

Random number generators

Random number generators (RNGs) serve many important uses in statistics: bootstrapping, **Monte Carlo, simulation**, and **numerical integration** of distribution functions are just a few examples. In most applications the random numbers are generated from an algorithm and so are technically called *quasi-* or *pseudo-*random numbers, because such numbers are not random at all, but deterministic. However, what matters in applications is that the numbers appear to be random. Therefore, as long as the pseudo-random numbers appear to be random, the fact that they are calculated deterministically is largely irrelevant.

The field of random number generation was born with the publication of Metropolis and Ulam [24]. Shortly thereafter, Lehmer [18] proposed the first linear congruential random number generator (LCRNG). With slight modifications by Thomson [31] and (independently) by Rotenberg [28], it became and still is the best known (and most studied) form of RNG. The LCRNG has the general form

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0 \quad (1)$$

where X_0 is the *seed* used to initialize the sequence, m is the modulus ($0 \leq m$), a is the *multiplier* ($0 \leq a < m$), and c is the *increment* ($0 \leq c < m$). Lehmer's original formulation had $c = 0$; the generalization to $c \neq 0$ was the contribution of Thomson and Rotenberg.

Beginning with the seed X_0 , the LCRNG is 'called' to produce X_1, X_2, \dots , a sequence of integers with each X_n less than m (conversion to a random uniform sequence, the usual output of an RNG, is effected by dividing X_n by m). Clearly, there can be no more than m distinct random numbers generated, so if the RNG is called N times and $N > m$, then at some point $X_{n+1} = X_0$ and the sequence repeats itself. The number of calls that can be made until the sequence repeats is the *period* of the RNG, denoted p ($p \leq m$). It is desirable that m be large so that p can be large. In fact, L'Ecuyer [11] prescribes that 'No generator should be used for any serious purpose if its period (or, at least, a lower bound on it) is unknown'. Moreover, only a fraction of the period should be used, because the discrepancy between the RNG's output and

true randomness increases as N approaches m [14, section 4.2.3]. Ripley [26, p. 26] suggests that p be greater than N^2 . Knuth [10, p. 195] suggests a more modest $p < N/1000$. Most PCs have 32-bit word length, and so implement RNGs that have period no larger than 2^{32} . Combining words so that the period can be increased makes the RNG run more slowly – an important consideration for simulation studies. Consequently, many PC software packages implement RNGs with a period length approximately $2^{31} \approx 2.1$ billion. Following Knuth's suggestion, at most 2.1 million calls should be made to such an RNG; all but the smallest simulation studies will exceed this number.

While the LCRNG has the advantage that it is very fast, it also produces output such that successive calls are correlated [3, 21]. If k successive random numbers are used to fill a k -dimensional hypercube (e.g. pairs of numbers to fill a square, triples of numbers to fill a cube, etc.), then the points will not fill the k -dimensional space but will lie on $(k - 1)$ planes in the space. If m, a and c are well chosen, then there will be about $m^{1/k}$ such planes.

Three more points about (1) merit mention. First, the period will be of length m only if m, a and c are chosen carefully. Poor choices for these parameters can produce a period that is much shorter than m . Second, the seed determines the sequence of random numbers. In particular, the same seed produces the same sequence, so the LCRNG is *reproducible*. Third, poor choices for these parameters can yield an LCRNG whose output is not very random. These choices have been poorly made in many RNGs that are part of operating systems [25, 29]. In fact, Knuth [10, p. 193] advises that system-supplied RNGs never be used and warns that, should a user examine such RNGs 'Try to avoid being too shocked at what you find'.

In addition to the LCRNG, there are many other types, including *multiple recursive generators, combined generators, multiply-with-carry*, and the *Tausworthe generator*, which are all linear generators. See [13, Section 4.3] for more details. There are also nonlinear generators, such as the *inversive congruential generators* and *quadratic congruential generators*. See [13, Section 4.4] for more details.

According to Ripley [27], a good RNG should:

1. be reproducible from a simply specified starting point;

2 Random number generators

2. have a very long period;
3. produce numbers that are a very good approximation to a uniform distribution;
4. produce numbers that are very close to independent in a moderate number of dimensions.

Points 1 and 2 have already been discussed. Points 3 and 4 suggest two useful directions for testing RNGs: test that the output is uniform, and test that successive calls are uncorrelated. There are two approaches to such testing: theoretical and empirical. See the classical reference by Knuth [10, Section 3.3] for detailed treatment of each approach. L'Ecuyer [14] argues that the construction of RNGs and the selection of their parameters should be guided by theory, but that empirical testing also is important. Discussions of empirical tests can be found in [16] and references cited therein. Gentle [8] also provides a useful overview of the subject.

The first comprehensive battery of tests was proposed by an early edition of [10] – see Dwyer and Williams [6] for implementations in the C language. These tests were designed for an era when perhaps only millions of random numbers were needed. Marsaglia [22] noted that these tests were not sufficiently stringent, and so offered the DIEHARD program [23] that implements many stringent tests of randomness. There exist RNGs that pass all the Knuth tests, yet do not pass all the DIEHARD tests. More ambitious batteries of tests than DIEHARD are being developed currently, but release dates for these programs are unknown.

No RNG will pass all empirical tests, and for any RNG a test can be constructed such that the RNG will fail. That is, no amount of empirical testing can prove that a particular RNG is perfect. L'Ecuyer [15] advises that ‘a *bad* RNG is one that fails *simple* tests, and a *good* RNG is one that fails only complicated tests’. Such testing is extremely difficult, and the researcher who does not wish to delve too far into this topic is advised to use RNGs that have already been tested by specialists.

Frequently, a researcher will need a random number drawn from a distribution that is not uniform, e.g. the exponential or normal distributions. As a general rule, there is no RNG that produces such numbers directly. Rather, the uniform RNG is used as the basis for producing these other RNGs, two common methods being the *transformation method* (also called the *inversion method*) and the *rejection method*. The

classical reference [4] is out of print and no in-print monograph is its equal. However, see the relevant chapter in [7].

The transformation method is based on the probability integral transform. Heuristically, suppose that a random quantity X with cumulative distribution function $F(x)$ is desired. Let U have the uniform distribution. Suppose that $F(x)$ can be inverted so that $x = F^{-1}(y)$, $0 < y < 1$. Then a random quantity X with distribution $F(x)$ can be computed by $X = F^{-1}(U)$. This method will work for the exponential [$F(x) = e^{-x}$], logistic [$F(x) = 1/(1 + e^{-x})$], and many other distributions. See [5].

For many distributions, no such inverse exists. In the case of the normal distribution, the *polar method* [1, 20] can be employed. As described by Knuth [10, Algorithm P, p. 122] the method has four steps, and actually generates a pair of independent normal variates, X_1 and X_2 .

Step 1: generate $V_1 = 2U_1 - 1$ and $V_2 = 2U_2 - 1$, where U_1 and U_2 are uniform on the interval $(0,1)$ and V_1 and V_2 are uniform on the interval $(-1, 1)$.

Step 2: compute $S = V_1^2 + V_2^2$.

Step 3: if $S \geq 1$, then return to the first step.

Step 4: compute $X_1 = V_1(-2 \ln S/S)^{\frac{1}{2}}$ and $X_2 = V_2(-2 \ln S/S)^{\frac{1}{2}}$.

This is a specific example of a more general technique known as the *rejection method*, in which the desired distribution is bounded by a known distribution for which it is easy to generate random numbers. Steps 1–3 enclose a circle with a square and, if the variates V_1 and V_2 do not fall within the circle, they are *rejected*. If they do fall within the circle, then they are used to form the normal variates. See [10, Section 3.4] and references cited therein for further discussion of the rejection method.

Coveyou [2] entitled an article ‘Random number generation is too important to be left to chance’, and every user of RNGs should take these words to heart. Perhaps because researchers are unaware that their results have been corrupted, published accounts of bad RNGs are all too rare. However, in one case it was observed that simulations for pricing financial derivatives produced an incorrect price; Tajima et al. [30] traced the error to the RNG. Also, MacKinnon [19] describes how a period-deficient RNG ruined a Monte Carlo study. Bad RNGs are still

proposed in journals [12], and some bad RNGs are in widespread use [17]. To assist users in choosing from among the many possible RNGs, some of them good and some of them bad, Hellekalek [9] offers a concise survey of how to find good RNGs, as well as a checklist for assessing RNGs. At the very least, extending Knuth's above-mentioned admonition from system-supplied to package-supplied RNGs: if the developer of a software package does not supply evidence that the RNG offered by the package meets Ripley's desiderata, then do not use it.

References

- [1] Box, G. & Muller, M. (1958). A note on the generation of random normal deviates, *Annals of Mathematical Statistics* **29**, 610–611.
- [2] Coveyou, R. (1970). Random number generation is too important to be left to chance, *Studies in Applied Mathematics* **3**, 70–111.
- [3] Coveyou, R. & MacPherson, R. (1967). Fourier analysis of uniform random number generators, *Journal of the ACM* **14**, 100–119.
- [4] Devroye, L. (1986). *Non-uniform Random Variate Generation*, Springer-Verlag, New York.
- [5] Devroye, L. (1996). Random variate generation in one line of code, in *1996 Winter Simulation Conference Proceedings*, J.M. Charnes, D.J. Morrice, D.T. Brunner & J.J. Swain, eds, IEEE, Piscataway, pp. 265–272.
- [6] Dwyer, J. & Williams, K.B. (1996). Testing random number generators, *C/C++ Users Journal* **June**, 39–48.
- [7] Fishman, G. (1996). *Monte Carlo: Concepts, Algorithms and Applications*, Springer-Verlag, New York.
- [8] Gentle, J.E. (1998). *Random Number Generation and Monte Carlo Methods*, Springer-Verlag, New York.
- [9] Hellekalek, P. (1998). Good random number generators are (not so) easy to find, *Mathematics and Computers in Simulation* **46**, 484–505.
- [10] Knuth, D.E. (1998). *The Art of Computer Programming, 3e*, Addison-Wesley, Reading.
- [11] L'Ecuyer, P. (1992). Testing random number generators, in *Proceedings of the 1992 Winter Simulation Conference*, J.J. Swain, D. Goldsmith, R.C. Crain & J.R. Wilson, eds, IEEE Press, New York, pp. 305–313.
- [12] L'Ecuyer, P. (1994). Uniform random number generation, *Annals of Operations Research* **53**, 77–120.
- [13] L'Ecuyer, P. (1997). Random number generation, in *Handbook on Simulation*, chapter 4, J. Banks, ed., Wiley, New York.
- [14] L'Ecuyer, P. (1998). Random number generators and empirical tests, in *Monte Carlo and Quasi-Monte Carlo Methods 1996*, Lecture Notes in Statistics, no. 127, Springer, New York, pp. 124–138.
- [15] L'Ecuyer, P. (1999). Uniform random number generators, in *Proceedings of the 1998 Winter Simulation Conference*, IEEE, Piscataway, pp. 97–104.
- [16] L'Ecuyer, P. & Hellekalek, P. (1998). Random number generators: selection criteria and testing, in *Random and Quasi-Random Point Sets*, Lecture Notes in Statistics, no. 138, Springer-Verlag, New York, pp. 223–266.
- [17] L'Ecuyer, P. & Simard P. (1999). Beware of linear congruential generators with multipliers of the form $a = +/ - 2^q + / - 2^r$, *ACM Transactions on Mathematical Software* **25**, 367–374.
- [18] Lehmer, D.H. (1951). *Mathematical Methods in Large-Scale Computing Units*, Harvard University Press, Cambridge.
- [19] MacKinnon, J.G. (2000). Computing numerical distribution functions in econometrics, in *High Performance Computing Systems and Applications*, A. Pollard, D. Mewhort & D. Weaver, eds, Kluwer, Dordrecht, pp. 455–470.
- [20] Marsaglia, G. (1962). Improving the polar method for generating a pair of normal random variables, Boeing Scientific Laboratory Report D1-82–0203.
- [21] Marsaglia, G. (1968). Random numbers fall mainly in the planes, *Proceedings of the National Academy of Sciences* **60**, 25–28.
- [22] Marsaglia, G. (1985). A current view of random number generators, in *Proceedings of the 16th Symposium on the Interface*, L. Billard, ed., Elsevier, New York, pp. 3–10.
- [23] Marsaglia, G. (1996). DIEHARD: A battery of tests of randomness. <http://stat.fsu.edu/pub/diehard>
- [24] Metropolis, N. & Ulam, S. (1949). The Monte Carlo method, *Journal of the American Statistical Association* **44**, 335–341.
- [25] Park, S.K. & Miller, K.W. (1988). Random number generators: good ones are hard to find, *Communications of the ACM* **31**, 1192–1201.
- [26] Ripley, B.D. (1987). *Stochastic Simulation*, Wiley, New York.
- [27] Ripley, B.D. (1990). Thoughts on pseudorandom number generators, *Journal of Computational and Applied Mathematics* **31**, 153–163.
- [28] Rotenberg, A. (1960). A new pseudo-random number generator, *Journal of the ACM* **7**, 75–77.
- [29] Sawitzki, G. (1985). Another random number generator which should be avoided, *Statistical Software Newsletter* **11**, 81–82.
- [30] Tajima, A., Ninomiya, S. & Tezuka, S. (1997). On the anomaly of RAN1() in Monte Carlo pricing of financial derivatives, in *1996 Winter Simulation Conference Proceedings*, J. Charnes, D. Morrice, D. Brunner & J. Swaine, eds, IEEE, Piscataway, pp. 360–366.
- [31] Thomson, W.E. (1958). A modified congruence method for generating pseudo-random numbers, *Computer Journal* **1**, 83–86.

B.D. MCCULLOUGH