

RANDOM NUMBER GENERATORS

Sorin R. Straja, Ph.D., FRM

Montgomery Investment Technology, Inc.

200 Federal Street

Camden, NJ 08103

Phone: (610) 688-8111

sorin.straja@fintools.com

www.fintools.com

Uniform deviates are random numbers that are uniformly distributed between 0 and 1. The other random numbers are usually generated through transformations of the uniform deviates. Most efforts regarding the design of the random number generators are focused on uniform deviates.

“In IBM’s early computing days, it used to deliver its mainframe system equipped with its infamous RANDU generator [...]. This generator has meanwhile repeatedly been reported to be highly inadequate [...]. Sadly, it has been copied to a variety of other company systems, including even the Russian ES system [...]. Since it is well conceivable that other computer manufacturers have slipped up similarly, and since there is according to Murphy’s law a tendency for mistakes to proliferate, I advise the reader never to rely on black box number generators that come with any one system and allegedly have been tested.” (Jäckel 2002)

“The C++ standard library provides a facility for generating random numbers using the function rand(). [...] The version of rand() that comes with your C++ compiler will in all probability be a pretty simple generator and wouldn’t be appropriate for scientific use but it can still be used for some purposes. It may well be random enough for use in simple programs and games. If you want something a little better, you’ll have to look beyond the C++ standard library but for now we’ll concern ourselves with the functions C++ provides.” (Jacobs 2004).

Park and Miller (1988) have surveyed a large number of random number generators for uniform deviates that have been used starting with the 1950s. The simple multiplicative congruential algorithm

$$I_{j+1} = a I_j \pmod{m}$$

is competitive with any more general linear congruential generators if the multiplier **a** and the modulus **m** are chosen adequately. Park and Miller (1988) selected the “Minimal Standard” generator with

$$a = 7^5 = 16,807 \quad \text{and} \quad m = 2^{31} - 1 = 2,147,483,647$$

It may not be possible to implement it directly in a high-level language because the products required may exceed the maximum value for an integer. Assembly language implementation is straightforward, but it is not portable from machine to machine. The Schrage algorithm (Schrage 1979; Bratley et al. 1983), based on an approximate factorization of **m**, allows the “Minimal Standard” algorithm to be implemented in high-level programming languages without being



machine specific:

$$m = a q + r, \text{ or } q = [m/a], r = m \bmod a$$

with $[\]$ denoting integer part. If r is small, $r < q$, and $0 < z < (m-1)$, then

$$a (z \bmod m) = a (z \bmod q) - r [z/q] \quad \text{if } a (z \bmod q) - r [z/q] \geq 0$$

and

$$a (z \bmod m) = a (z \bmod q) - r [z/q] + m \quad \text{if } a (z \bmod q) - r [z/q] < 0.$$

For the “Minimal Standard” algorithm selected values are:

$$q = 127,773 \text{ and } r = 2,836.$$

For uniform deviates, we selected to use a portable random number generator that relies on the “Minimal Standard” algorithm, but shuffles the output to remove low-order serial correlations. A random deviate derived from the j th value in the sequence, I_j , is output not on the j th call, but rather on a randomized later call, $j+32$ on average (Bays and Durham 1976; Knuth 1981). This random number generator, labeled Ran1 by Press et al. (1992), apparently passes all known tests except when the number of calls approaches the order of its period (i.e., 2,147,483,647) (Press et al. 1992).

For simulations requiring even longer random sequences, L’Ecuyer (1988) recommended to combine two different sequences with different periods obtaining a new sequence whose period is the least common multiple of the two original periods. Press et al. (1992) consider this one to be the perfect random number generator. A practical definition of “perfect” is that they “*will pay \$1000 to the first reader who convinces us otherwise*” (Press et al. 1992). However, balancing speed and length period, Press et al. (1992) recommend using the Bays and Durham (1976) and Knuth (1981) algorithm, unless there is a need for very long random sequences when the L’Ecuyer (1988) approach is required. Jäckel (2002) endorses this recommendation.

For the Gaussian random numbers with zero mean and unit variance we decided to couple the Ran1 algorithm described above with the Box and Muller (1958) method. The polar form of the Box-Muller transformation is numerically fast and robust because it does the equivalent of the sine and cosine geometrically without a call to the trigonometric function library. Like other transformations, the Box and Muller method requires a good quality uniform random number generator.

More details regarding the application of the Monte Carlo simulation to option pricing are provided by Montgomery and Pizzi (1998).



REFERENCES

- Bays, C.; Durham, S. "Improving Poor Random Number Generator," *ACM Transaction on Mathematical Software* **2** (1): 59-64; 1976.
- Bratley, P.; Fox, B. L.; Schrage, E. L. *A Guide to Simulation*. New York, NY: Springer Verlag; 1983.
- Box, G. E. P; Muller, M. E. "A Note on the Generation of Random Normal Deviates," *Annals of Mathematical Statistics* **29**: 610-611; 1958.
- Jäckel, P. *Monte Carlo Methods in Finance*. Chichester: John Wiley and Sons; 2002.
- Jacobs, B. *C++ Random Numbers. Introducing the functions srand() and rand()*. Revised 5th May, 2004. <http://www.robertjacobs.fsnet.co.uk/random.htm>; Accessed on July 14, 2004.
- Knuth, D. E. *The Art of Computer Programming. Vol. 2. Seminumerical Algorithms*. 2nd Edition. Reading, MA: Addison-Wesley; 1981.
- L'Ecuyer P. "An Efficient and Portable Combined Random Number Generator," *Communications of the ACM* **31** (6): 742-749; 1988.
- Montgomery, G. L.; Pizzi, M. A. "The pricing of path-dependent European options via Monte Carlo simulation", *Technology* **335A**(1): 57-63; 1998.
- Park, S. K.; Miller, K. W. "Random number generators: good ones are hard to find", *Communications of the ACM* **31** (10): 1192-1201; 1988.
- Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. *Numerical Recipes in FORTRAN. The Art of Scientific Computing*. 2nd Edition. Cambridge: Cambridge University Press; 1992.
- Schrage, L. *A More Portable FORTRAN Random Number Generator*. *ACM Transactions on Mathematical Software* **5** (2): 132-138; 1979.

