Stochastic Simulation
Brian D. Ripley

CHAPTER 2

Pseudo-Random Numbers

Almost all the simulation methods and algorithms to be discussed in later chapters derive their randomness from an infinite supply $U_0, U_1, U_2, \ldots$ of *random numbers*; that is, an independent sequence (U_i) of random variables uniformly distributed on (0, 1). Many users of simulation are content to remain ignorant of how such numbers are produced, merely calling standard functions to produce them. Such attitudes are dangerous, for random numbers are the foundations of our simulation edifice, and problems at higher levels are frequently traced back to faulty foundations.

This chapter is of rather specialized appeal. Do not yield to the temptation to skip it without working exercise 2.17, for many of the random number generators in use (at the time of writing) have serious defects.

2.1. HISTORY AND PHILOSOPHY

There is no mathematical problem with random numbers: their existence is provable from Kolmogorov's axioms for probability. [See, for example, Neveu (1965, Section 5.1).] However, this result does not produce a realization of a sequence of random numbers for us to use; we have to find some observable process of which the mathematics is a reasonable model. The philosophical problem hinges on that much-abused word "reasonable." How can we decide from a finite sequence $(U_1, \ldots, U_n)$ whether (U_i) is an adequate model? We have immediately all the philosophical problems of statistical inference.

The earliest users of simulation used physical processes which were accepted as random. Most readers will have performed experiments on tossing coins or throwing dice when learning about probability. Such simulation experiments have a long history. A more sophisticated variant from the 18th century is Buffon's needle experiment to estimate π. (See also Section 7.5.) Mechanical devices are still widely used in gambling (dice, roulette wheels) and in lotteries [see West (1955) and Inoue et al. (1983)].

Tippett (1927) produced a table of 40,000 digits "taken at random from census reports." Later tables, such as the RAND (1955) one of a million digits, were produced from electronic noise, which is also the random input to the British "Premium Bond" draw (Thompson, 1959).

All these physical methods have been widely accepted as random, presumably on the basis of observation and explicit or implicit testing. However, many of them have been found to exhibit biases and dependencies. In the case of the RAND machine there were mechanical faults in the recording mechanism that marred the randomness suggested by the theory of electronics (Hacking, 1965, p. 129). Thus even physical devices need to be tested.

Simulation was one of the earliest uses of electronic computers. The pioneers of computing the 1940s found that physical devices did not mesh well with digital computers. Even when tables of random numbers were available on punched cards or tape, they were too slow and cumbersome. So they looked for simple ways to produce haphazard sequences, and considered various nonlinear recursive schemes. One of the earliest was von Neumann's "middle square" method. Suppose we want a sequence of four-digit decimal numbers. Starting from 8653 we square it (74874409) and extract the middle four digits, 8744. This can be repeated to obtain

$$8653, 8744, 4575, 9306, 6016, 1922, 6940, \ldots \tag{1}$$

a deterministic sequence that appears random. Hence, the terminology of *pseudo-random* (*pseudo-*: false, apparent, supposed but not real—*Concise Oxford Dictionary*) or *quasi-random* (*quasi-*: seeming, not real, half-, almost-—*Concise Oxford Dictionary*). We give a formal definition.

Definition: A sequence of *pseudo-random* numbers (U_i) is a deterministic sequence of numbers in $[0, 1]$ having the same relevant statistical properties as a sequence of random numbers.

This needs clarifying by specifying which properties are relevant and statistical. Informally, what is meant is that any statistical test applied to a finite part of (U_i) which aims to detect relevant departures from randomness would not reject the null hypothesis. In practice it seems sufficient to insist that the joint distributions of $(U_{i+1}, \ldots, U_{i+k})$ are not far from uniformity in $[0, 1]^k$ for small values of k (say, $k \leqslant 6$).

One of the most appealing ways of viewing this definition is in terms of *predictability*. In everyday speech we call things "random" if we cannot predict them. For example, one would quickly reject the output of the algorithm

$$U_i = (U_{i-1} + U_{i-2}) \bmod 1 \tag{2}$$

given in Table 2.1 when one notices that U_i never lies in (U_{i-2}, U_{i-1}) and so can be predicted to some extent. The middle square example looks unpredictable for a while, then settles down to

$$2100, 4100, 8100, 6100, 2100, \ldots$$

What we need are sequences that are hard to predict unless the mechanism generating them is known.

This introduces a connection with *cryptography*, the art or science of turning meaningful sequences into apparently random noise in such a way that a key-holder can recover the original data. The author's first acquaintance with pseudo-random numbers came in this way. A sonar device was to be constructed using pseudo-random acoustical noise. The pseudo-randomness made it unlikely that an enemy would recognize the sonar as a signal amongst oceanic noise, whereas the known structure enabled the sonar to recognize echoes of its own emissions.

Unpredictability is also the key as to why we accept physical devices as random. We know that if we had a sufficiently precise knowledge of the initial position and spin of a roulette wheel we could predict its outcome. However, the mechanism used magnifies the initial conditions to make imprecise knowledge useless for prediction. Thus we use randomness to cover our ignorance of the details of the process used, and we can do the same for nonlinear recursions.

Some Common Generators

The middle-square method and (2) were quickly rejected as sources of pseudo-random numbers, but one method of that era has survived. Lehmer (1951)

Table 2.1. Fifty Numbers from (2); read Down Columns

0.563	0.478	0.218	0.396	0.455
0.624	0.527	0.163	0.527	0.692
0.187	0.005	0.382	0.923	0.147
0.811	0.531	0.545	0.450	0.839
0.999	0.536	0.926	0.373	0.986
0.810	0.067	0.471	0.824	0.825
0.809	0.603	0.397	0.197	0.811
0.620	0.671	0.867	0.020	0.635
0.429	0.274	0.264	0.217	0.446
0.049	0.945	0.132	0.238	0.082

reported experiences with the *congruential* generator

$$U_i = aU_{i-1} \bmod 1 \tag{3}$$

which mimics the magnification effect of a roulette wheel provided the *multiplier* a is large. In practice (3) must be computed in finite-precision arithmetic, so it is usual to generate integers X_i by

$$X_i = aX_{i-1} \bmod M \tag{4}$$

and set $U_i = X_i/M$. Provided a and M are integers, (3) is then performed exactly. (Lehmer used $a = 23$, $M = 10^8 + 1$ on a *decimal* computer.) This family of generators and its cousins is now widespread.

Cryptographers are more concerned with pseudo-random sequences of *bits* and the use of special hardware. Thus they have tended to prefer pseudo-random number generators based on *shift-registers*. These record the last d bits $b_{i-1}, \ldots, b_{i-d}$, so

$$b_i = f(b_{i-1}, \ldots, b_{i-d})$$

for some function $f: \{0, 1\}^d \to \{0, 1\}$. The usual choice is

$$b_i = (a_1 b_{i-1} + \cdots + a_d b_{i-d}) \bmod 2 \tag{5}$$

for binary constants $a_1, \ldots, a_d$. There are many ways to obtain pseudo-random numbers from a sequence of pseudo-random bits. The simplest is to let

$$U_i = 0.b_{iL} b_{iL+1} \cdots b_{iL+M}$$

for integers L and M with $0 < M < L$.

These generators are discussed in detail in the next three sections. It is worth noting that minor variations may give rise to very different behavior. For example, (3) implemented in floating-point arithmetic may behave quite differently from an exact implementation via (4) (cf. exercise 2.4). All these generators have the property that eventually they reach a sequence that repeats itself; the middle square example had such a sequence of length four. This length is called the *period* of the generator. Clearly four is unacceptable, and the period should be as long as possible.

There are other generators implemented on popular microcomputers, apparently without reference to the existing literature. The BASIC inter-

preter on the Research Machines 380Z forms

$$V_i = 38965U_{i-1} + 26664$$

and then renormalizes, $U_i = 2^\alpha V_i$, where α is chosen so that $\frac{1}{2} < U_i \leqslant 1$. This reaches a cycle of period 1995 after about 10,000 calls (Research Machines Ltd., 1982).

The BASIC interpreters on both the APPLE II and CBM PET microcomputers have used

$$V_i = 0.708076143 \times U_i$$

again renormalized so $\frac{1}{2} < W_i = 2^\alpha V_i \leqslant 1$. Now W_i is a 32-bit number, so

$$W_i = 0.B_1B_2B_3B_4$$

for bytes $0 \leqslant B_i < 256$ (and $B_1 \geqslant 128$). Then on the Apple $U_i = 0.B_4B_2B_3B_1$, whereas on the Pet $U_i = 0.B_4B_3B_2B_1$ (Henery, 1983).

Testing

The sequences produced by all these generators do have some structure. Most of the rest of this chapter is devoted to identifying that structure and assessing its consequences. Any generator can be tested empirically by applying statistical tests for independence and uniformity to $(U_1, \ldots, U_N)$ for large N. However, this can be very time-consuming and always leaves open the possibility that there is some relevant structure which has not been detected.

For certain congruential and shift-register generators it is possible to find exactly the distribution of $(U_i, \ldots, U_{i+k-1})$ for small k. These theoretical tests have proved more searching than empirical tests. Thus, one is recommended to choose a generator for which theoretical tests are available and have been performed before it is put to serious use. (A mild amount of predictability might be an asset in an arcade-style game.)

The undesirable structure of (3) and (5) has led some authors to suggest applying further algorithms to their output in an attempt to destroy the structure. This might involve permuting the (U_i) or choosing between two or more generators for each i (See Section 2.5). Beware of the assumption that they improve matters. Very little progress has been made on their theoretical analysis, and the possibility remains that the known structure is transformed to something worse or that further structure is introduced. Complex algo-

rithms are by no means necessarily more "random" than simple ones, as is shown by the striking example of Knuth (1981, pp. 4–5).

The conviction of this author is that it is better to use simple and well-understood algorithms, and that within families meeting those conditions it is possible to choose pseudo-random number generators good enough for any prespecified purpose.

*Random Sequences

Philosophers have discussed several ways to define randomness, but few are relevant to our purposes. The point of view taken above is close to that of Hacking (1965), who was most interested in finite sequences in attempting to understand the foundations of statistics. Another approach particularly associated with von Mises (1919, 1957) is to define probability directly in terms of limiting frequencies of infinite sequences. A *collective* K is an infinite sequence of outcomes satisfying certain conditions. The probability of an event E is defined as the limit as $n \to \infty$ of the frequency of E in the first n terms of K. Elementary texts often introduce P(coin tossed gives heads) $= \frac{1}{2}$ in this way.

Von Mises' original conditions were too strong, but they were relaxed by others and put into definitive form by Church (1940). We say (U_i) is k-distributed if the empirical distribution of $(U_i, \ldots, U_{i+k-1})$ converges to the uniform distribution on $[0, 1]^k$. Then (U_i) should be k-distributed for all k, called ∞-distributed. Furthermore, any subsequence of (U_i) should be ∞-distributed. One has to confine attention to *computable* subsequences to avoid allowing the choice of all $U_i \geqslant \frac{1}{2}$. Computable subsequences are what are known to probabilists as optional sampling rules and insist that whether or not U_{i+1} is included is determined by knowledge of $U_1, \ldots, U_i$. There exist sequences (U_i) for which all computable subsequences are ∞-distributed; these are the von Mises–Church collectives.

All our algorithms give rise to periodic sequences and so are not even 1-distributed as only a finite set of values will occur. However, only a theory of random finite sequences seems relevant to simulation. Kolmogorov (1963) had one idea, and Chaitin (1966) and Martin-Löf (1966) defined randomness in terms of the complexity of the algorithm necessary to generate the sequence.

None of these helps with our practical problem, and we will take the pragmatic approach of making (U_i) as featureless as possible; where structure is unavoidable we will aim to make its scale small.

*Starred subsections are optional reading.

2.2. CONGRUENTIAL GENERATORS

Congruential generators are defined by

$$X_i = (aX_{i-1} + c) \bmod M \tag{1}$$

for a *multiplier* a, *shift* c, and *modulus* M, all integers. We can and will take a, c, X_i to all be in the range $\{0, 1, \ldots, M-1\}$. The pseudo-random sequence (U_i) is determined by (1) and

$$U_i = X_i/M \tag{2}$$

once the *seed* X_0 is given. We saw in Section 2.1 that generators of this form with $c = 0$ were first described by Lehmer (1951); such generators are called *multiplicative*. The early literature contains some confusion about the general (*mixed*) case; the first example published seems to be $a = 2^7, c = 1, M = 2^{35}$ by Rotenburg (1960).

The future of (X_i) is determined by its current value. Since the $M + 1$ values $(X_0, \ldots, X_M)$ cannot be distinct, at least one value must occur twice, as X_i and X_{i+k}, say. Then $X_i, \ldots, X_{i+k-1}$ is repeated as $X_{i+k}, \ldots, X_{i+2k-1}$, and so the sequence (X_i) is periodic with a period $k \leqslant M$. The *full period* M can always be achieved with $a = c = 1$. Table 2.2 illustrates the range of behavior that can occur. Clearly, the period depends on the choice of a, c and perhaps also on the seed. For multiplicative generators the *maximal* period is $M - 1$, for if 0 ever occurs it is repeated indefinitely.

It is usual to choose M to make the modulus operation efficient, and then to choose a and c to make the period as long as possible. It is known how to find the period of an arbitrary congruential generator (Fuller, 1976; Dudewicz and Ralley, 1981) but this seems unnecessary as the following theorems suffice. Proofs are given in Section 2.7.

Theorem 2.1. A congruential generator has full period M if and only if

(i) $\gcd(c, M) = 1$.
(ii) $a \equiv 1 \bmod p$ for each prime factor p of M.
(iii) $a \equiv 1 \bmod 4$ if 4 divides M.

Note that if M is a prime, full period is attained only if $a = 1$.

Theorem 2.2. A multiplicative generator with modulus $M = 2^\beta \geqslant 16$ has maximal period $M/4$, attained if and only if $a \bmod 8 = 3$ or 5. In the case $a \equiv 5 \bmod 8$, let $b = X_0 \bmod 4$. Then $(U_i - b/M)$ is the sequence output

Table 2.2. Examples of Congruential Generators

(a) $M = 16, a = 1, c = 1$
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, ...
(b) $M = 16, a = 5, c = 1$
0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3, 0, ...
(c) $M = 16, a = 5, c = 4$
0, 4, 8, 12, 0, ... or 1, 9, 1, ... or 2, 14, 10, 6, 2, ... or 3, 3, ...
or 5, 13, 5, ... or 7, 7, ... or 11, 11, ... or 15, 15, ...
(d) $M = 16, a = 5, c = 0$
1, 5, 9, 13, 1, ... or 2, 10, 2, ... or 3, 15, 11, 7, 3, ... or
4, 4, ... or 6, 14, 6, ... or 8, 8, ... or 12, 12, ...
(e) $M = 16, a = 3, c = 0$
1, 3, 9, 11, 1, ... or 2, 6, 2, ... or 4, 12, 4, ... or
5, 15, 13, 7, 5, ... or 8, 8, ... or 10, 14, 10, ...
(f) $M = 16, a = 4, c = 0$
1, 4, 0, 0, ... or 2, 8, 0, 0, ..., etc.
(g) $M = 13, a = 2, c = 0$
1, 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1, ...
(h) $M = 13, a = 4, c = 0$
1, 4, 3, 12, 9, 10, 1, ... or 2, 8, 6, 11, 5, 7, 2, ...
(i) $M = 13, a = 5, c = 0$
1, 5, 12, 8, 1, ... or 2, 10, 11, 3, 2, ... or 4, 7, 9, 6, 4, ...
(j) $M = 13, a = 12, c = 0$
1, 12, 1, ... or 2, 11, 2, ... or 3, 10, 3, ..., etc.

from the full-period generator

$$X_i = \{aX_{i-1} + b(a-1)/4\} \bmod M/4$$

The final assertion is due to Thompson (1958) and contradicts an earlier folklore that mixed generators were somehow less random than multiplicative ones. The sole advantage of a multiplicative generator seems to be to avoid $U_i = 0$.

Theorem 2.3. A multiplicative generator has period $M - 1$ only if M is prime. Then the period divides $M - 1$, and is $M - 1$ if and only if a is a *primitive root*, that is, $a \neq 0$ and $a^{(M-1)/p} \not\equiv 1 \bmod M$ for each prime factor p of $M - 1$.

Thus prime moduli are much more useful for multiplicative generators.

It may not be easy to find a primitive root, but once one is found all the others follow from:

Theorem 2.4. If a is a primitive root for a prime M, so is $a^k \bmod M$ provided $\gcd(k, M-1) = 1$.

One example of the use of this theorem is the Mersenne prime $2^{31} - 1$. We know $2^{31} - 2 = 2.3^2.7.11.31.151.331$ from which one can check that 7 is a primitive root and hence so is $7^5 = 16807$, recommended by Lewis et al. (1969) and Gustavson and Liniger (1970).

*Reduction Modulo M

The modulus M is usually chosen to make it easy to implement $Y \bmod M$ without division. If the computer works to base r and $M = r^\beta$, all we have to do is to retain the bottom β digits of Y. For example, $12345678 \bmod 10^5 = 45678$. Thus one finds powers of 2 used on binary computers and powers of 10 used on calculators.

It is only a little more difficult to implement $M = r^\beta - s$ for small s. Let $Z = Y \bmod M$ and $z = Y \bmod r^\beta$ (which is easy). Then

$$Y = z + tr^\beta = tM + (z + st)$$

so $Z = (z + ts) \bmod M$, which can be performed by subtracting M from $z + ts$ until the result is less than M. Exercise 2.7 shows s subtractions will suffice.

More care is needed to implement a multiplicative generator with $M = r^\beta + 1$. This takes the r^β values $\{1, \ldots, r\}$. If $Y = aX_{i-1}$, $Y = z + tr^\beta$ and $(aX_{i-1}) \bmod M = (z - t) \bmod M = z - t$ if $z \geqslant t$, otherwise $z - t + M$. The one remaining problem is that we will have to arrange to store the value r^β as 0 in a β-digit word.

These tricks have proved quite popular. The Mersenne prime $2^{31} - 1$ has been used with several multipliers; Lehmer originally used the prime $10^8 + 1$ and the prime $2^{16} + 1$ has been popular more recently. For $r^\beta + 1$ it is usual to let $U_i = X_i / r^\beta$, to avoid a time-consuming division.

Choosing a Generator

Thus far we have restricted our choice by choosing M so that mod M is easy, and a and c to achieve full or maximal period. There is still a lot of freedom left! We will now confine attention to full period generators and multiplicative generators with a prime modulus and maximal period. These take

values evenly spaced in [0, 1), each occurring once per cycle. Provided M is sufficiently large, the (U_i) will be nearly uniformly distributed. They may, however, be completely predictable, for example if $a = c = 1$. A less obvious example is the once very popular generator RANDU with $M = 2^{31}$, $a = 2^{16} + 3$, $c = 0$. Then

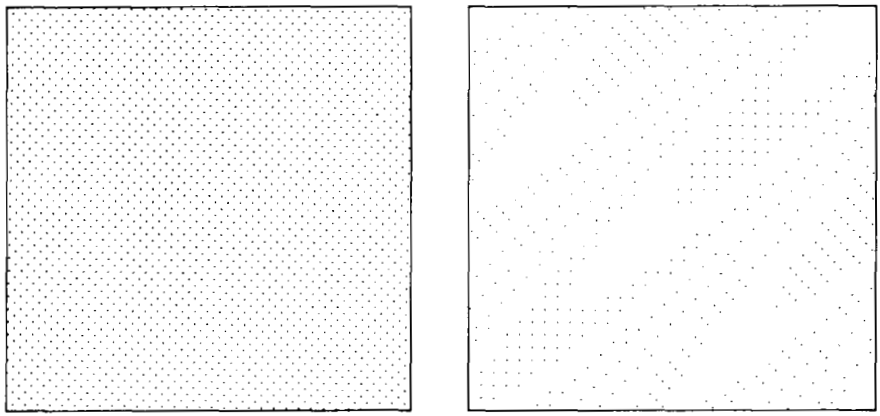
$$\begin{aligned} X_{i+2} &= (2^{16} + 3)X_{i+1} + c_1 2^{31} = (2^{16} + 3)^2 X_i + c_1 2^{31}(2^{16} + 3) + c_2 2^{31} \\ &= (6.2^{16} + 9)X_i + \{(2^{16} + 3)c_1 + c_2 + 2X_i\}2^{31} \\ &= 6(2^{16} + 3)X_i - 9X_i + c_3 2^{31} \\ &= 6X_{i+1} - 9X_i + c_4 2^{31} \end{aligned}$$

where each c_i is an integer. Thus

$$U_{i+2} - 6U_{i+1} + 9U_i \text{ is an integer}$$

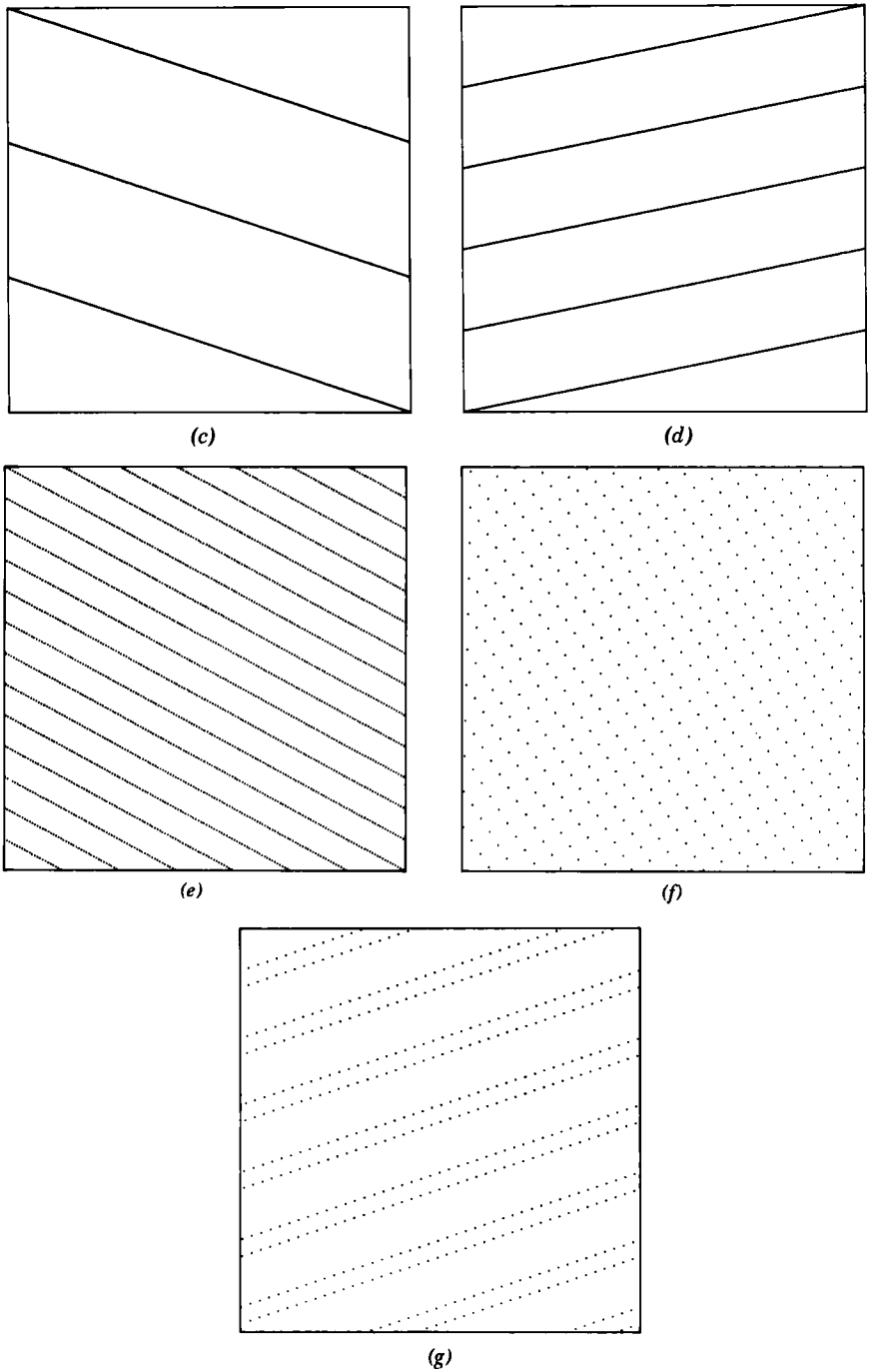
and (U_i, U_{i+1}, U_{i+2}) lies on one of 15 planes in the unit cube. This means that if (U_{i-1}, U_{i-2}) is known even to limited accuracy, then U_i is quite predictable. This is a fairly extreme example, but it has been very widely used on IBM 360/370 and PDP-11 machines!

The remedy is to choose a and c to avoid this happening. Marsaglia (1968) pointed out and Fig. 2.1 demonstrates that the k-tuples $(U_i, \ldots, U_{i+k-1})$



(a) (b)

Figure 2.1. Plots of pairs (U_i, U_{i+1}) for various congruential generators modulo 2048. (*a*) $a = 65$, $c = 1$, all 2048 points. (*b*) First 512 points of (*a*) with $X_0 = 0$. (*c*) $a = 1365$, $c = 1$. (*d*) $a = 1229$, $c = 1$. (*e*) $a = 157$, $c = 1$. (*f*) $a = 45$, $c = 0$. (*g*) $a = 43$, $c = 0$.



(c)

(d)

(e)

(f)

(g)

Figure 2.1 (*Continued*)

will always lie on a finite number of hyperplanes in $[0, 1)^k$. More precisely, we have (Beyer et al., 1971, Smith, 1971, Ripley 1983*a*).

Theorem 2.5. Let Λ_k be the lattice $\{t_1\mathbf{e}_1 + \cdots + t_k\mathbf{e}_k \mid t_i \text{ integer}\}$ with basis

$$\begin{aligned} \mathbf{e}_1 &= \frac{1}{N}(1, a, a^2, \ldots)^T \\ \mathbf{e}_j &= j\text{th unit vector for } 2 \leqslant j \leqslant k \end{aligned} \tag{3}$$

Then

(i) For full-period generators

$$\{(U_i, \ldots, U_{i+k-1})\} = [0, 1)^k \cap \{(U_0, \ldots, U_{k-1}) + \Lambda_k\}$$

with $N = M$, and $\{(U_{ki}, \ldots, U_{ki+k-1})\}$ is of the same form with $N = M/\gcd(k, M)$.

(ii) For a maximal-period multiplicative generator with prime modulus M,

$$\{(U_i, \ldots, U_{i+k-1})\} = (0, 1)^k \cap \Lambda_k$$

with $N = M$, and

$$\{(U_{ki}, \ldots, U_{ki+k-1})\} = (0, 1)^k \cap \Lambda_k$$

if $\gcd(k, M - 1) = 1$, otherwise it is a nonlattice subset of $(0, 1)^k \cap \Lambda_k$.

PROOF. See Section 2.7. □

Figure 2.1 shows what happens in some other cases. Note that we can deduce the behavior of multiplicative generators with $M = 2^\beta$, $a \equiv 5 \bmod 8$ from the full-period case by Theorem 2.2; one merely replaces M by $M/4$ in defining Λ_k.

One important conclusion of Theorem 2.5 is that the choice of c merely shifts the lattice. It has been traditional that c be chosen to minimize the correlation between U_i and U_{i+1}. However, for a fine lattice like Fig. 2.1*a* the correlation will be small for any c, whereas for Fig. 2.1*c* to minimize the correlation will merely mask the lack of independence of (U_i, U_{i+1}). There seems no compelling advantage of any other value over $c = 1$.

This reduces our choice to a few values of M and to the choice of a. In the Section 2.4 we show that a can be chosen to make the lattice of values described in Theorem 2.5 rather evenly spread in $[0, 1)^k$ and thus about $N^{-1/k}$ apart. {$[0, 1)^k \cap \Lambda_k$ contains N points.} We would like this distribution to be as even as possible, which means choosing M as large as possible. An

idea of how large M might need to be can be obtained from:

Theorem 2.6. Suppose n points are uniformly and independently distributed in $(0, 1)^k$. Let D be the smallest distance between a pair of points. Then D^k is approximately exponentially distributed with mean $2\Gamma(\frac{1}{2}k + 1)/\pi^{k/2}n^2 = \alpha_k/n^2$, say.

PROOF. Ripley (1983*a*). The approximation is asymptotic as $n \to \infty$ but remarkably accurate for n as small as 25. □

Suppose our simulations need n k-tuples. Then provided $N^{-1/k} \leqslant (\alpha_k/99.50n^2)^{1/k}$, the 1% point of the distribution of D, the nonuniformity of the pseudo-random numbers will be negligible. This reduces to $N \geqslant 200n^2$, say 2^{27} for $n = 1000$. Thus:

Recommendation. A congruential generator should have period as large as possible, at least 2^{30}, a multiplier a chosen to give period M or $M - 1$, *and* a good lattice structure as described in Section 2.4.

2.3. SHIFT-REGISTER GENERATORS

Shift registers were introduced in Section 2.1. In their most general form they have $M \geqslant 2$ states, but we will confine attention to the binary case which has been the only one used for pseudo-random numbers. We have

$$b_i = (a_1 b_{i-1} + \cdots + a_d b_{i-d}) \bmod 2 \tag{1}$$

This is easy to implement in a hardware circuit by use of a shift register, hence the name. Note that addition modulo 2 and exclusive or have the same truth table, and so we may replace (1) by

$$b_i = b_{i-j_1} \text{ EOR } b_{i-j_2} \cdots b_{i-j_k}$$

where $a_{j_1} = \cdots = a_{j_k} = 1$ and all other $a_j = 0$.

Each b_i is determined by $(b_{i-1}, \ldots, b_{i-d})$, which has at most 2^d possible values. Furthermore, if this is the zero vector, then $b_i = 0$, and $b_j = 0$ for all $j \geqslant i$. Thus, the maximal period is $2^d - 1$. The details of how to find the period of (1) (or even if the maximal period is attained) depend on methods of factorizing polynomials over finite fields. Golomb (1967) summarizes the algebra needed. Recursion (1) is associated with the polynomial

$$f(x) = x^d + a_1 x^{d-1} + \cdots + a_d$$

It has been usual to consider trinomials $1 + x^q + x^p$ with $1 \leqslant q < p$, so

$$b_i = b_{i-p} \text{ EOR } b_{i-(p-q)} \tag{2}$$

Reversing the sequence shows $1 + x^{p-q} + x^p$ and

$$b_i = b_{i-p} \text{ EOR } b_{i-q}$$

must have the same period. Table 2.3 lists some pairs (p, q) that give maximal period $2^p - 1$. [From Golomb (1967). Further values are given by Lewis and Payne (1973, Fig. 9), Zierler and Brillhart (1968, 1969), and Zierler (1969).] Some specific suggestions are $p = 98$, $q = 27$ (Lewis and Payne, 1973); $p = 521$, $q = 32$ (Bright and Enison, 1979); and $p = 607$, $q = 273$ (Tootill et al., 1973).

Tausworthe (1965) suggested using

$$U_i = \sum_{1}^{L} 2^{-s} b_{it+s} = 0.b_{it+1} \cdots b_{it+L}$$

that is, L-bit binary fractions taken t apart. Consequently, such random-number generators are called Tausworthe generators. The parameter t is called the *decimation*. A decimation is said to be *proper* if $\gcd(t, 2^p - 1) = 1$. For a proper decimation (U_i) has period $2^p - 1$ (since this is the period of each of its bits by Lemma C of Section 2.7).

The BBC microcomputer has a Tausworthe generator with $p = 33$, $q = 13$, $t = L = 32$. This is a proper decimation, and so has period $2^{33} - 1$. (The order of the *bytes* in U_i is reversed, but this has no consequence.) The following algorithm is a neat way to implement a Tausworthe generator with

Table 2.3. All Values of (p, q) for which $1 + x^q + x^p$ gives a Maximum-Period Shift Register, with $p \leqslant 36$

p	q	p	q	p	q
2	1	11	2, 9	25	3, 7, 18, 22
3	1, 2	15	1, 4, 7, 8, 11, 14	28	3, 9, 13, 15, 19, 25
4	1, 3	17	3, 5, 6, 11, 12, 14	29	2, 27
5	2, 3	18	7, 11	31	3, 6, 7, 13, 18,
6	1, 5	20	3, 17		24, 25, 28
7	1, 3, 4, 6	21	2, 19	33	13, 20
9	4, 5	22	1, 21	35	2, 33
10	3, 7	23	5, 9, 14, 18	36	11, 25

$q \leqslant p/2$, $p = t = L$ = word length. [It is used with $p = 36$, $q = 11$ on the Honeywell Multics system (Sibson, 1984).]

Algorithm 2.1 (Whittlesey, 1968; Payne 1970). Assume U_i is stored in a word X with b_{it+1} on the left:

1. Copy X to T.
2. Left shift X by q bits, filling with zeroes.
3. Let $X = X$ EOR T, copy X to T (bitwise exclusive or).
4. Right shift T by $p - q$ bits, filling with zeroes.
5. $X = X$ EOR T now contains U_{i+1}.

One can use $p = t = L <$ word length by padding with zeroes on the right. Exercise 2.9 shows that this algorithm works.

Lewis and Payne suggested making up an L-bit integer from nonconsecutive terms in (b_i), for example,

$$Y_i = b_i b_{i-l_2} \cdots b_{i-l_L} \tag{3}$$

for delays $l_2, \ldots, l_L$. Each bit of Y_i still obeys (2), so we can form

$$Y_i = Y_{i-p} \text{ EOR } Y_{i-(p-q)} \tag{4}$$

which can be implemented by a simple circular buffer. Such generators are called *generalized feedback shift registers* (GFSRs). They were introduced to be faster than Tausworthe generators, but Algorithm 2.1 may be faster for $p = t = L$. The p starting values for recursion (4) need not satisfy (3). However, the period of (Y_i) will depend on the starting values. Obviously we will obtain random numbers by $U_i = 2^{-L} Y_i$, so $0 < U_i < 1$.

Example. $p = 5, q = 2$ gives the bit sequence

$$1111100011011101010000100101100\ldots$$

of maximum period 31. The Tausworthe sequence with $t = L = 5$ is

$$31, 3, 14, 20, 4, 22, 15, 17, 23, 10, 2, 11, 7, 24, 27, 21, 1, 5, 19,$$
$$28, 13, 26, 16, 18, 25, 30, 6, 29, 8, 9, 12, \ldots$$

If we take the GFSR $b_i b_{i-6} b_{i-12} b_{i-18} b_{i-24}$ we obtain

$$1, 13, 8, 29, 30, 9, 16, 22, 20, 14, 31, 4, 24, 11, 10, 7, 15, 18,$$
$$12, 5, 21, 3, 23, 25, 6, 2, 26, 17, 27, 28, 19, \ldots$$ □

The theoretical analysis of Tausworthe and GFSR sequences concentrates on the k-tuples of integers $(Y_i, \ldots, Y_{i+k-1})$. We would like all k-tuples to be equally frequent in a period. There is a minor problem with the missing p-fold zero in (b_i). We say (Y_i) is *k-distributed* if all k-tuples are equally frequent except zero, which occurs one less time in each period.

Theorem 2.7. A Tausworthe generator with proper decimation is k-distributed for $1 \leqslant k \leqslant \text{int}[p/t]$.

PROOF. $(Y_i, \ldots, Y_{i+k-1})$ is made up from kL bits of $(b_{it+1}, \ldots, b_{it+kt})$. If $kt \leqslant p$, this is a subset of $(b_{it+1}, \ldots, b_{it+p})$ that takes all possible values except all zeroes once in a period. Thus every nonzero k-tuple of Y_i's is equally frequent in a period. □

Suppose $kL > p$. Then only $2^p - 1$ of the possible $2^{kL} - 1$ values of $(Y_i, \ldots, Y_{i+k-1})$ can occur. Consequently, k-distribution is impossible for $k > \text{int}[p/L]$. Figure 2.2 shows that there may be advantages in taking $t > L$ to improve the k-dimensional structure, and that when k-distribution fails, it can fail dramatically.

The analogue of Theorem 2.7 is not automatic for GFSRs; it depends on the starting values. Let A be the $p \times L$ matrix whose rows are the bits of $Y_1, \ldots, Y_p$, called the *seed matrix*.

Theorem 2.8. A GFSR sequence is 1-distributed if and only if its seed matrix is nonsingular.

PROOF. Let A_i be the corresponding matrix for $(Y_i, \ldots, Y_{i+p-1})$. Define a $p \times p$ matrix C by

$$C_{ij} = \delta_{i,j-1} \text{ for } 1 \leqslant i < p, 1 \leqslant j \leqslant p$$

$$C_{pj} = \delta_{ij} + \delta_{qj} \text{ for } 1 \leqslant j \leqslant p$$

so $A_i = CA_{i-1} = C^{i-1}A$ with addition modulo 2.

Now $(b_{i+j}, \ldots, b_{i+j+p-1})^T = C^j(b_i, \ldots, b_{i+p-1})^T$ so $C^0, \ldots, C^s$, $s = 2^p - 2$, are distinct matrices; hence Y_i is 1-distributed if and only if A is nonsingular. □

Theorem 2.9. A GFSR sequence is k-distributed if and only if both $k \leqslant \text{int}[p/L]$ and the matrix with row i, the bits of $(Y_i, \ldots, Y_{i+k-1})$, $i = 1, \ldots, p$ is nonsingular.

PROOF. Apply theorem 2.8 to the kL-bit integers made up by concatenating $Y_i, \ldots, Y_{i+k-1}$. □

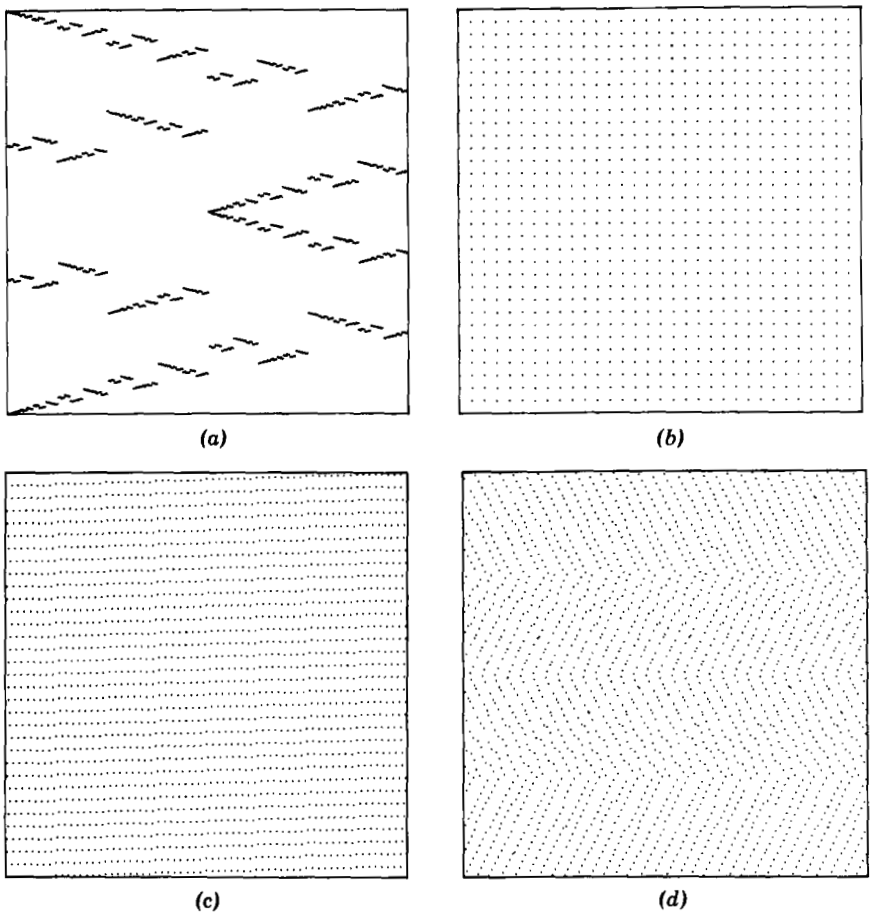


Figure 2.2. Plots of all pairs (U_i, U_{i+1}) from shift-register generators with $p = 11$, $q = 2$. (*a*) $t = L = 11$. (*b*) $t = L = 5$, so 2-distributed. (*c*) $t = 5$, $L = 8$. (*d*) $t = 17$, $L = 8$.

Example. Consider $p = 7$, $q = 1$ with period 127. A GFSR sequence with $L = 3$ gives

0 1 2 3 4 5 6 1 3 1 7 1 3 7 2 2 6 6 2 4
5 0 4 0 4 6 1 5 4 4 4 2 7 4 1 0 0 6 5 3
5 1 0 6 3 6 6 4 1 6 5 5 0 2 5 7 3 0 5 2
7 2 4 3 5 7 5 5 6 7 6 2 2 0 3 1 1 4 0 2
3 2 0 5 4 2 1 1 2 5 1 6 3 0 3 7 4 7 5 3
3 4 3 3 2 6 0 7 7 0 1 4 6 7 0 7 1 5 2 1
7 7 6 4 7 3 6 . . .

which is 2-distributed. The seed matrix is

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

which is clearly nonsingular. For 2-distribution we consider

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

which is nonobviously of rank six. (See below.) If we were to start with 6, 5, 4, 3, 2, 1, 0 we would obtain a nonsingular seed matrix and so 1-distribution. However, for 2-distribution we have

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{5}$$

which is singular since the sum (mod 2) of columns 2, 5, and 6 is zero. Thus this sequence is not 2-distributed. (See Exercise 2.11.) □

Theorems 2.7 and 2.9 show the need for very long periods. For example, Bright and Enison (1970) and Fushimi and Tezuka (1983) both consider $p = 521$, $q = 32$, with $L = 64$ and $L = 32$, respectively. By Theorem 2.9 these are candidates to be 8-distributed and 16-distributed, and Fushimi and Tezuka check that this is so. However, even with a period of $2^{521} - 1$ the

k-tuples have a spacing of 2^{-32} in dimensions 1–16, whereas congruential generators of similar periods will do much better for $k \ll \text{int}[p/L]$.

In general a very long period will be easier to achieve with a GFSR generator than a Tausworthe or congruential generator. All one has to do is to increase the size of the buffer retaining $Y_{i-1}, \ldots, Y_{i-p}$ when increasing p. One then uses the following algorithm.

Algorithm 2.2. Locations $Y[1] \cdots Y[p]$ are set aside as a buffer, initialized with $Y_{-1}, \ldots, Y_{-p}$, and pointers I and J set to $p - q$ and p

1. $Y = Y[I]$ EOR $Y[J]$, $Y[J] = Y$.
2. $I = I - 1$; if $I = 0$ then $I = p$.
3. $J = J - 1$; if $J = 0$ then $J = p$.
4. Return Y.

Normally each Y will be held within a computer word; if this is not possible, operation 1 is applied to each part of Y independently.

There remains the problem of choosing the starting values to achieve maximal k-distribution. Trial-and-error checking the conditions of Theorem 2.9 seems to be the only general way known. To achieve 1-distribution is easy; merely including $1, 2, \ldots, 2^{L-1}$ in $(Y_1, \ldots, Y_p)$ ensures that the seed matrix is nonsingular. Nonsingularity of a $p \times kL$ binary matrix is easily checked by reducing it to upper triangular form by exclusive-oring rows. For example, consider (5). The following process consists of exclusive-oring each row in turn with lower rows to remove 1's from the next column, or permuting rows. One rapidly finds the matrix to be of rank 5 and so singular.

$$\begin{bmatrix} 1&1&0&1&0&1 \\ 1&0&1&1&0&0 \\ 1&0&0&0&1&1 \\ 0&1&1&0&1&0 \\ 0&1&0&0&0&1 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&1 \end{bmatrix} \rightarrow \begin{bmatrix} 1&1&0&1&0&1 \\ 0&1&1&0&0&1 \\ 0&1&0&1&1&0 \\ 0&1&1&0&1&0 \\ 0&1&0&0&0&1 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&1 \end{bmatrix} \rightarrow \begin{bmatrix} 1&1&0&1&0&1 \\ 0&1&1&0&0&1 \\ 0&0&1&1&1&1 \\ 0&0&0&0&1&1 \\ 0&0&1&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&1 \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 1&1&0&1&0&1 \\ 0&1&1&0&0&1 \\ 0&0&1&1&1&1 \\ 0&0&0&0&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&0&1&1 \end{bmatrix} \rightarrow \begin{bmatrix} 1&1&0&1&0&1 \\ 0&1&1&0&0&1 \\ 0&0&1&1&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&0&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&0&1&1 \end{bmatrix} \rightarrow \begin{bmatrix} 1&1&0&1&0&1 \\ 0&1&1&0&0&1 \\ 0&0&1&1&1&1 \\ 0&0&0&1&1&1 \\ 0&0&0&0&1&1 \\ 0&0&0&0&0&0 \\ 0&0&0&0&1&1 \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Another method is to select the initial values of the form (3). Then the bits of $(Y_i, \ldots, Y_{i+k-1})$ are $\{b_{i-j+k-1} | j = t, l_2 + t, \ldots, l_L + t; t = 0, \ldots, k-1\}$. Provided this set of values is distinct, the proof of Theorem 2.7 shows that these k-tuples are 1-distributed and hence that (Y_i) is k-distributed. One must then choose the delays at least k apart and with $l_L \leqslant p - k$, which is always possible for $kL \leqslant p$, for example by

$$Y_i = b_i b_{i-k} \ldots b_{i-(L-1)k}$$

Specific implementations of generators of this type are considered by Arvillias and Maritsas (1978) and Fushimi and Tezuka (1983).

Fellen (1969) and Toothill et al. (1971, 1973) study less relevant properties of Tausworthe generators.

2.4. LATTICE STRUCTURE

We saw in Section 2.2 that the k-tuples $(U_i, \ldots, U_{i+k-1})$ from certain congruential generators lie on lattices in the unit hypercube. Both congruential and shift-register generators suffer from the same problem: for a period of length N there are only N k-tuples. For shift-register generators with k-distribution the word length L is restricted, so that these N points lie on the cubic lattice of side 2^{-L}, with $(2^{-L})^k \leqslant N + 1$. Figure 2.2 shows what happens if we increase the word length.

Congruential generators can achieve a very similar k-dimensional behaviour, *provided* that the multiplier is chosen suitably. The rest of this section is devoted to a detailed study of the k-dimensional output of full-period congruential generators, and those of maximal period with a prime modulus.

Lattices

A lattice Λ in $\mathbb{R}^k$ is defined by k linearly independent vectors $\mathbf{e}_1, \ldots, \mathbf{e}_k$. Then

$$\Lambda = \{t_1\mathbf{e}_1 + \cdots + t_k\mathbf{e}_k \mid t_i \text{ integer}\}$$

is the set of sums of integer multiples of the $\mathbf{e}_i$. The set $\{\mathbf{e}_i\}$ is called a *basis* for Λ.

Various measures of the "granularity" of a lattice have been developed. For a cubic lattice we use the smallest spacing between a pair of points, l_1, which is also the length of the smallest nonzero vector in Λ. Most people envisage a lattice as being made up by repeating a basic parallelogram cell. (Look at Fig. 2.1 again to convince yourself.) One way to define such a cell is to take $\mathbf{e}_1$ as a shortest nonzero vector in Λ, $\mathbf{e}_2$ as the shortest vector linearly independent of $\mathbf{e}_2$, $\mathbf{e}_3$ as the shortest linearly independent of $\mathbf{e}_1$ and $\mathbf{e}_2$, and so on. For $k \leqslant 4$ this generates a basis for Λ (except for one exceptional lattice for $k = 4$, for which only some of the choices for shortest work). However, one's intuition about lattices fails for $k \geqslant 5$. Let l_i be the length of $\mathbf{e}_i$ chosen in this way. Then l_k is one measure of "granularity," and $r = l_k/l_1$ measures the "uniformity" of the lattice. (We can usually achieve $r \leqslant 2$.)

Yet another method of measuring uniformity was given in Section 2.2 where we saw that the triples from RANDU lie on only 15 planes. It has proved more useful to measure the maximal spacing between parallel planes that cover the lattice. Clearly Fig. 2.1*a* will have a smaller spacing than Fig. 2.1*c*. Call this spacing s_k and its reciprocal v_k.

Computing Lattice Constants

The theory behind the following methods is described later in this section. The case $k = 2$ is easiest.

Theorem 2.10. Start with any basis $(\mathbf{e}, \mathbf{f})$ for Λ_2. Relabel if necessary so that $\|\mathbf{e}\| \leqslant \|\mathbf{f}\|$. Compute $s = \text{nint}(\mathbf{e}^T\mathbf{f}/\|\mathbf{e}\|^2)$. If $s \neq 0$, replace $\mathbf{f}$ by $\mathbf{f} - s\mathbf{e}$ and repeat. If $s = 0$, then $l_1 = \|\mathbf{e}\|$, $l_2 = \|\mathbf{f}\|$, and $v_2 = Nl_1$.

Remarks. (i) $\text{nint}(x)$ is the nearest integer to x, halves being rounded toward zero, so $\text{nint}(-3.5) = -3$, for example. (ii) The coordinates of all vectors in Λ_2 are multiples of $1/N$ (since this is true of the basis 2.3). Thus it may be convenient to perform the calculations on $N\mathbf{e}$ and $N\mathbf{f}$.

PROOF. (i) $\|\mathbf{f} - s\mathbf{e}\|^2 = \|\mathbf{f}\|^2 + s^2\|\mathbf{e}\|^2 - 2s\mathbf{e}^T\mathbf{f} < \|\mathbf{f}\|^2$ if and only if $s \neq 0$ and $2\mathbf{e}^T\mathbf{f} > s\|\mathbf{e}\|^2$, if and only if $s \neq 0$. Thus the algorithm strictly reduces the length of $\mathbf{f}$ and by remark (ii) must terminate.

(ii) Clearly $\mathbf{f} - s\mathbf{e} \in \Lambda_2$, and $(\mathbf{e}, \mathbf{f} - s\mathbf{e})$ is another basis.

(iii) Now suppose we have a basis with $s = 0$. By replacing $\mathbf{f}$ by $-\mathbf{f}$ if necessary, we may assume $0 \leqslant \mathbf{e}^T\mathbf{f} \leqslant \frac{1}{2}\|\mathbf{e}\|^2$. Suppose $\mathbf{g} \in \Lambda_2$ and $\|\mathbf{g}\| < \|\mathbf{f}\|$. Then there are integers u and v with $\mathbf{g} = u\mathbf{e} + v\mathbf{f}$, and by changing the sign we may assume $u > 0$. Then if $\alpha = v/u$,

$$\|\mathbf{g}\|^2 = u^2(\|\mathbf{e}\|^2 + \alpha^2\|\mathbf{f}\|^2 + 2\alpha\mathbf{e}^T\mathbf{f})$$

Thus if $v > 0$ we can find a shorter $\mathbf{g}$ by replacing v by $-v$. Then

$$\begin{aligned}\|\mathbf{f}\|^2 > \|\mathbf{g}\|^2 &\geqslant \|\mathbf{e}\|^2 + \alpha^2\|\mathbf{f}\|^2 + 2\alpha\mathbf{e}^T\mathbf{f} \\ &\geqslant (1+\alpha)\|\mathbf{e}\|^2 + \alpha^2\|\mathbf{f}\|^2\end{aligned}$$

This implies $-1 < \alpha \leqslant 0$, or $0 \leqslant -v < u$. Suppose $v \neq 0$. Let $\beta = -u/v > 1$. Then

$$\|\mathbf{f}\|^2 > \|\mathbf{g}\|^2 \geqslant \|\mathbf{f} - \beta\mathbf{e}\|^2 \geqslant \beta(\beta - 1)\|\mathbf{e}\|^2 + \|\mathbf{f}\|^2$$

a contradiction. Thus the only vectors shorter than $\mathbf{f}$ in Λ_2 are integer multiples of $\mathbf{e}$, so $l_1 = \|\mathbf{e}\|$, $l_2 = \|\mathbf{f}\|$.

(iv) Choose H as the line through the origin of a family of parallel lines with spacing s_2. Choose $\mathbf{f}$ as a shortest vector in $H \cap \Lambda_2$. Then the basic parallelogram has base $\mathbf{f}$ and height s_2, so area $= s_2\|\mathbf{f}\| = 1/N$. Thus, $s_2 = 1/N\|\mathbf{f}\| \geqslant 1/Nl_1$, and equality is attained if H contains a vector attaining l_1. Thus $v_2 = 1/s_2 = Nl_1$. □

This method of changing basis was proposed by Beyer et al. (1971) and Marsaglia (1972) but has a long history in number theory. It normally works extremely rapidly.

Example. For Fig. 2.1*f* we have $N = 512$, $\mathbf{e}_1 = (1, 45)/512$, and $\mathbf{e}_2 = (0, 1)$. For ease of working we multiply both vectors by 512.

(i) $\mathbf{e} = (1, 45)$, $\mathbf{f} = (0, 512)$ gives $s = 11$, $\mathbf{f} \to (-11, 17)$.
(ii) $\mathbf{e} = (-11, 17)$, $\mathbf{f} = (1, 45)$ gives $s = 2$, $\mathbf{f} \to (23, 11)$.
(iii) $\mathbf{e} = (-11, 17)$, $\mathbf{f} = (23, 11)$ so $s = 0$

Hence, $l_1 = 0.0395$, $l_2 = 0.0498$, $v_2 = 20.25$, $r = 1.26$. □

In three or more dimensions we can give bounds on the lattice constants from Theorem 2.11. The vectors $\mathbf{e}_j^*$ defined there are known as the *dual basis*. They are the rows of E^{-1}, where E has columns $\mathbf{e}_1 \cdots \mathbf{e}_k$.

Theorem 2.11. Let $(\mathbf{e}_i)$ be any basis of Λ_k with increasing $\|\mathbf{e}_i\|$. Let $\mathbf{e}_1^*, \ldots, \mathbf{e}_k^*$ be defined by $\mathbf{e}_i^T\mathbf{e}_j^* = \delta_{ij}$, and $w = \min \|\mathbf{e}_k^*\|$. Then

(i) $1/w, \dfrac{1}{N}\prod_1^{k-1}\|\mathbf{e}_i\| \leqslant l_k \leqslant \|\mathbf{e}_k\|$

(ii) $1/\|\mathbf{e}_k\| \leqslant v_k \leqslant w$

PROOF. From the definitions and Theorem 2.16. □

To use Theorem 2.11 we need to find a basis made up of short vectors. We can extend the ideas of Theorem 2.10 by:

Algorithm 2.3. Fix some order of the pairs $\{i, j\}$. Apply the following until no change is made for any pair:

Assume $\|\mathbf{e}_i\| \leqslant \|\mathbf{e}_j\|$. Let $s = \text{nint}(\mathbf{e}_i^T \mathbf{e}_j / \|\mathbf{e}_i\|^2)$. If $s \neq 0$, replace $\mathbf{e}_j$ by $\mathbf{e}_j - s\mathbf{e}_i$.

Exactly as for $k = 2$ this will terminate in a finite number of steps and find a basis with shorter vectors. In most cases the right-hand inequalities in Theorem 2.11 are then equalities, but not always. The bounds are usually quite close.

Example. $k = 3$, $N = 2^{16}$, $a = 249$. Using $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$ as the order reduces the basis (2.3) to

$$N\mathbf{e}_1 = (260, -796, -1596)$$

$$N\mathbf{e}_2 = (-519, 1841, -343)$$

$$N\mathbf{e}_3 = (1316, 4, 996)$$

and $\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3$ is shorter than $\mathbf{e}_2$, the longest of these vectors, but even so Theorem 2.11 yields $0.0259 \leqslant l_3 \leqslant 0.0296$. □

This suggests trying further transformations, as does

Theorem 2.12. (Minkowski). For $k = 3$ or 4, basis $\mathbf{e}_i$ has $\|\mathbf{e}_i\| = l_i$, $i = 1, \ldots, k$ provided

(i) $\|\mathbf{e}_1\| \leqslant \cdots \leqslant \|\mathbf{e}_k\|$ and

(ii) $\|\mathbf{e}_i\| \leqslant \|\mathbf{e}_i + \sum_{j<i} c_j \mathbf{e}_j\|$, each $c_j \in \{0, +1, -1\}$.

PROOF. Section 2.7. □

In applying Algorithm 2.3 we have already checked all combinations with just one $c_i \neq 0$. ($s = 0$ implies $\|\mathbf{e}_i \pm \mathbf{e}_j\| \geqslant \|\mathbf{e}_i\|$.) For $k = 3$ this leaves four combinations for $\mathbf{e}_3$, and for $k = 4$ four for $\mathbf{e}_3$ and 20 for $\mathbf{e}_4$. If we do find a shorter vector, we can replace $\mathbf{e}_i$ by that vector and repeat Algorithm 2.3 and the test of Theorem 2.12. In our example this gives $l_3 = 0.0275$.

This gives us a way to find l_1, l_k, and r exactly for $k \leqslant 4$. For $k \geqslant 5$ it is possible that no basis attains $l_1, \ldots, l_k$, but the bounds of Theorem 2.11 almost always suffice after applying Algorithm 2.3.

The Spectral Test

The vectors $(\mathbf{e}_i^*)$ introduced in Theorem 2.11 form a basis for another lattice Λ^*, known as the *polar* (or dual) lattice.

Theorem 2.13. Let $\mathscr{H} = \{\mathbf{x} | \mathbf{x}^T\mathbf{u} \text{ integer}\}$ be a family of parallel hyperplanes covering Λ. Then $\mathbf{u} \in \Lambda^*$ and $v_k = l_1^*$, the length of the shortest nonzero vector in Λ^*.

PROOF. The basis $(\mathbf{e}_i^*)$ spans $\mathbb{R}^k$, so $\mathbf{u} = t_1\mathbf{e}_1^* + \cdots + t_k\mathbf{e}_k^*$ for *real* t_i. For each i, $\mathbf{e}_i \in \mathscr{H}$, so $t_i = \mathbf{e}_i^T\mathbf{u}$ is an integer; hence $\mathbf{u} \in \Lambda^*$. Conversely, if $\mathbf{u} \in \Lambda^*$, $\mathbf{u} \neq 0$, then all $\mathbf{e}_i \in \mathscr{H}$ and so $\Lambda \subset \mathscr{H}$.

The spacing of $\mathscr{H}$ is $\min\{\|\mathbf{x}\| \mid \mathbf{x}^T\mathbf{u} = 1/\|\mathbf{u}\|$, so $v_k = \min\{\|\mathbf{u}\| \mid \mathscr{H} \supset \Lambda\} = \min\{\|\mathbf{u}\| \mid 0 \neq \mathbf{u} \in \Lambda^*\} = l_1^*$. □

We can rewrite this conclusion by noting that $\mathbf{u} \in \Lambda^*$ if and only if $\mathbf{e}_1^T\mathbf{u} \cdots \mathbf{e}_k^T\mathbf{u}$ are integers. Thus

$$v_k = \min\{\|\mathbf{u}\| \mid 0 \neq \mathbf{u},\ u_i \text{ integers},\ u_1 + au_2 + \cdots + a^{k-1}u_k \text{ is a multiple of } N\}$$

Such a quantity was defined by Coveyou and Macpherson (1967), who called testing for large values of v_k the "spectral test." In that context "large" is often assessed by forming $\mu_k = \omega_k v_k^k/N$, where $\omega_k = \pi^{k/2}/\Gamma(k/2 + 1)$ is the volume of the unit ball in $\mathbb{R}^k$. Values of μ_k larger than 1 are thought good. We can use Theorem 2.16 on Λ^* to show that $v_k \leqslant c_k N^{1/k}$, so $\mu_k \leqslant \omega_k c_k^k$, which helps explain why values greater than 1 are thought good. It seems preferable to use the inequality for v_k in a similar way to $r \geqslant 1$, remembering that v_k is an absolute measure of the granularity of Λ_k.

It remains to find v_k. For v_2 we have Theorem 2.10. For $k = 3$ or 4 we could apply Algorithm 2.3 and Theorem 2.12 to the polar lattice Λ^*. However, if the bounds of Theorem 2.11 are not sufficient, we can carry out a finite search by

Theorem 2.14 (Dieter, 1975). $v_k = \min\{\|\mathbf{u}\| \mid \mathbf{u} = t_1\mathbf{e}_1^* + \cdots + t_k\mathbf{e}_k^* \neq 0,$ $|t_i| \leqslant \text{int}[w\|\mathbf{e}_i\|]\}$ for any polar basis $(\mathbf{e}_i^*)$.

PROOF. $|t_i| = |\mathbf{e}_i^T\mathbf{u}| \leqslant \|\mathbf{e}_i\|\ \|\mathbf{u}\|$ by Cauchy–Schwartz. For the minimal $\mathbf{u}$, $\|\mathbf{u}\| = v_k \leqslant w$ by Theorem 2.11. □

In practice $\text{int}[w\|\mathbf{e}_i\|]$ is almost always 0 or 1, and by Theorem 2.12 we can take it to be 1 for $k \leqslant 4$. One further useful trick [from Knuth (1981)] is to take as the starting basis for Λ_k the basis (2.3) transformed by the transformations used for Λ_{k-1}, and to update the polar basis $(\mathbf{e}_i^*)$ with $(\mathbf{e}_i)$. Fortran code is given by Hopkins (1983) and in Appendix B.

Example. $N = 512, a = 45$ (continued). The final basis for Λ_2 was

$$N\mathbf{e}_1 = (-11, 17), \qquad N\mathbf{e}_2 = (23, 11)$$

This gives us as initial basis

$$N\mathbf{e}_1 = (-11, 17, 253)$$
$$N\mathbf{e}_2 = (23, 11, -529)$$
$$N\mathbf{e}_3 = (0, 0, 512)$$

since $a^2 \equiv -23 \bmod N$, and the last element must be a^2 times the first. From this we have

$$\mathbf{e}_1^* = (-11, 23, 0)$$
$$\mathbf{e}_2^* = (17, 11, 0)$$
$$\mathbf{e}_3^* = (23, 0, 1) \qquad (23 \equiv -a^2)$$

We now apply Algorithm 2.3, noting that if $\mathbf{e}_j \to \mathbf{e}_j - s\mathbf{e}_i$, then $\mathbf{e}_i^* \to \mathbf{e}_i^* + s\mathbf{e}_j^*$, to yield

$$N\mathbf{e}_1 = (23, 11, -17) \qquad \mathbf{e}_1^* = (12, 6, -10)$$
$$N\mathbf{e}_2 = (22, -34, 6) \qquad \mathbf{e}_2^* = (7, -10, 3)$$
$$N\mathbf{e}_3 = (82, 106, 162) \qquad \mathbf{e}_3^* = (1, 1, 2)$$

when reordered in increasing length. This passes Theorem 2.12, so $l_1 = 0.0598$, $l_3 = 0.411$, $r = 6.86$, and $w = \sqrt{6}$. We have $|t_1| \leqslant 0$, $|t_2| \leqslant 0$, $|t_3| \leqslant 1$ in Theorem 2.14, so $v_3 = w$ attained at $\mathbf{e}_3^*$. Note that $v_3 l_3 = 1.007$. □

Assessing Congruential Generators

Table 2.4 lists some of the results of applying the preceding methods to Λ_2, Λ_3, Λ_4 for some commonly used generators. Only r and v_k are shown, since in all cases l_k is very close to $1/v_k$.

Line 1 is the generator G05CAF of the NAG Fortran library. Line 2 from Marsaglia (1972) is used by DEC for its VAX compilers. Line 7 is from CDC Fortran (FTN 4.x and 5.x compilers). All seem quite acceptable. Line 3 is used by BASIC on the Sinclair ZX81 (Tootill, 1982). Figure 2.3 confirms its two-dimensional granularity, which is due to both a bad choice of multiplier (Exercise 2.13) and too short a period. Lines 4 and 5 are for IBM 360/370

Table 2.4. Lattice Criteria for Certain Congruential Generators

				$k = 2$		$k = 3$		$k = 4$	
	M	a	c	r	v	r	v	r	v
1	2^{59}	13^{13}	0	1.23	3.44×10^{8}	1.57	4.29×10^{5}	1.93	1.55×10^{4}
2	2^{32}	69069	1	1.06	6.51×10^{4}	1.29	1440	1.30	230
3	$2^{16} + 1$	75	0	11.7	75	1.59	31.4	3.43	9.17
4	$2^{31} - 1$	7^{5}	0	7.60	1.68×10^{4}	3.39	639	2.07	147
5	$2^{31} - 1$	630360016	0	1.29	4.09×10^{4}	2.92	625	1.64	201
6	2^{35}	8404997	1	2.81	1.11×10^{5}	1.93	2930	5.98	147
7	2^{48}	44, 485, 709, 377, 909	0	1.29	7.45×10^{6}	1.85	3.44×10^{4}	3.85	1370
8	2^{32}	2147001325	715136305	1.13	6.40×10^{4}	1.09	1540	1.16	269
9	$10^{8} + 1$	23	0	1.89×10^{5}	23	8211	23	357	23
10	10^{9}	314159221	211324863	3.89	1.61×10^{4}	2.12	800	2.46	103
11	2^{48}	5^{17}	1	1.87	1.23×10^{7}	2.86	4.74×10^{4}	1.67	3400
12	$2^{31} - 1$	397204094	0	2.82	2.77×10^{4}	2.63	832	1.50	171

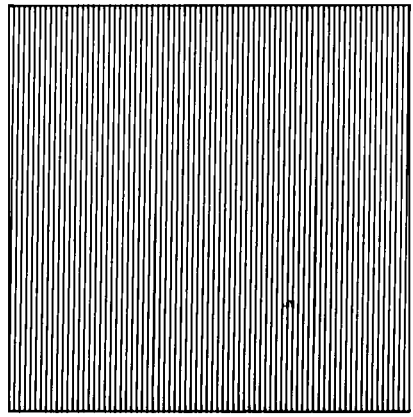


Figure 2.3. Plot of all pairs (U_i, U_{i+1}) from the Sinclair ZX81 generator, $X_i = 75X_{i-1} \mod(2^{16} + 1)$.

series machines from Lewis et al. (1969) and Payne et al. (1969). Line 6 is the default generator of GLIM3 (Baker and Nelder, 1978), a widely used statistical package, and line 8 is that built into BCPL (Richards and Whitby-Strevens 1979), a progenitor of C. Line 9 is Lehmer's original suggestion and lines 10 and 11 are from van Es et al. (1983). Lines 4 and 12 are routines GGUBFS and GGUBT of IMSL. Many other generators have been tested in this way, and the calculations can even be done on a microcomputer (Ripley, 1983b).

*Theory

Let E be a $k \times k$ matrix whose columns form a basis for a lattice Λ. Let $d(\Lambda)$ be the modulus of the determinant of E. Then $d(\Lambda)$ is the k-dimensional volume of the basic lattice "cell."

Theorem 2.15.

$$d(\Lambda) \leqslant \prod_{1}^{k} l_i \leqslant (c_k)^k d(\Lambda)$$

where

$k =$	2	3	4	5	6	7	6
$(c_k)^{2k} =$	$\frac{4}{3}$	2	4	8	$\frac{64}{3}$	64	256

PROOF. Cassels (1959, Appendix; 1978, Section 12.2). □

For our congruential generators $d(\Lambda_k) = 1/N$ from (2.3).

Theorem 2.16.

$$1 \leqslant v_k l_k \leqslant \prod_1^k l_i/d(\Lambda) \leqslant (c_k)^k$$

PROOF. (i) Choose linearly independent vectors $\mathbf{f}_1, \ldots, \mathbf{f}_k \in \Lambda_k$ with $\|\mathbf{f}_i\| = l_i$, and $\mathbf{g} \in \Lambda_k^*$ with $\|\mathbf{g}\| = v_k = l_1^*$. Since $\{\mathbf{f}_i\}$ span $\mathbb{R}^k$, $\mathbf{g}^T\mathbf{f}_s \neq 0$ for at least one s. Now $\mathbf{g} = \sum t_i \mathbf{e}_i^*$, $\mathbf{f}_s = \sum s_i \mathbf{e}_i$, so $\mathbf{g}^T\mathbf{f}_s = \sum s_i t_i$ is an integer and

$$1 \leqslant |\mathbf{g}^T\mathbf{f}_s| \leqslant \|\mathbf{g}\| \, \|\mathbf{f}_s\| = v_k l_s \leqslant v_k l_k$$

by Cauchy–Schwartz.

(ii) Let Λ_{k-1} be the lattice with basis $\mathbf{f}_1, \ldots, \mathbf{f}_{k-1}$ and $H = \text{span}(\Lambda_{k-1})$. Consider parallel hyperplanes to H covering Λ_k with maximal spacing S. Let $\mathbf{f}$ be any member of Λ_k on a nearest hyperplane to the origin (excluding H). Let $\Lambda^\dagger$ be the lattice with basis $(\mathbf{f}_1, \ldots, \mathbf{f}_{k-1}, \mathbf{f})$. Then $\Lambda^\dagger \subset \Lambda_k$ and so $d(\Lambda_k) \leqslant d(\Lambda^\dagger) = Sd(\Lambda_{k-1})$ by volume = basal area × height. Thus $S \geqslant 1/Nd(\Lambda_{k-1})$, so

$$v_k \leqslant 1/S \leqslant Nd(\Lambda_{k-1}) \leqslant N \prod_1^{k-1} l_i$$

and $v_k l_k \leqslant N \prod l_i \leqslant (c_k)^k$ by Theorem 2.15. □

This result implies that $1/v_k$ and l_k are essentially equivalent measures of "granularity" of a lattice. The relation with the ratio $r = l_k/l_1$ comes from:

Theorem 2.17.

(i) For $k = 2$, $\sqrt{(r/N)} \leqslant l_2 \leqslant c_2\sqrt{(r/N)}$

$$\sqrt{(N/r)} \leqslant v_2 \leqslant c_2\sqrt{(N/r)}$$

(ii) For $k \geqslant 2$, $1 \leqslant r \leqslant Nl_k^k$

$$N^{-1/k} \leqslant l_k \leqslant c_k N^{-1/k} r^{(1-1/k)}$$

PROOF. (i) From Theorem 2.15 $1/N \leqslant l_1 l_2 = l_2^2/r = rv_2^2/N \leqslant c_2^2/N$

(ii) $1/N \leqslant l_1 \cdots l_k \leqslant l_k^k$, so $l_k \geqslant N^{-1/k}$

$c_k^k/N \geqslant l_1 \cdots l_k \geqslant l_k l_1^{k-1} = l_k^k/r^{k-1}$

$1/N \leqslant l_1 l_k^{k-1}$, so $r = l_k/l_1 \leqslant l_k^k N$ □

2.5. SHUFFLING AND TESTING

The best we have been able to do in the theoretical analysis of Sections 2.2–2.4 is to find for a limited class of generators the exact distribution of k-tuples $(U_i, \ldots, U_{i+k-1})$ over a period. Even this says nothing about the distribution of k-tuples over less than a period. For example, Fushimi and Tezuka (1983) tested pairs from the GFSR of period $2^{521} - 1$ considered by Bright and Enison (1979); although this *is* 2-distributed, the part of the period tested of length about 10,000 had considerably too many pairs near the diagonal of the unit square. We again discover philosophical problems, for such events will happen with true random numbers, and by asking for our pseudo-random numbers to conform too closely to expectation we will damage their credibility for some purposes. It does seem essential to test several subsequences from a generator with different starting points before jumping to conclusions.

There is a mistaken belief that taking seeds widely spaced apart in (X_i) and running the same congruential generator with these seeds will give "more independent" streams than sampling from a single sequence. Consider seeds X_0 and $Y_0 = X_j$. Then $Y_i = X_{i+j} = \{a^j X_i + (a^j - 1)c/(a-1)\} \bmod M$, so $\{(X_i, Y_i)\}$ lie on a lattice corresponding to the multiplier $(a^j \bmod M)$. It is entirely possible that $\{(X_i, Y_i)\}$ has much coarser structure than $\{(X_{2i}, X_{2i+1})\}$ and extremely unlikely that it has better lattice constants. If (X_i) is not thought sufficiently random to be used as the sole source of random numbers, one needs a better generator!

Shuffling

Various methods are available to modify the output of a suspect generator. They are not recommended since they are little understood, but they may provide a quick "fix" where necessary.

A. Generate output in blocks of length L, and apply a fixed permutation to each block before use. This should be sufficient to repair RANDU, for example. [See Atkinson (1980).]

B. Apply a random shuffle to (U_i). Suppose we have $T[0], \ldots, T[k-1]$ initially filled with $U_1, \ldots, U_k$, and a second pseudo-random sequence (V_i). At each step we use V_n to select a random member of T; that is, we set $J = \text{int}[kV_n]$, then return $T[J]$ and replace it by U_n. This idea is due to MacLaren and Marsaglia (1965).

C. A subtly different method to B was proposed by Bays and Durham (1976). In their method the last value output is used rather than V_n to choose

the next member of T to be replaced. Whereas B can make a sequence worse if (U_i) and (V_i) are closely related, no such examples are known for the Bays–Durham method. (They may of course exist.)

D. Given two sequences (X_i) and (Y_i) with moduli M, set $U_i = (X_i + Y_i)/M \bmod 1$. An extension of this idea to three sequences was used by Wichmann and Hill (1982). Determining the period can be tricky, as those authors found.

E. Instead of adding we could form $U_i = (X_i \text{ EOR } Y_i)/M$.

The only theoretical analysis of these schemes have been on simplified versions that may not be reliable models—Bays and Durham (1976), Brown and Solomon (1979), Nance and Overstreet (1978), and Rosenblatt (1975).

Empirical Testing

Any significance test of independence or uniformity or both can be applied to the output $(U_1, \ldots, U_n)$ of a pseudo-random number generator. Many tests have been used and it is most convenient to group them according to the property tested.

Tests for Independence

Any nonparametric test for independence can be applied to (U_i) or $(Y_i = \text{int}[U_i \times K])$ for any integer K. Often it is easiest to take K a power of 2 and so examine the first few bits of X_i.

(a) *Gaps Test for* (U_i). Fix constants $0 < \alpha < \beta < 1$ and consider the lengths of intervals for which $U_i \notin (\alpha, \beta)$. If the sequence (U_i) is independent, the distribution of lengths should be geometric with parameter $P(\alpha < U_1 < \beta) = (\beta - \alpha)$. Furthermore, independence means that successive gap lengths are independent, so we can compare observed and empirical distributions by a chi-squared test. As an example, consider Table 2.1 with $\alpha = 0.4, \beta = 0.6$. Then the gap lengths are 0, 7, 1, 0, 1, 0, 8, 1, 5, 1, 6, 7 so

$k =$	0	1	2	3	4	5	6	7	8	>8
Observed	3	4	0	0	0	1	1	2	1	0
Expected	2.4	1.92	1.54	1.23	0.98	0.79	0.63	0.50	0.40	1.61

which needs no statistical test to reject independence.

(b) *Runs Test for* (U_i). Runs up are monotone increasing subsequences; runs down are defined by replacing increasing by decreasing. There are several subtly different "runs tests" depending on whether both runs up and down are used, and on the exact definition of a run. Probably the easiest is to discard the first element of a run, so the first two runs up in Table 2.1 are (0.563, 0.624) and (0.811, 0.999). In that case the run-up lengths are independent and a chi-squared test can be used to compare their observed and expected frequencies. Tedious but elementary probability shows that $E(\text{number with run length} = k) = (n+1)k/(k+1)! - (k-1)/k!$, $k = 1, \ldots, n$ for a sequence of length n. Barton and Mallows (1965) discuss runs tests in more detail.

(c) *Permutation Tests for* (U_i). Divide (U_i) into blocks of length t, $(U_1, \ldots, U_t)$, $(U_{t+1}, \ldots, U_{2t}), \ldots$. There are $t!$ possible orderings of a block of t distinct numbers, and these should be equally probable. Counting the occurrences of all possible orderings and using a chi-squared test gives us a test for independence. This is only useful for moderate t, since we will need $n \gg t!$ We saw in Section 2.1 that the Fibonacci sequence (1.2) will fail this test for $t = 3$.

(d) *Coupon Collectors' Test for* (Y_i). Consider the lengths of sequences needed to "collect" all integers $0, \ldots, K-1$. This gives us a frequency distribution on $\{K, K-1, \ldots\}$. The probability of a length r being needed is found by combinatorial arguments (Greenwood, 1955).

Note that (a) and (d) depend on uniformity, whereas (b) and (c) work for any continuous marginal distribution of the (U_i).

Tests for Uniformity

Tests for uniformity can be any nonparametric test of a known distribution. The most commonly used are a chi-squared test based on dividing (0, 1) into intervals, and the Kolmogorov–Smirnov test $\max|F_n(x) - x|$, where $F_n(x) = (\text{number of } U_i \leqslant x)/n$, the empirical distribution function of $(U_1, \ldots, U_n)$. Its computation is discussed by Gonzalez et al. (1977).

Tests of Pairs, and k-tuples

The chi-squared test can also be applied to test the uniformity of k-tuples $\{(U_{ki}, \ldots, U_{ki+k-1})\}$, dividing $[0, 1]^k$ into a number of small regions. To do so effectively and ensure a reasonable number of counts in each cell of the test needs a very large number of observations, so this tends to be a weak test. Note that this test cannot be applied to $\{(U_i, \ldots, U_{i+k-1})\}$ since these k-tuples

are not independent. Good (1953, 1957) provides a correct modified test for pairs $\{(U_i, U_{i+1})\}$.

An alternative is to use time-series methods to examine the correlation structure of (U_i). These methods are meant for normally distributed sequences, and it may be better to apply them to $V_i = \Phi^{-1}(U_i)$, which is normally distributed if Φ is the cumulative distribution function for the normal (see Theorem 3.1). We can then test whether the correlation between U_i and U_{i+t} or V_i and V_{i+t} is zero. This is again a weak test, for lack of correlation does not imply independence.

More sensitive tests are provided by tests of the k-tuples as a point pattern. Theorem 2.6 provides one such test statistic (Ripley and Silverman, 1978) and others are described in Ripley (1981, Chapters 7 and 8).

The theoretical tests of Sections 2.2–2.4 have been found to be more powerful than empirical tests in the sense that "good" generators by the theoretical criteria have been found to fail the empirical tests no more often than would be expected by chance. Nevertheless it is always worth conducting some empirical tests to check that the generator has been implemented correctly. (Microcomputer implementations work incorrectly surprisingly often from faulty compilers or side effects of operating systems.)

We would of course expect the statistical tests to be failed occasionally by chance. In extensive investigations it is a good idea to try each test on a large number of nonoverlapping subsequences of (U_i). For each *test* we obtain an observation of either pass/fail or a significance level and can test these observations against their known distribution. Perhaps the most commonly used example of this procedure is to apply the Kolmogorov–Smirnov test of uniformity. This gives rise to a significance level P uniformly distributed on (0, 1), and the Kolmogorov–Smirnov test is applied again to the observed significance levels. An example is given by van Es et al. (1983).

2.6. CONCLUSIONS

The net effect of both theoretical analysis and empirical investigations is that a good pseudo-random number generator should:

- **(a)** use a simple algorithm and so be rapid, taking considerably less time than evaluating a logarithm;
- **(b)** be periodic with a long period, at least 2^{27} or 10^8, and take values evenly spread in [0, 1), preferably excluding zero;
- **(c)** have k-tuples for $k = 2, 3, 4$ and preferably $k \leqslant 10$ as uniformly distributed as possible in $[0, 1)^k$;
- **(d)** have been checked carefully to see that it does implement the stated algorithm.

Unfortunately a large proportion of generators in common use fail to have some of these properties, principally b and c. However, a number of generators are available with the desired properties which are fairly simple to implement.

Among congruential generators, line 2 of Table 2.4,

$$X_i = (69069X_{i-1} + 1) \bmod 2^{32},\ U_i = 2^{-32} X_i$$

has been implemented successfully in Fortran and assembler on a range of machines from 8-bit microcomputers to 64-bit supercomputers. If the NAG library is available, line 1 is an obvious choice. It is preferable to have a generator with period greater than 2^{32}, and it will often be possible to use line 11 with period 2^{48}. Implementing these generators in a high-level language is normally done using double-precision reals, which usually can represent exactly considerably larger integers than integer types. (See Appendix B.1.)

The GFSR generators represent an easier solution in environments with only limited precision arithmetic, provided a word-wise EOR operation is available. Generally we will choose at least 15-bit integers and ask for at least 4-distribution. One such recommendation is based on $p = 98$, $q = 27$, $L = 15$, which has a "granularity" of 2^{-15} and almost exact independence in up to six dimensions. To initialize it we take $b_i = b_{i-98}$ EOR b_{i-71} and make up Y_i out of $(b_i, b_{i+6}, b_{i+12}, \ldots, b_{i+84})$, $i = 1, \ldots, 98$ and then use algorithm 2.2.

It is always helpful to have two or more generators available and to run important simulations using each, to reduce the likelihood that anomalous results are due to the quirks of the generator used.

*2.7. PROOFS

We need three lemmas for the results of Section 2.2.

Lemma A. Let $p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ be the prime factorization of M. Then the period of any congruential generator with modulus M is the lowest common multiple of its periods modulo $p_i^{\alpha_i}$.

PROOF. By induction we need only consider $M = m_1 m_2$ with $\gcd(m_1, m_2) = 1$. Let $Y_j = X_j \bmod m_1$, $Z_j = X_j \bmod m_2$, with X_0 any value in the periodic cycle. Suppose X_i, Y_i, Z_i have periods d, d_1, and d_2. Then $Y_j = Y_0$ iff j is a multiple of d_1. Since $X_d = X_0$, $Y_d = Y_0$, so d is a multiple of d_1. By symmetry it is a multiple of d_2 and hence of $\operatorname{lcm}(d_1, d_2) = l$. Now $X_l - X_0$ is a multiple of both m_1 and m_2, since $Y_l = Y_0$, $Z_l = Z_0$, hence of M. Thus $X_l = X_0$ and $d = l = \operatorname{lcm}(d_1, d_2)$. □

Lemma B (Fermat, 1640). Suppose p is prime and $0 < a < p$. Then $a^{p-1} \equiv 1 \mod p$.

PROOF. Consider $\{ra \bmod p | 0 \leqslant r < p\}$. This is a set of p distinct numbers ($ra \equiv sa \bmod p$ implies $r = s$), so it is $\{0, \ldots, p-1\}$. Thus $\{a \bmod p, 2a \bmod p, \ldots, (p-1)a \bmod p\} = \{1, \ldots, p-1\}$ and

$$\prod_1^{p-1} ra \equiv \prod_1^{p-1} r \bmod p$$

whence $a^{p-1} \equiv 1 \bmod p$. □

Lemma C. (X_{ki}) has period $d/\gcd(k, d)$ if (X_i) has period d.

PROOF. $X_{ki} = X_0$ if and only if ki is a multiple of d if and only if i is a multiple of $d/\gcd(k, d)$. □

Proof of Theorem 2.3.

(i) Since $c = 0$, $X_i = a^i X_0 \bmod M$. Let $p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ be the prime factorization of M. Then $(X_j \bmod p_i^{\alpha_i})$ has period at most $(p_i^{\alpha_i} - 1)$ (omitting zero), so (X_j) has period at most $\prod_1^r (p_i^{\alpha_i} - 1)$, $< M - 1$ unless $r = 1$.

(ii) Suppose $M = p^\alpha$ for a prime p, $\alpha > 1$. Then $(X_i \bmod p)$ has period d dividing $(p-1)$ by lemma B. Thus $(X_{id} - X_0)$ is a multiple of p, whence (X_{id}) has period at most $p^{\alpha-1}$, and (X_i) has period at most $dp^{\alpha-1} < M - 1$.

(iii) Suppose M is prime. By lemma B the period d divides $M - 1$. Suppose $M - 1 = nd$, and that p is a prime factor of n. Then $s = (M-1)/p$ is a multiple of d, and $X_s = a^s \bmod M = X_0$, so $a^s \bmod M = 1$. Hence if $n > 1$, a is not a primitive root. Conversely, if a is not a primitive root, let $s = (M-1)/p$, when $X_s = a^s X_0 \bmod M = X_0$, so the period divides s and is less than $M - 1$. □

Proof of Theorem 2.4

Let $b = a^k \bmod M$. Then (X_{ki}) corresponds to multiplier b, and has period $(M-1)$ if and only if $\gcd(k, M-1) = 1$ by lemma C. □

Proof of Theorem 2.1

(i) By lemma A we can confine attention to $M = p^\alpha$, p prime.

(ii) If $c = 0$, the period is at most $M - 1$ by the proof of Theorem 2.3. We assume $c > 0$.

(iii) If $a = 1$, $X_i = (X_0 + ic) \bmod M$, so $X_i = X_0$ iff $ic \bmod M = 0$ iff i is a multiple of $M/\gcd(c, M)$, the period. We can now assume $M = p^\alpha$, $c > 0$, $a > 1$. We have

$$X_{i+j} = \left\{ a^j X_i + \frac{(a^j - 1)c}{(a - 1)} \right\} \bmod M \text{ for } j \geqslant 0$$

(iv) Suppose the period is M. Then we may take $X_0 = 0$. Some $X_i = 1$, when $(a^i - 1)c/(a - 1) \equiv 1 \mod M$ and hence $\gcd(c, M) = 1$ [for $(a^i - 1)/(a - 1) = 1 + a + \cdots + a^{i-1}$, an integer]. Also, $X_M = 0$, so $(a^M - 1)c/(a - 1)$ is a multiple of $M = p^\alpha$. If $a \not\equiv 1 \mod p$ this implies $a^M - 1 \equiv 0 \mod M$, hence $a^M \bmod p = 1$. However, lemma B shows $a^p \equiv a \mod p$, so $a^M \equiv a \mod p$. We conclude $a \equiv 1 \mod p$.

(v) Suppose $M = 2^\alpha$, $a \geqslant 2$. If $a \equiv 1 \bmod 2$ but $a \not\equiv 1 \bmod 4$, $a \equiv 3 \bmod 4$. Then

$$X_i = \{a^2 X_{i-2} + (a + 1)c\} \bmod M$$

hence $X_2, X_4, \ldots$ are multiples of $(a + 1)c \bmod M = 4c \bmod M$. Thus (X_{2i}) takes at most $M/4$ values, and (X_i) has period at most $M/2$.

This establishes necessity. For sufficiency assume $M = p^\alpha$. We will use induction on α. For $\alpha = 1$, $a \equiv 1 \bmod M$, whence $(X_i) = (ic \bmod M)$ starting from zero, which has period M. Now suppose the theorem holds for $M = p^{\alpha-1}$. Fix $X_0 = 0$. From the conditions $a = 1 + qp^e$ for $p^e > 2$. Thus

$$a^p = (1 + qp^e)^p = 1 + pqp^e + \cdots + q^p p^{ep} = 1 + sqp^{e+1}$$

for an integer s with $s \equiv 1 \bmod p$. Now $X_p = (a^p - 1)c/(a - 1) \bmod p^\alpha = sqp^{e+1}c/qp^e \bmod p^\alpha = scp \bmod p^\alpha$. By induction we find X_{ip} is a multiple of p for all i. Let $Y_i = X_{ip}/p$. Then

$$\begin{aligned} Y_i &= \{a^p Y_{i-1} + (a^p - 1)c/(a - 1)p\} \bmod p^{\alpha-1} \\ &= \{a^p Y_{i-1} + sc\} \bmod p^{\alpha-1} \end{aligned}$$

Applying the theorem for modulus $p^{\alpha-1}$ shows Y_i has period $p^{\alpha-1}$ since $s \equiv 1 \bmod p$, so $\gcd(sc, p^{\alpha-1}) = 1$. We can deduce $X_M = 0$, so (X_i) has period dividing $M = p^\alpha$. However $X_{p^{\alpha-1}} = pY_{p^{\alpha-2}} \neq 0$, and (X_i) has period M. □

Proof of Theorem 2.2

(i) Suppose X_0 is even. Then $X_0 = 2^r Y$, for Y odd, and

$$(X_i 2^{-r}) = a(X_{i-1} 2^{-r}) \bmod 2^{\beta-r}$$

reduces to the case of an odd seed.

(ii) Suppose a is even. Then $X_\beta = a^\beta X_0 \bmod 2^\beta = 0$. We now suppose X_0 and a are odd, so X_i is always odd. Choose X_0 as the smallest value in the period. Then

$$(X_i - X_0) = \{a(X_{i-1} - X_0) + (a-1)X_0\} \bmod 2^\beta$$

and $(a-1)$ is even, so $X_i - X_0$ is even, $= 2Y_i$ say.

$$Y_i = \{aY_{i-1} + a'X_0\} \bmod 2^{\beta-1}$$

where $a' = (a-1)/2$. From Theorem 2.1 this has period less than $M/2$, for either a' is even or $a \not\equiv 1 \bmod 4$.

(iii) Suppose $a \equiv 1 \bmod 4$, so $a = 1 + 4b$. Then $a' = 2b$ and Y_i is even. Let $Z_i = Y_i/2$, with

$$Z_i = \{aZ_{i-1} + bX_0\} \bmod 2^{\beta-2}$$

From Theorem 2.1 this has period $M/4$ if b is odd ($a \equiv 5 \bmod 8$) otherwise less than $M/4$.

(iv) Suppose $a = 3 + 4b$. Then

$$Y_i = \{a^2 Y_{i-2} + a'(a-1)\} \bmod 2^{\beta-1}$$

and $a'(a+1) = 4(b+1)(2b+1)$ is a multiple of 4. Hence so is Y_{2i}. Let $W_i = Y_{2i}/4$, with

$$W_i = \{a^2 W_{i-1} + (b+1)(2b+1)\} \bmod 2^{\beta-3}$$

Now $a^2 \bmod 4 = (16b^2 + 24b + 9) \bmod 4 = 1$, so (W_i) has period $M/8$ only if b is even. Thus if $a \equiv 7 \bmod 8$, (Y_i) has period less than $M/8$. If $a \equiv 3 \bmod 8$, (Y_{2i}) are multiples of 4 with period $M/8$, whereas (Y_{2i+1}) are odd. Thus (Y_i) and (X_i) have period $M/4$.

(v) Suppose $a \equiv 5 \bmod 8$. Then $X_i = X_0 + 4Z_i$, so the smallest value in the sequence, $b = (X_i \bmod 4)$ for any i. Then $U_i = X_i/M = b/M + \{Z_i/(M/4)\}$ as required. □

Proof of Theorem 2.5

Let $\oplus$ denote addition modulo M.

(i) $\{U_i\} = \{(X_0 \oplus s)/M \mid s = 0, 1, \ldots, M-1\}$, so

$$\begin{aligned}\{(U_i, \ldots, U_{i+k-1})\} &= \{(X_0 \oplus s, a(X_0 + s) \oplus c, \ldots)/M \mid s = 0, \ldots, M-1\} \\ &= [0,1)^k \cap \{(X_0 + s, aX_0 + as + c + t_2 M, \ldots)/M \mid s, t_2, \ldots, t_k \text{ integer}\} \\ &= [0,1)^k \cap \{(X_0, \ldots, X_{k-1})/M + \Lambda_k\}\end{aligned}$$

(ii) $\{(U_i, \ldots, U_{i+k-1})\} = \{(s, as, a^2s, \ldots)/M \mid s = 1, \ldots, M-1\}$
$= (0, 1)^k \cap \{(s, as + t_2M, \ldots)/M \mid s, t_2, \ldots, t_k \text{ integer}\}$
$= (0, 1)^k \cap \Lambda_k$

For the nonoverlapping k-tuples we note

$$\{U_{ki}\} = \{(U_0 + s/N) \bmod 1\}$$

by lemma C in the cases claimed and modify the above accordingly.

Proof of Theorem 2.12

(i) Suppose $\mathbf{g} \in \Lambda_k$, $\mathbf{g} = t_1\mathbf{e}_1 + \cdots + t_k\mathbf{e}_k$. If $t_j \neq 0$ we will show $\|\mathbf{g}\| \geq \|\mathbf{e}_j\|$. This establishes $\|\mathbf{e}_i\| = l_i$ for $i = 1, \ldots, k$.

(ii) By changing $\mathbf{e}_i$ to $-\mathbf{e}_i$ if necessary we may assume all $t_i \geq 0$. Let $T = \max t_i$. We proceed by induction on T. If $T = 1$ then $\|\mathbf{g}\| \geq \|\mathbf{e}_j\|$ by hypothesis (ii). Suppose the result is true for $\max t_i \leq T - 1$. Let $m = \min\{t_i | t_i > 0\}$ and $r = \max\{i | t_i = m\}$. Define $\mathbf{E}$ as the sum of $\mathbf{e}_i$ over all indices i except r with $t_i > 0$. Let $\mathbf{h} = \mathbf{g} - m\mathbf{E}$. We will show $\|\mathbf{g}\| \geq \|\mathbf{h}\|$. Now *either* $\max h_i \leq T - 1$ and $\|\mathbf{h}\| \geq \|\mathbf{e}_j\|$ *or* all nonzero t_i were equal to m, when $\mathbf{h} = m\mathbf{e}_r$, so $\|\mathbf{h}\| \geq \|\mathbf{e}_j\|$. In either case $\|\mathbf{g}\| \geq \|\mathbf{h}\| \geq \|\mathbf{e}_j\|$.

(iii) Fix $s < t$. Then $\|\mathbf{e}_s + \mathbf{e}_t\|^2 \geq \|\mathbf{e}_t\|^2$, whence $2\mathbf{e}_s^T\mathbf{e}_t \geq -\|\mathbf{e}_s\|^2$ and hence $\mathbf{e}_s^T\mathbf{e}_t \geq -\frac{1}{2}\|\mathbf{e}_s\|^2$ whether $s < t$ or $s > t$. Now consider

$$(\mathbf{h} - m\mathbf{e}_r)^T\mathbf{E} = \sum_{i \neq r} h_i\mathbf{e}_i^T\mathbf{E} = \sum_{i \neq r \text{ in } E} h_i\mathbf{e}_i^T\mathbf{E}$$

$$= \sum h_i\{\|\mathbf{e}_i\|^2 + (\text{up to } 2)\,\mathbf{e}_i^T\mathbf{e}_j\} \geq 0 \text{ since } h_i \geq 0$$

(iv) $\|\mathbf{g}\|^2 - \|\mathbf{h}\|^2 = \|\mathbf{h} + m\mathbf{E}\|^2 - \|\mathbf{h}\|^2$

$$= 2m\mathbf{h}^T\mathbf{E} + m^2\|\mathbf{E}\|^2 = 2m(\mathbf{h} - m\mathbf{e}_r)^T\mathbf{E} + m^2\{\|\mathbf{E} + \mathbf{e}_r\|^2 - \|\mathbf{e}_r\|^2\}$$

which is nonnegative by (iii) and by hypothesis. □

EXERCISES

2.1. Complete the sequence (1) and show that it eventually repeats. Try other starting values. Is the behavior starting from 8653 typical?

2.2. Investigate all starting values for the two-digit decimal and eight-bit binary middle square methods.

2.3. Prove that for the *Fibonacci* recursion (2) that $U_{i-2} < U_i < U_{i-1}$ never occurs, whereas this event has probability 1/6 for random numbers.

2.4. Try implementing the generator $U_i = 1013U_{i-1} \bmod 1$ in floating-point arithmetic and via (4) with $M = 10^5$ and $M = 2^{16}$. What periods are obtained?

2.5. Compute the outputs of the following congruential generators with $M = 64$. (a) $a = 29$, $c = 17$. (b) $a = 9$, $c = 1$. (c) $a = 13$, $c = 0$. (d) $a = 11$, $c = 0$.

2.6. Find the periods corresponding to multipliers 10, 12, 16, and 18 in a multiplicative congruential generator with $M = 67$.

2.7. Show for $M = r^\beta - s$, $Y = aX_{i-1} + c$, that $Y \bmod r^\beta + s(Y \operatorname{div} r^\beta) < (1 + s)M$.

2.8. Plot the lattices of (U_i, U_{i+1}) and (U_{2i}, U_{2i+1}) for the examples of Exercises 2.5 and 2.6.

2.9. Show that Algorithm 2.1 works.

2.10. Generate the shift-register sequence with $p = 7$, $q = 1$. Form the Tausworthe sequence with $t = L = 3$ and show that it is 2-distributed.

2.11. Generate the GFSR with $p = 7$, $q = 1$, starting 6, 5, 4, 3, 2, 1, 0.... How does it fail to be 2-distributed?

2.12. Find starting values for the GFSR with $p = 7$, $q = 1$, $L = 3$ by the delay method and verify that it is 2-distributed both via Theorem 2.9 and by generating the sequence.

2.13. Find a better multiplier than 75 for $M = 2^{16} + 1$, $c = 0$.

2.14. Compute the lattice constants where appropriate for the examples of Exercises 2.5 and 2.6 in two and three dimensions.

2.15. Try the effect of the Bays–Durham shuffling algorithm on Table 2.1.

2.16. Apply the gaps and runs tests and the permutation test for $k = 3$ to both Table 2.1 and the output from Exercise 2.15.

2.17. Test empirically the pseudo-random number generators on all the computers you use. If possible, find out the algorithms claimed to be used, check that these are implemented as stated and pass appropriate theoretical tests. Alternatively, replace these generators with ones of known quality, and test your implementations.