 10
SOBOL SEQUENCE — FIRST THREE DIMENSIONS $[\phi_2^1(n), \phi_2^2(n), \text{AND } \phi_2^3(n)]$

Cycle	1		2		3		4		5		6		7		8	
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi_2^1(n)$	0	1/2	3/4	1/4	3/8	7/8	5/8	1/8	3/16	11/16	15/16	7/16	5/16	13/16	9/16	1/16
$\phi_2^2(n)$	0	1/2	1/4	3/4	3/8	7/8	1/8	5/8	5/16	13/16	1/16	9/16	3/16	11/16	7/16	15/16
$\phi_2^3(n)$	0	1/2	3/4	1/4	5/8	1/8	3/8	7/8	5/16	13/16	9/16	1/16	15/16	7/16	3/16	11/16

Efficiency of the Halton, Faure, and Sobol Sequences

Boyle, Broadie, and Glasserman [1995], Joy, Boyle, and Tan [1995], Papageorgiou and Traub [1996], Paskov [1994], and Paskov and Traub [1995] demonstrate that low-discrepancy sequences are more efficient than random sequences. Bratley, Fox, and Niederreiter [1992] conclude instead that random sequences are more efficient than low-discrepancy sequences for dimensions higher than twelve. The authors disagree on which of the low-discrepancy sequences may be more efficient.

What can we conclude from this research? First, low-discrepancy sequences outperform random sequences for low-dimension simulations. Second, they may or may not be more efficient for higher dimensions. Finally, some low-discrepancy sequences may be more efficient than other low-discrepancy sequences, but this conclusion is by no means clear.

We can shed some light on these disparate results by 1) examining the impact of "high-dimensional clustering" on the Halton, Faure, and Sobol sequences, and 2) applying the sequences to the valuation of several complex derivatives contracts.

To measure the impact of high-dimensional clustering, we examine how each of the three sequences fill an arbitrary two-dimensional hyperplane (unit square) as the number of dimensions increases. Specifically, the ability of low-discrepancy sequences to fill a two-dimensional hyperplane uniformly deteriorates with increasing dimensionality. We expect high-dimensional clustering to have the greatest impact on the Halton sequence, next on the Faure sequence, and then on the Sobol sequence.

We apply a crude metric to measure the adverse effect of clustering as a function of dimensionality. The metric is similar in spirit to "discrepancy," which is a common measure of uniformity used for low-discrepancy sequences.[7]

Using 500 simulations, we generate a set of low-discrepancy numbers in the interval (0, 1) from each of the three low-discrepancy sequences (as in Exhibits 6, 7, 8, and 10). For each sequence, we arbitrarily select two dimensions and plot the corresponding low-discrepancy numbers as *points* on a two-dimensional plot. Because the numbers are in the interval (0, 1), the plots are bounded by the unit square. From these plots, we determine the highest dimension that results in a plot with no more than 30% of its area devoid of points.

We believe the degree of non-uniformity measured by our metric is sufficient for us to conclude that the sequence does not fill the unit hypercube uniformly. Increasing the number of simulations beyond 500 slowly fills in the unit square at roughly the same density as the existing points. Once the points fill the unit square at this density, however, each additional point tends to cluster near an existing point.

Note in Exhibit 11 that the Halton sequence begins to degrade at approximately the fourteenth dimension. The Faure sequence shows degradation at approximately the twenty-fifth dimension (Exhibit 12). Surprisingly, the Sobol sequence does not show any degradation through the 260th dimension (Exhibit 13).[8] (Although not included here, two-dimensional plots for non-consecutive dimensions exhibit similar patterns of low clustering for the Sobol sequence.)

III. EXAMPLES

Plain Vanilla European Options (one dimension)

We begin with a plain vanilla European call option on a non-dividend-paying asset. Because the option follows a path-independent process, we need simulate only the terminal asset price. Independent sim-

EXHIBIT 11
HALTON SEQUENCE
(500 SIMULATIONS, k = 14 DIMENSIONS)

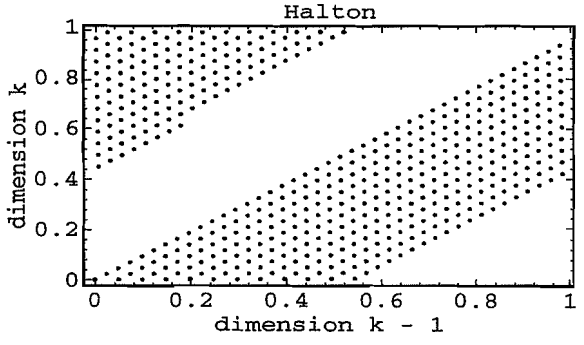
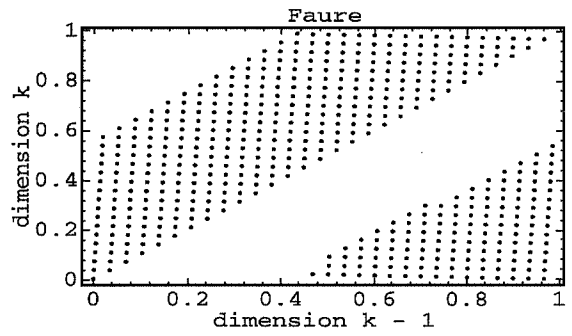


EXHIBIT 12
FAURE SEQUENCE
(500 SIMULATIONS, k = 25 DIMENSIONS)



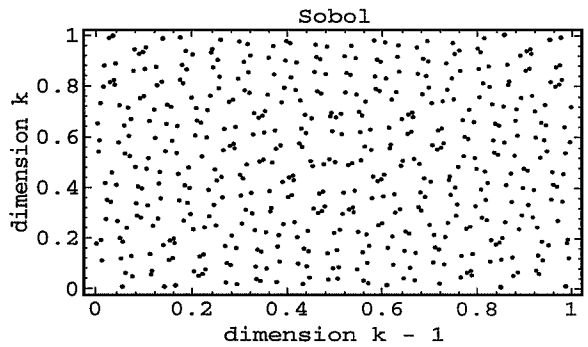
ulations of the terminal date price are generated from

$$S_T^n = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}\varepsilon_n} \tag{16}$$

where n = 1, 2, ..., N; S_0 is the current asset price; and $\varepsilon_n \sim N(0, 1)$. The price of the call with strike price K is

$$\hat{C} = \frac{1}{N} e^{-rT} \sum_{n=1}^{N} \max(0, S_T^n - K) \tag{17}$$

EXHIBIT 13
SOBOL SEQUENCE
(500 SIMULATIONS, k = 260 DIMENSIONS)



We measure the quality of the simulation by calculating the "relative percentage error" with respect to the Black and Scholes price:

$$\hat{e} = \left| \frac{\hat{C} - P_{BS}}{P_{BS}} \right| 100 \tag{18}$$

where P_{BS} is the Black and Scholes price.

The algorithms for the random (ran2) and Sobol sequences are provided by Press et al. [1992], and the Halton and Faure sequences follow from Bratley and Fox [1988]. The uniform [0, 1) sequences are transformed into standard normal sequences [N(0, 1)] using Moro's [1995] inverse normal function (see appendix). All algorithms are compiled on a 133 MHz Pentium in Borland's C++.

Because the additional computing cost is minimal, we implement antithetic variance reduction with ran2. We postpone, for future research, examining the efficacy of using variance reduction techniques on the low-discrepancy sequences.[9]

For a European call option on a non-dividend-paying asset with S_0 = \$100, σ = 20%, K = \$100, r = 10%, and T = 1 year, Exhibit 14 compares the pricing errors obtained from the random sequence labeled ran2 with the Halton, Faure, and Sobol sequences. The graph expands the horizontal scale for the first 25,000 simulations of the plot, which we denote by a vertical dividing line that separates the plot into two regions.

Over the range of 1,000 to 200,000 simulations, the random sequence with antithetic variance reduction is inferior to each of the low-discrepancy sequences. In general, the Sobol sequence outperforms the Faure sequence, and the Faure marginally outperforms the Halton sequence. At 15,000 simulations, the random sequence exhibits an error of 0.07%; the Halton and Faure sequences have errors of 0.1%; and the Sobol sequence has an error of 0.03%. These errors decrease as the number of simulations increases.

Plain Vanilla European Options (first 250 time intervals)

Rather than simulate just the asset's terminal date price, we simulate the entire price path using a discrete approximation. We explore the impact of

EXHIBIT 14
EUROPEAN CALL OPTION (ONE TIME PERIOD)

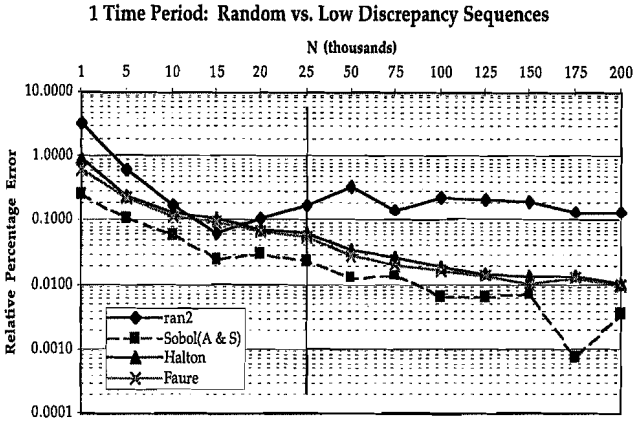
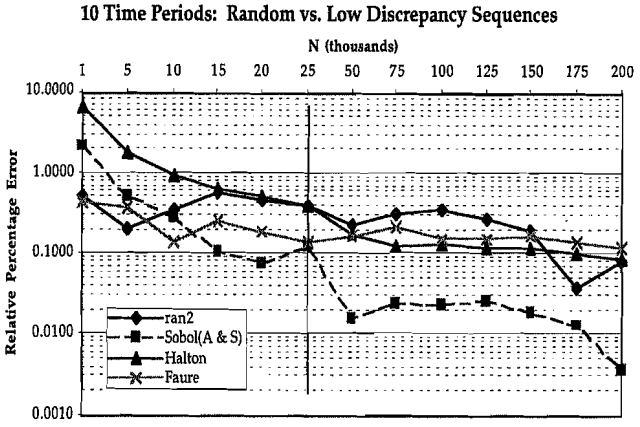


EXHIBIT 15
EUROPEAN CALL OPTION (TEN TIME PERIODS)



increasing dimensionality on the low-discrepancy sequences in two cases: 1) 10 time intervals, and 2) 250 time intervals.

Exhibits 15 and 16 plot the relative percentage errors for the ran2, Halton, Faure, and Sobol sequences. As the number of time intervals increases from 10 to 250, the performance of the low-discrepancy sequences degrades compared to the random sequence.

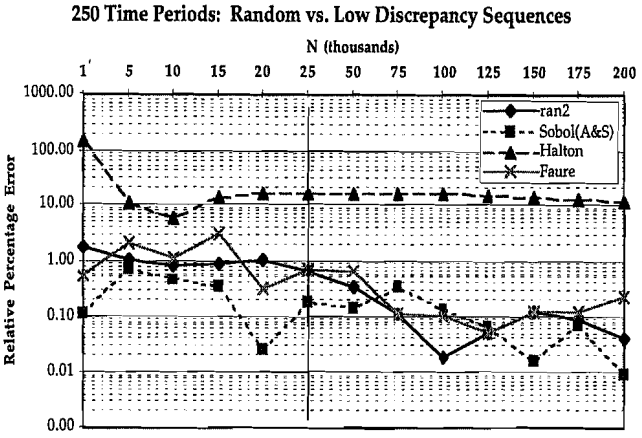
For example, with only 10 time intervals (Exhibit 15), all three low-discrepancy sequences decidedly outperform the random sequences after 10,000 simulations. With 250 time intervals (Exhibit 16), only the Sobol sequence outperforms the random sequence after 10,000 simulations (and up to 75,000 simulations). The latter requires approximately 50,000 simulations to achieve the same level of error as 10,000 to 15,000 Sobol simulations.

Therefore, the advantage of low-discrepancy sequences, in particular, the Sobol sequence, lies in small sample sizes. This is the primary reason for employing low-discrepancy sequences with Monte Carlo simulation.

At 15,000 simulations and 250 dimensions (Exhibit 16), the ran2, Halton, Faure, and Sobol sequences have errors of 0.90%, 13.68%, 3.25%, and 0.34%, and the corresponding computation times are 2.17, 2.43, 4.87, and 2.27 minutes. Additionally, 50,000 simulations of the ran2 sequence, which achieves the same level of error as 15,000 Sobol simulations, take 6.88 minutes. Similar computational times are obtained for the barrier, look-back, and Asian options examined.

Exhibit 17 reinforces these observations by plotting the ran2, Halton, Faure, and Sobol sequences for the first 250 time intervals in increments of ten (time intervals). The plot uses 20,000 simulations for each dimension. All three low-discrepancy sequences outperform the random sequence through the first fifteen time intervals. From the fifteenth to the sixtieth time interval, only the Faure and Sobol sequences outperform the random sequence. In fact, the Halton sequence is inferior to the other sequences after fifteen time intervals.

EXHIBIT 16
EUROPEAN CALL OPTION (250 TIME PERIODS)



Beyond 60 time intervals, the results are mixed. For example, from 60 to 110 time intervals, the random sequence outperforms the Faure and Sobol sequences, but after 110 time intervals the random sequence performs the same as the Faure sequence and is inferior to the Sobol sequence.

Down-and-Out Barrier Call Option (250 time intervals)

Barrier options lie in complexity between plain vanilla European options and plain vanilla American options. They either come into existence or die if the price of the underlying asset crosses a predetermined price barrier during the life of the contract. These options are weakly path-dependent in that the payoff depends not only on the final price of the underlying asset, but also on whether the asset's price crosses the barrier. Some barrier options specify a rebate, usually a fixed amount.

We consider only the down-and-out barrier call option with no rebate. The terminal date cash flow is

$$\begin{aligned} &\text{Max } [0, S_T - K] && \text{if for all } \tau \le T,\ S_\tau > H \\ &0 && \text{if for some } \tau \le T,\ S_\tau \le H \end{aligned} \quad (19)$$

where H is the barrier.

Using 250 time intervals of length $\Delta t = 1/250$, Exhibit 18 plots the relative percentage errors for a down-and-out barrier call option with $S_0 = \$100$, $\sigma = 20\%$, $K = \$100$, $r = 10\%$, barrier = \$90, and $T = 1$ year. The error is calculated using Rubinstein and Reiner's [1991] analytic formula.[10]

Exhibit 18 indicates that the Sobol sequence outperforms the Halton and Faure sequences. We note one anomaly. The ran2 sequence with only 1,000 simulations appears to outperform all the other sequences. We have no explanation for this anomaly.

Up to 50,000 simulations the Faure is inferior to the ran2 sequence. At 15,000 simulations, the ran2, Halton, Faure, and Sobol sequences have errors of 1.85%, 45.57%, 2.45%, and 1.52%, and the corresponding computation times are 1.82, 2.00, 4.53, and 1.62 minutes, respectively. In all cases, the Halton sequence is inferior.

Average Price Asian Call Option (250 time intervals)

The price of European Asian options depends on the average price experienced by the underlying asset during the life of the option. Consequently, Asian options are path-dependent. Asian options use both arithmetic and geometric price averages. We consider only the European geometric average price

EXHIBIT 17
EUROPEAN CALL OPTION (TIME INTERVALS 1 TO 250 WITH 20,000 SIMULATIONS)

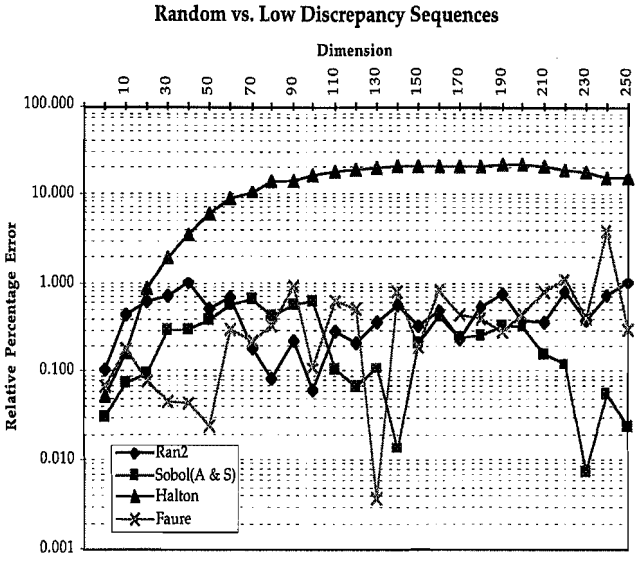
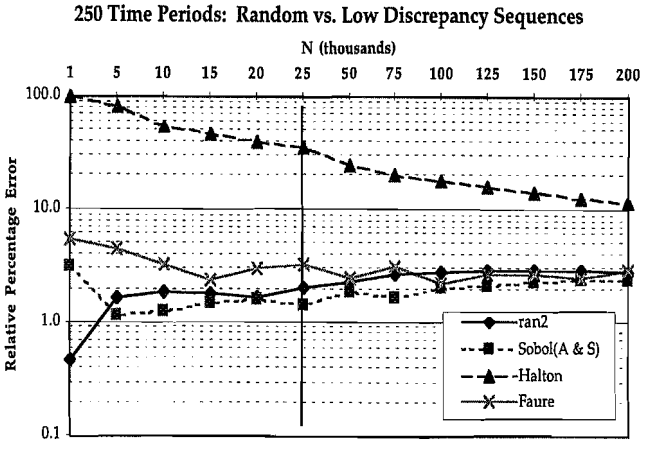


EXHIBIT 18
EUROPEAN DOWN-AND-OUT BARRIER CALL OPTION (250 TIME PERIODS)



Asian call option with terminal payoff:

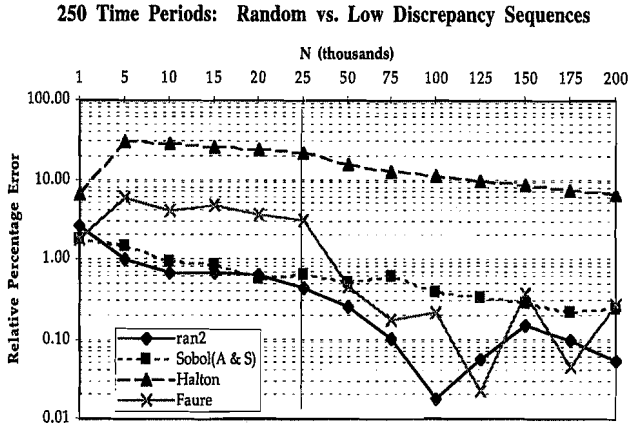
$$\max\left[0, \left(\prod_{i=0}^{T} S_i\right)^{-T-1} - K\right] \quad (20)$$

where S_i for $i = 0, 1, \ldots, T$ is a series of sequentially observed prices of the underlying asset over the life of the option that are used to determine the average price.

Using 250 time intervals of length $\Delta t = 1/250$, Exhibit 19 plots the relative percentage errors for the geometric average price Asian call option with $S_0 = \$100$, $\sigma = 20\%$, $K = \$100$, $r = 10\%$, and $T = 1$ year. The error is calculated using Rubinstein and Reiner's [1991] analytic formula.[11]

As in all the previous examples, the Halton sequence underperforms the random and the other low-discrepancy sequences. The Sobol sequence outperforms the Faure sequence up to 50,000 simulations, but after 50,000 simulations the situation is reversed. At 15,000 simulations, the ran2, Halton, Faure, and Sobol sequences have errors of 0.68%, 26.37%, 4.91%, and 0.86%; the corresponding computation times are 2.60, 2.15, 5.28, and 2.41 minutes. Between 5,000 to 100,000 simulations, the ran2 sequence outperforms the low-discrepancy sequences.

EXHIBIT 19
GEOMETRIC AVERAGE PRICE CALL OPTION
(250 TIME PERIODS)



Look-Back Option Call Option (250 time intervals)

Look-back options also exhibit path-dependence. The price of a European look-back option depends not only on the terminal price of the underlying asset at expiration, but also on either the minimum or maximum price experienced by the asset during the life of the option. The terminal payoff for a European look-back call is

$$\max[0, S_T - \min(S_0, S_1, \ldots, S_T)] =$$

$$S_T - \min(S_0, S_1, \ldots, S_T) \quad (21)$$

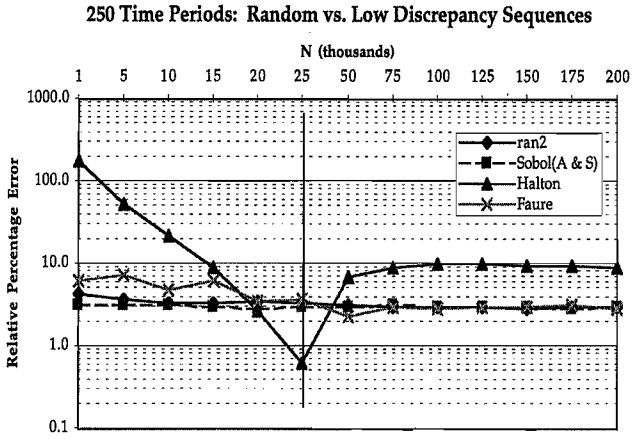
where $S_0, S_1, \ldots, S_T$ is a series of sequentially observed prices of the underlying asset over the life of the option. The right-hand side of the equality exists because it always pays to exercise a look-back option at maturity.

Using 250 time intervals of length $\Delta t = 1/250$, Exhibit 20 plots the relative percentage errors for the look-back call option with $S_0 = \$100$, $\sigma = 20\%$, $K = \$100$, $r = 10\%$, and $T = 1$ year. The error is calculated using Rubinstein and Reiner's [1991] analytic formula.[12]

Again, the Halton sequence substantially underperforms the random and the other low-discrepancy sequences, with one anomaly; it experiences a substantial drop in error at 25,000 simulations. Other than the fact that the algorithms for the low-discrepancy sequences (as well as random sequences) may occasionally produce extraneous errors (e.g., bad numbers), we have no explanation for this decline in error, except that similar drops appear in several of the other graphs.

Up to 50,000 simulations, the Sobol sequence easily outperforms the Faure sequence. At 15,000 simulations, the ran2, Halton, Faure, and Sobol sequences have errors of 3.38%, 8.80%, 6.12%, and 3.06%, with corresponding computation times of 2.05, 2.25, 4.60, and 1.87 minutes. To achieve the same error reduction as a 15,000-point Sobol sequence, approximately 75,000 ran2 simulations (10.38 minutes) are required. After 75,000 simulations, the Faure and Sobol sequences are indistinguishable from the ran2 sequence.

EXHIBIT 20
EUROPEAN LOOK-BACK CALL OPTION (250 TIME PERIODS)



IV. CONCLUSION

As we have verified in the case of the complex derivatives contracts, low-discrepancy sequences for small sample sizes exhibit results consistent with our predictions. Specifically, because of high-dimensional clustering, we expect the Halton sequence to be inferior to the Faure sequence, and the Faure sequence to be inferior to the Sobol.

On average, for the high-dimensional integrals considered, the Sobol sequence exhibits better convergence properties than either the Faure or Halton sequences. The Halton sequence is inferior to the other low-discrepancy sequences as well as the random sequences. For large sample sizes, there is very little distinction between the efficiency of random sequences with antithetic variance reduction and either the Sobol or Faure sequences.

The computation times for the Halton sequence are approximately 15% to 22% longer than the random sequence (15,000 simulations with 250 time intervals). The Faure sequence is significantly slower, taking approximately 137% to 205% longer. With the exception of the plain vanilla call option, the Sobol sequence is approximately 4% to 18% faster. Where the Sobol sequence excels, the random sequence takes approximately 360% to 450% longer to achieve the same level of accuracy.

Depending on the options examined and the corresponding complexity of the integrals involved in the pricing process, low-discrepancy sequences may perform no better than random sequences. In general, for small sample sizes and for the complex options examined here, random sequences with antithetic variance reduction are more efficient than Halton and Faure sequences, but inferior to the Sobol sequence.

APPENDIX
MORO'S ALGORITHM

Moro [1995] provides an algorithm for the inverse normal function (see also Joy, Boyle, and Tan [1996]). The usual Box-Muller algorithm should not be used, because it alters the order of the low-discrepancy sequences. Moro indicates that his algorithm is faster than the usual Box-Muller algorithm and has a maximum error of 3×10^{-9} (see Press et al. [1992]).

Moro's algorithm has high accuracy for all values of the cumulative normal density function $\Phi(x)$ in the interval $10^{-10} \leq \Phi(x) \leq 1 - 10^{-10}$, where $\Phi(x)$ is given by

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt$$

Following the discussion in Joy, Boyle, and Tan [1996], Moro's algorithm employs two approximations: 1) one for the center of the distribution (i.e., $0.08 \leq \Phi(x) \leq 0.92$), and 2) another for the tails of the distribution (i.e., $\Phi(x) < 0.08$ and $\Phi(x) > 0.92$). For 1) the center of the distribution, the inverse normal function $\Phi^{-1}(x)$ is given by the rational approximation:

$$\Phi^{-1}(x) = y \frac{\sum_{n=0}^{3} a_n y^{2n}}{\sum_{n=0}^{4} b_n y^{2n}} \quad \text{if } |y| \leq 0.42$$

where $y = \Phi(x) - 0.5$, and the constants a_n and b_n are given in Exhibit 21. For 2) the tails of the distribution (i.e., $|y| > 0.42$), $\Phi^{-1}(x)$ is given by the truncated Chebyschev series:

$$\Phi^{-1}(x) = \begin{cases} \sum_{n=0}^{8} c_n T_n(z) - \frac{c_0}{2} & \text{if } 0.42 < y < 0.5 \\ \frac{c_0}{2} - \sum_{n=0}^{8} c_n T_n(z) & \text{if } -0.5 < y < -0.42 \end{cases}$$

EXHIBIT 21
CONSTANTS FOR MORO'S INVERSE NORMAL FUNCTION

n	a_n	b_n	n	c_n
0	2.50662823884	1.00	0	7.7108870705487895
1	–18.61500062529	–8.47351093090	1	2.7772013533685169
2	41.39119773534	23.08336743743	2	0.3614964129261002
3	–25.44106049637	–21.06224101826	3	0.0373418233434554
4		3.13082909833	4	0.0028297143036967
			5	0.0001625716917922
			6	0.0000080173304740
	k_1	k_2	7	0.0000003840919865
	0.4179886424926431	4.2454686881376569	8	0.0000000129707170

where $z = k_1[2\ln(-\ln[0.5 - |y|]) - k_2]$, and the constants c_n, k_1, and k_2 are given in Exhibit 21. Clenshaw's recurrence formula efficiently approximates the truncated Chebyschev series (see Press et al. [1992, p. 193]).

If

$$f(z) = \sum_{n=0}^{8} c_n T_n(z) - \frac{c_0}{2}$$

then f(z) is given by

$$d_{10} \equiv d_9 \equiv 0$$

$$d_j = 2zd_{j+1} - d_{j+2} + c_j \qquad j = 8, 7, \ldots, 1$$

$$f(z) = d_0 = zd_1 - d_2 + 1/2c_0$$

ENDNOTES

The authors would like to thank Martin W. Fong for his invaluable assistance.

A significant portion of this article was completed while Mr. Galanti was a candidate for the Master of Business Administration degree at the College of Business at San Francisco State University.

[1]For more information, see Bratley, Fox, and Schrage [1987], Boyle [1977], or Niederreiter [1992].

[2]For an excellent overview, see chapter 2 in Bratley, Fox, and Schrage [1987]; or Boyle [1977] and Boyle, Broadie, and Glasserman [1995] for applications to financial problems.

[3]The random numbers in the interval (0, 1) are obtained using the ran2 algorithm from Press et al. [1992]. It is based on the linear congruential method, and the authors indicate that the generator produces "perfect" random numbers (see p. 281).

[4]The term, "speed," refers to the number of nonnegative consecutive integers n = 0, 1, 2, ..., that is required to fill the interval [0, 1) uniformly. Higher speeds require fewer integers.

[5]Bratley and Fox [1988], Morokoff and Caflisch [1995], and Boyle, Broadie, and Glasserman [1995].

[6]The term, "primitive," implies that the polynomials cannot be factored into a polynomial of lower degree using modulo 2 integer arithmetic. Press et al. [1992] or Peterson and Weldon [1972] provide an extensive list of primitive polynomials modulo 2, but some examples are:

Degree	Polynomial
1	$x + 1$
2	$x^2 + x + 1$
3	$x^3 + x + 1$
3	$x^3 + x^2 + 1$
4	$x^4 + x + 1$

[7]Let A equal a rectangular region in the unit hypercube I^s with sides parallel to the coordinate axes, and let V(A) be its volume. The discrepancy of the sequence $x_1, \ldots, x_N$ of N points is

$$D_N = \sup_{A \in I^s} \left| \frac{\#\text{ of points in A}}{N} - V(A) \right|$$

Discrepancy provides a measure of the average distance between points in the unit hypercube. When the average is large, discrepancy is small. See Niederreiter [1992] for a more detailed discussion.

[8]Our code can generate Sobol sequences up to 260 dimensions.

[9]In general, variance reduction methods should improve the performance of low-discrepancy sequences, provided they do not alter the order of the sequences. Because the control variate method has no effect on the sequences, it should improve the performance of the simulation. Moskowitz and Caflisch [1995] demonstrate that importance sampling also improves performance. Joy, Boyle, and Tau [1995], however, state that antithetic variables and stratified sampling alter the order of the low-discrepancy sequences, and hence may not improve the performance of the simulation.

[10]From Rubinstein and Reiner [1991] for continuously observed prices, the price of a down-and-out call option with no rebate is

$$C_{K>H} = [1] - [3] \qquad \text{if } K > H$$
$$C_{K<H} = [2] - [4] \qquad \text{if } K < H$$

where H is the barrier and

$$[1] = SN(x) - Kr^{-T}N(x - \sigma\sqrt{T})$$
$$[2] = SN(x_1) - Kr^{-T}N(x_1 - \sigma\sqrt{T})$$
$$[3] = S\left(\frac{H}{S}\right)^{2\lambda} N(y) - Kr^{-T}\left(\frac{H}{S}\right)^{2\lambda-2} N(y - \sigma\sqrt{T})$$
$$[4] = S\left(\frac{H}{S}\right)^{2\lambda} N(y_1) - Kr^{-T}\left(\frac{H}{S}\right)^{2\lambda-2} N(y_1 - \sigma\sqrt{T})$$

$$x = \frac{\ln(S/K)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}, \qquad x_1 = \frac{\ln(S/H)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}$$
$$y = \frac{\ln(H^2/SK)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}, \qquad y_1 = \frac{\ln(H/S)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}$$
$$\lambda = 1 + \frac{\mu}{\sigma^2}, \qquad \mu = \ln(r) - \frac{1}{2}\sigma^2$$

[11]From Rubinstein and Reiner [1991] for continuous sampling of the underlying asset price, the price of a geometric average price Asian call option is

$$C = Sr^{-T/2}e^{-\sigma^2 T/12}N(x) - Kr^{-T}N(x - \sigma\sqrt{T/3})$$
$$x = \frac{\ln(S/Kr^{-T/2})}{\sigma\sqrt{T/3}} + (1/4)\sigma\sqrt{T/3}$$

[12]From Rubinstein and Reiner [1991] for continuously observed prices, the price of a look-back call is

$$C = S - Mr^{-T}N\left[\frac{b + \mu T}{\sigma\sqrt{T}}\right] + Mr^{-T}\lambda e^{b\left(1-\frac{1}{\lambda}\right)}N\left[\frac{-b + \mu T}{\sigma\sqrt{T}}\right] - S(1 + \lambda)N\left[\frac{-b - \mu T - \sigma^2 T}{\sigma\sqrt{T}}\right]$$

where $b = \ln\left[\frac{S}{M}\right]$, $\mu = \ln(r) - (\sigma^2/2)$, $\lambda = \frac{\sigma^2/2}{\ln(r)}$, and $M \leq S$ is the current minimum price that the underlying asset has experienced during the life of the option.

REFERENCES

Antonov, I.A., and V.M. Saleev. "An Economic Method of Computing LP_T-sequences." *USSR Comput. Math. Phys.*, 19 (1979), pp. 252-256.

Boyle, P. "Options: A Monte Carlo Approach." *Journal of Financial Economics*, Vol. 4 (1977), pp. 323-338.

Boyle, P., M. Broadie, and P. Glasserman. "Monte Carlo Methods for Security Pricing." *Journal of Economic Dynamics and Control*, 1995.

Bratley, P., and B.L. Fox. "Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator." *ACM Transactions on Mathematical Software*, Vol. 14, No. 1 (1988), pp. 88-100.

Bratley, P., B.L. Fox, and H. Niederreiter. "Implementation and Tests of Low-Discrepancy Sequences." *ACM Transactions on Modeling and Computer Simulation*, Vol. 2, No. 3 (1992), pp. 195-213.

Bratley, P., B.L. Fox, and L. Schrage. *A Guide to Simulation*, 2nd edition. Amsterdam: Springer Verlag, 1987.

Halton, J.H. "On the Efficiency of Certain Quasi-Random Sequences of Points in Evaluating Multi-Dimensional Integrals." *Numerische Mathematik*, Vol. 2 (1960), pp. 84-90.

Joy, C., P. Boyle, and K.S. Tan. "Quasi Monte Carlo Methods in Numerical Finance." *Management Science*, Vol. 42, No. 6 (1996), pp. 926-936.

Moro, B. "The Full Monte." *Risk*, Vol. 8, No. 2 (February 1995), pp. 57-58.

Morokoff, W., and R. Caflisch. "Quasi-Monte Carlo Integration." *Journal of Computational Physics*, 122 (1995), pp. 218-230.

Moskowitz, B., and R. Caflisch. "Smoothness and Dimension Reduction in Quasi-Monte Carlo Methods." *Math. Comput. Modeling*, Vol. 23, No. 8/9 (1995), pp. 37-54.

Niederreiter, H. *Random Number Generation and Quasi-*

Monte Carlo Methods. CBMS-NSF, Vol. 63. Philadelphia: SIAM, 1992.

Papageorgiou, A., and J. Traub. "Beating Monte Carlo." *Risk*, Vol. 9, No. 6 (June 1996), pp. 63-65.

Paskov, S. "Computing High Dimensional Integrals with Applications in Finance." Technical Report CUCS-023-94, Department of Computer Science, Columbia University, 1994.

Paskov, S., and J. Traub. "Faster Valuation of Financial Derivatives." *Journal of Portfolio Management*, Fall 1995, pp. 113-120.

Peterson, W.W., and E.J. Weldon. *Error Correcting Codes*, 2nd edition. Cambridge, MA: MIT Press, 1972.

Press, W., S. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edition. Cambridge: Cambridge University Press, 1992.

Rubinstein, M., and E. Reiner. "Exotic Options." Working paper, Haas School of Business, University of California at Berkeley, 1991.