Discrete Applied Mathematics 20 (1988) 145-164
North-Holland

THE PERIOD OF THE FIBONACCI RANDOM NUMBER GENERATOR

J.D. PARKER

Department of Computer Science, Boston College, 432 Fulton Hall, Chestnut Hill, MA 02167, USA

Received 6 December 1983
Revised 6 March 1987

Smith, Green, and Klem introduced the Fibonacci RNG in [7]. A starting vector of k integers is chosen, and new numbers are generated by the recurrence $r_n \equiv r_{n-1} + r_{n-k} \pmod M$. For a prime M and some choices of the parameter k, any non-zero initial vector v gives a sequence with a period of $M^k - 1$. However, in most cases, different initial values give rise to very different periods. This behavior was noted by the authors, but left unexplained. In this paper we review how sequences with short periods arise, and provide an algorithm that selects different starting vectors that give a maximal period.

1. Introduction

The pseudo-random number generator (RNG) in widest use today is the multiplicative generator proposed by Lehmer in [13]:

$$r_n \equiv (a \times r_{n-1} + b) \pmod M,$$

known as the Linear Congruential Generator. The period depends upon the choice of M, a, b, and the starting 'seed' r_1, but it is never more than M. Although it has many strong attributes, it exhibits n-space non-uniformity as discussed by Coveyou and MacPherson [4], and Marsaglia [15], making it unsuitable for some applications which require the selection of random n-tuples.

This observation has sparked an interest in RNG's with n-space uniformity. One such RNG is the Tausworthe generator introduced in [19],

$$r_n = r_{n-j} \oplus r_{n-k}.$$

The Tausworthe RNG keeps a vector v of the k previous random numbers. Each element of the vector is a string of bits, and the next number in the sequence is computed with a bitwise exclusive-or. Since it avoids multiplication, the Tausworthe RNG is faster than the Lehmer generator. The period of the generator is fixed by the choice of parameters j and k, and can be as large as $2^k - 1$. The period is not a function of the word size, as the bits within a word have no interaction.

This paper discusses another fast generator introduced in the fifties, the Fibonacci

RNG

$$r_n \equiv r_{n-1} + r_{n-k} \pmod{M}, \tag{1.1}$$

introduced by Smith, Green, and Klem [7]. For a prime M and some choices of the parameter k, any non-zero initial vector v gives a unique sequence with a period of $M^k - 1$. This is much longer than the period of either of the random number generators mentioned above. However, if M is composite, or for some choices of degree k, different initial values give rise to distinct sequences with different periods. This behavior was noted by the authors, but left unexplained.

Our purpose in this paper is twofold: first, we review how sequences with short periods arise, and then we provide an algorithm that selects different starting values that give a maximal period for any fixed choice of parameters.

In Section 2, we review related work. In Section 3, we give the original algorithm, and discuss the differences in period observed by the authors. Section 4 gives an example illustrating how short periods arise. We use linear algebra to study the maximal period of the generator in Section 5, and in Section 6 we study the effect that factors of the characteristic polynomial have on the period. In Section 7 we discuss the choice of parameters, and present our algorithm for selecting initial values. Section 8 uses the run test to look at the 'randomness' of the sequence. We summarize our conclusions in Section 9.

2. Relation to previous work

Random number generators have always been important in experimental design, computer simulations, and cryptography. Before 1927, scientists who needed random numbers would produce them 'by hand', rolling dice or flipping coins. L.H.C. Tippett [20] produced a table of random numbers from figures in census reports to simplify this process. It was not long before people were questioning the randomness of the numbers in Tippett's tables (Kendall [10] and others), and better means of producing numbers we sought. One technique was to design mechanical or electrical devices to produce the random numbers. This was used by the RAND Corporation to produce their table of a million digits.

The generation of random numbers was an early application of general purpose computers. The publications in this area follow the same pattern set by Tippett's paper. A method is conceived and tested. It passes the investigators' tests, and is proposed for general use. Soon an alternate test is proposed that the generator fails to pass, and the cycle begins anew. The interested reader may trace the rise and fall of proposed generators, such as the mid-square method, in Sowey's [16] bibliography on random numbers.

The Lehmer RNG was one of the first discussed, and it has aged relatively well. It is a generator of degree 1: the current number depends only upon the previous number. There have been doubts raised about its generation of n-tuples, [4], [15] and speculation that this problem is endemic with any generator of degree 1. The concern is that when the random number generator is used to produce a sequence of vectors with n entries, the vectors will not be uniformly distributed in n-space. For example, a generator of degree 1 cannot produce the same number twice in a row without entering a loop of length 1, thus sequences with long periods must avoid the diagonal in 2-space.

Knuth [11] described a technique for generating a random sequence with n-space uniformity for all n. Essentially, he generates all tuples in a prescribed order, starting with the digits, and then systematically generating all tuples of length two, etc. It is not practical for several reasons: the algorithm is not suitable for rapid generation of numbers, and the sequence exhibits too much regularity to be considered random.

A candidate for a practical generator with acceptable n-space uniformity is the generalized Fibonacci random number generator, of degree k, first discussed in print in Green, Smith, and Klem's 1959 paper [7]. The authors mention an attribution of the method to A. van Wijngaarden, and cite previous studies by Taussky and Todd [18] and Duparc, Lekkerkerker, and Peremans [6], of the special case when $k = 2$. This case has always been a good example of a bad generator: see for example Knuth [12] or Bratley, Fox, and Schrage [1]. We must pick a value for k that is bigger than 2 to obtain decent sequences. The generator has suffered from the sins of this special case: Carrol and McLelland [3] are among the few that admit using the generator.

As well as altering the degree, we may vary the choice of recurrence relation. The interested reader will find that our results and algorithms apply to general linear recurrence relations of the form

$$r_n \equiv a_1 r_{n-1} + a_2 r_{n-2} + \cdots a_k r_{n-k} \pmod{M}.$$

For example, if we pick a recurrence similar to the Tausworthe generator, we obtain a family of additive congruential generators of the form:

$$r_n \equiv r_{n-j} + r_{n-k} \pmod{M}.$$

We still have a fast generator, but we gain flexibility in selecting the characteristic polynomial. Dieter [5] suggests yet another variant:

$$r_n \equiv r_{n-1} + a r_{n-2} \pmod{p},$$

with a prime modulus p. This generator is slower, as it requires a multiplication, and it has a shorter period since the degree is small, but it only needs to store two previous values, and for the proper choice of modulus and constant a, it produces well-behaved sequences.

Green et al. address the issue of the *randomness* of the sequence rather than its period. They show empirically that the numbers generated are uniformly distributed, have no significant serial correlation, and pass the run test and the poker test if the degree k is at least 16. They recommend 'decimation', or discarding elements of the sequence, to assure better empirical results. They direct the reader interested in the period to an excellent paper by Zierler [21]. This citation has persistently accompanied discussion of this generator, despite a lack of relevance. Zierler discusses the case of the *field* GF(p^α), while we are confronted with the *ring* Z_{p^α}. If M contains a power of a prime, several of Zierler's results do not generalize. In Section 4 we give a counter-example to a 'theorem' cited in one source based upon this confusion.

In this paper, we review criteria for predicting the period of the RNG, given any choice of parameters. We base our derivation of these well-known results upon classic work of Hall [8]. We then present an original technique for selecting initial values that give sequences with maximal periods.

The Fibonacci generators bear a strong family resemblance to the Tausworthe RNG. For example, Sedgewick [17] introduces additive congruential methods after a discussion of linear feedback registers, mechanical devices used in cryptographic encryption machines and the model for the Tausworthe RNG. To calculate the maximal period, we may view the Tausworthe RNG as a special case of the general linear recurrence relations with a modulus of 2. Similar issues are relevant to the two generators: an irreducible characteristic polynomial and a proper choice of initial values leads to an optimal period. However, our discussion differs from the classical treatment of the Tausworthe generator in the following four ways.

(1) In constructing a Tausworthe generator, it is customary to choose parameters so that the characteristic polynomial is irreducible, or better yet, primitive. In the Tausworthe generator with a primitive characteristic polynomial, any choice of a non-trivial initial vector gives the same period, $2^k - 1$. In this paper, we consider polynomials which factor. Factoring creates different classes of sequences, with potentially different periods.

(2) The presence of zero-divisors of the ring Z_{p^α}, if $\alpha > 1$, is a second cause of short periods. In the Tausworthe generators, all calculations are done over the field Z_2, so there are never zero-divisors.

(3) Given a primitive characteristic polynomial and non-zero initial values, the Tausworthe generator produces a sequence of maximal period. However, some sequences have poor distribution. For example, let the word size be 3, and apply the Tausworthe generator to the *unit* vector $(r_1, r_2, \ldots, r_k) = (0, 0, \ldots, 1) = (000, 000, \ldots, 001)$. We generate a sequence in which the most significant bits of every number produced are zero, since each bit position is autonomous. To obtain a uniform distribution, it is necessary that the initial vector is linearly independent, as shown by Lewis and Payne [14], and applied by Bright and Enison [2] to cryptography. The test proposed in [14] discards the unit vector. However, the Fibonacci random number generator uses addition, and carries propagate from the low order bits to the high. The Fibonacci recurrence applied to the unit vector yields a sequence with an acceptable distribution. We have a different concern in selecting our initial vector: if the characteristic polynomial factors, the wrong choice of initial vector leads to less than optimal periods.

(4) Lewis and Payne [14] suggest taking the determinant of a $k \times k$ **binary** matrix whose columns are the binary representations of the k initial values, as a test of the *distribution* of a sequence. We observe that the determinant of a $k \times k$ **integer** matrix (6.1) whose entries are the initial $2k - 1$ random numbers is a test of the *period* of the sequence. These two tests differ in more than their object: the unit vector passes our test and fails theirs.

3. The generator

In the generalized Fibonacci generator introduced by Green, Smith, and Klem [7], the user picks an initial vector $v = (r_1, r_2, \ldots, r_k)$ by calling the procedure **random-init**. The procedure uses a fixed vector, or a vector picked at 'random' by the system clock or another generator. Successive terms in the sequence are generated using procedure **random** given below. Since we only use addition to compute the pseudo-random numbers, the generator is relatively fast.

```
const
    modulus = $2^\alpha$;
    degree = $k$;
type
    vector = array [0··degree−1] of integer;
    index = 0··degree;
var
    seed: integer;            (*global variable for Lehmer RNG*)
    $v$: vector;              (*Last $k$ random numbers*)
    rand-index: index;        (*Index to most recent random number*)

function Lehmer (max: integer): integer;
        (*Use a Lehmer random number generator to alter global seed*)
        (*Return a number in the range 0··max−1*)
    begin
        seed := (seed * multiplier + increment) mod modulus;
        Lehmer := trunc ((seed/modulus)*max);
    end;

procedure random-init (var $v$: vector; var $j$: integer);
        (*Set up the vector $v$*)
    var $i$: integer;
    begin
        seed := clock mod modulus;        (* Set seed to the system clock*)
        for $i$ := 1 to degree do
            $v[i]$ := Lehmer (modulus);
        rand-index := 0;                    (*init global index i*)
    end;    (*random-init*)

function random: integer;
        (*Compute the next random number*)
    var i: integer;
    begin
        i := rand-index;
        rand-index := (rand-index + 1) mod degree;
        v[rand-index] := (v[rand-index] + v[i]) mod modulus;
        random := v[rand-index]
    end;
```

We are concerned in this paper with the period of the generator. The sequence $(r) = r_1, r_2, r_3, \ldots$ is said to have period P if P is the smallest positive integer such that $r_{n+P} = r_n$ for all n. Green, Smith and Klem studied generators with a modulus $M = 2^\alpha$ equal to a power of two. As we will see in Section 5, these generators have a maximal period of $(2^k - 1)(2^{\alpha-1})$ at best. For the right initial values, the period is usually quite large. Unlike the Tausworthe generator, the period increases with the word size α, and unlike the Lehmer generator, all non-zero k-tuples appear in some sequence. Thus we may encounter runs of the same number, $r_{i+1} = r_{i+2} = \cdots = r_{i+j}$, if $j < k$. This will never happen with a non-trivial Lehmer RNG.

But an increase in the word size or in the degree does not always lengthen the period of the Fibonacci RNG, and the same generator will give sequences with different periods, depending upon the starting value. For example, the authors note that when the degree is 15, the largest sequence has a period of $(2^{15} - 1)(2^{\alpha-1})$, while a degree of 16 gives a maximal period of $(2^8 - 1)(2^{\alpha-1})$. In fact, the generator of degree 16 and period 2^α can produce sequences with a period as short as 85. The authors express the period of their generator as a function of the degree k and the modulus M, while noting that it also depends upon the choice of an initial vector v, and state that "for some [choices of v] the period is less by a factor of 2, or, rarely, by a factor of 4 or 8, or more. The conditions under which this reduction occurs are not fully understood. (The problem is exceedingly complex analytically, and the occurrences are too rare to study empirically.)"

Though we do not have a closed form expression for the period of the generator as a function of the degree and the modulus, explaining how the period is reduced is quite simple. First we review briefly why some initial vectors produce shorter sequences.[1] Then we give an algorithm that will generate a large number of 'random' initial vectors that give sequences of maximal length.

[1] For example, the difference between degrees 15 and 16 is that $x^{15} - x^{14} - 1$ is primitive, while $x^{16} - x^{15} - 1 \equiv (x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + 1)(x^8 + x^5 + x^3 + x^2 + 1)$.

4. Examples

To give the reader a taste of the varied causes of short periods, and to provide a fund of examples for later use, we consider an extended example. Let $k = 3$, $p = 3$, and $M = p$ (and later p^2), giving the relation

$$r_{n+3} \equiv r_{n+2} + r_n \pmod{M}. \tag{4.1}$$

Any initial vector of length k might be used as a starting seed. Since $M^k - 1 = 26$, there are 26 non-zero vectors of length 3. If we pick a initial vector and continue to apply (4.1), we obtain a sequence of integers which will eventually repeat. There are 4 distinct sequences:

Table 1

Sequence	Period	
$(w) = 0, 0, 1, 1, 1, 2, 0, 1, \ldots$	8	(4.2)
$(u) = 0, 0, 2, 2, 2, 1, 0, 2, \ldots$	8	(4.3)
$(y) = 1, 2, 1, 2, \ldots$	2	(4.4)
$(z) = 0, 1, 2, 2, 0, 2, 1, 1, \ldots$	8	(4.5)

Associated with any linear recurrence relation is a characteristic polynomial. The characteristic polynomial of (4.1) is

$$f(x) = x^3 - x^2 - 1 \equiv (x+1)(x^2+x+2) \pmod 3.$$

(We define the relation $\equiv$ between polynomials in Section 6.) The reducibility of this polynomial creates sequences that satisfy a relation of lower degree than $k = 3$. The sequences are (y), which satisfies

$$r_{n+1} + r_n \equiv 0 \pmod 3$$

with characteristic polynomial $(x+1)$, and (z), which satisfies

$$r_{n+2} + r_{n+1} + 2r_n \equiv 0 \pmod 3$$

with characteristic polynomial $x^2 + x + 2$.

4.1. Increasing the modulus

If we increase the modulus to $M = 3^2$, then the period of the sequence (w) triples.

$$(w) = 0, 0, 1, 1, 1, 2, 3, 4, 6, 0, 4, 1, 1, 5, 6, 7, 3, 0, 7, 1, 1, 8, 0, 1, \ldots$$

This phenomenon is quite general. If the sequence containing the unit vector $(0, 0, \ldots, 1)$ has period P over Z_p, then it will usually have period Pp over Z_{p^2}, and period Pp^α over $Z_{p^{\alpha+1}}$. See Corollary 5 for a precise statement.

Table 2

Sequence	Period	
$p(y) = 3, 6, 3, \ldots$	2	(4.6)
$(y') = 1, 5, 7, 8, 4, 2, \ldots$	6	(4.7)
$(y'') = 1, 2, 1, 2, 4, 5, 7, 2, 7, 5, 7, 5, 1, 8, 4, 5, 4, 8, 4, 8, 7, 2, 1, 8, \ldots$	24	(4.8)

But some odd things happen to the period of the sequence (y). Three 'generalizations' are shown above. Note that $(y') \equiv (y'') \equiv (y) \pmod 3$.

If the sequence (v) has a period P over Z_p, then $p^\alpha(v)$ is a sequence of period P in $Z_{p^{\alpha+1}}$. The period of the sequence (v) over $Z_{p^{\alpha+1}}$ is more problematic. We see above that it may be longer than $p^\alpha P$, unlike the unit sequence above.

5. The vector space Z_M^k

In general, our modulus M will factor as a product of powers of primes,

$$M = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_l^{\alpha_l}.$$

The period of the sequence, mod M, is the least common multiple (lcm), of the periods, mod $p_i^{\alpha_i}$. Thus, without loss of generality, we will consider $M = p^\alpha$, and return to the general case in Section 8.

The behavior of the Fibonacci RNG depends only upon the previous k items in the sequence. There are M^k possible vectors of length k, each an element of the vector space $V^k = Z_M^k$. In general, we may view a sequence (r) as represented by a sequence of vectors

$$v_i \overset{\text{def}}{=} (r_i, r_{i+1}, \ldots, r_{i+k-1}) \in V^k,$$

and the RNG as a linear transformation $T\colon V^k \to V^k$ defined by the $k \times k$ companion matrix

$$T = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 1 \end{pmatrix},$$

with action

$$Tv_1 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{k-1} \\ r_k \end{pmatrix} = \begin{pmatrix} r_2 \\ r_3 \\ \vdots \\ r_k \\ r_1 + r_k \end{pmatrix} = \begin{pmatrix} r_2 \\ r_3 \\ \vdots \\ r_k \\ r_{k+1} \end{pmatrix} = v_2.$$

We define the **orbit** of a vector v to be the set of vectors $\{v_i\} = \{T^i v, i \in Z\}$. Since T is non-singular, the orbits form equivalence classes in the vector space V^k, and the zero vector $z = (0, 0, \ldots, 0)$ forms a sequence with a single element. The period of the sequence represented by v is simply the size of the orbit of v.

This gives an easy upper bound on the period of a sequence.

Lemma 1. *The largest period of the Fibonacci random number generator of degree k, modulo M, is less than or equal to $M^k - 1$.*

Proof. The orbit must lie in Z_M^k, and cannot include the origin. □

Theorem 2. *Let P be the order of the companion matrix T, mod M, and let P' be the period of the unit sequence (w), mod M. Then $P = P'$.*

Proof. (We use the special properties of (1.1). See Hall [8] for a proof of the general case.) Since $T^P \equiv I \,(\mathrm{mod}\, M)$, we have $w_{P+i} = T^P w_i \equiv w_i$. Thus $P' \mid P$. Assume that $P > P'$. Then for all i, $T^{P'} w_i \equiv w_i$, but $T^{P'} \not\equiv I$. Then $T^{P'}$ must alter some vector in V^k. However, the standard basis vectors are terms in the unit sequence (w) given by $w_P, w_{P-1}, w_{P-2}, \ldots, w_{P-k+2}, w_1$ (see, for example, sequence (4.2)). Since $T^{P'}$ fixes the standard basis, $T^{P'} \equiv I$, and $P = P'$ by the minimality of P. □

Corollary 3. *The period of any sequence (r) divides the period P of the unit sequence.*

Proof. We have seen that $T^P \equiv I \,(\mathrm{mod}\, M)$, so $T^P r \equiv r \,(\mathrm{mod}\, M)$. □

Theorem 4. *If the linear transformation T has period P over Z_p, and a longer period over Z_{p^2}, then the period of T over Z_{p^α} is $Pp^{\alpha-1}$.*

Proof. Since $T^P \equiv I \,(\mathrm{mod}\, p)$, we may decompose T^P as the sum $T^P = I + pN$. By the binomial theorem, $T^{Pp} = (I + pN)^p \equiv I + p^2 N \equiv I \;(\mathrm{mod}\, p^2)$, so the period of T over Z_{p^2} must divide Pp. But this period is greater than P by assumption, and p is prime, so the period must be exactly Pp. □

The same reasoning shows that $T^{Pp^{\alpha-1}} \equiv I \;(\mathrm{mod}\, p^\alpha)$. Now we must show that the period is not shorter than $Pp^{\alpha-1}$. Since the period must divide $Pp^{\alpha-1}$, and it contains P as a factor, we may assume without loss that the period is $Pp^{\alpha-2}$. The period of T increases as M goes from p to p^2 if and only if the matrix N does not contain a factor of p. By the decomposition, $T^{Pp^{\alpha-2}} = (I + pN)^{p^{\alpha-2}} \equiv I + p^{\alpha-1}N$ $(\mathrm{mod}\, p^\alpha)$. If N does not contain a factor of p, then $p^{\alpha-1}N$ is not the identity. This shows that the period of T is exactly $Pp^{\alpha-1}$ over Z_{p^α}.

If N does contain a factor, $N = pN'$, then $T^P \equiv I + p^2 N' \equiv I \,(\mathrm{mod}\, p^2)$, and the period does not increase as we move from Z_p to Z_{p^2}. Consider the largest power of p that divides N. If $p^b \mid N$ but $p^{b+1} \nmid N$, then the period of T over $Z_{p^{a+b}}$ will be Pp^α, by a third application of the binomial theorem.

Corollary 5. *If the unit sequence has period P over Z_p, and a longer period over Z_{p^2}, then it has a period $Pp^{\alpha-1}$ over Z_{p^α}.*

Proof. The matrix T and the unit sequence have the same period by Theorem 2. □

We note in passing that the word *unit* in the statement of Corollary 5 is crucial. In Jansson [9], it is incorrectly stated that the period of *any* vector $v \in V^k$ increases by at most p when M increases by p. The sequence (4.8) above provides a counter-example. Neither of the proofs provided by Jansson can be patched, as they rely on Zierler's results [21] for the *field* GF(p^α), rather than the *ring* Z_{p^α}.

The corollary above allows us to compute the period of the unit sequence over any power of the prime p. If the period over Z_p and the period over Z_{p^2} differ, then we know the period over Z_{p^α}. If the periods are the same, we need to check the period of the sequence mod $p^3, p^4, \ldots$ until the period changes.

Returning to the original generator, with a modulus of $M = 2^\alpha$, it is easy to see how sequences that are 'a factor of 2, or, rarely, by a factor of 4 or 8, or more' less than the unit sequence can arise. The period of the unit sequence will increase with the exponent α if the unit sequence (w) is longer over Z_4 than it is over Z_2. If each term of the original vector is divisible by 2, then the sequence will be at least twice as short as the unit sequence. If the original vector is divisible by 4, the sequence will be at least 4 times as short, and so on. Naturally, the odds that a vector contains a factor of 2^c decrease as c increases.

This is not the only cause of short periods. As our example (4.4) showed, if the characteristic polynomial factors, we may have short sequences, even if the original vector has no common factor. We discuss factorization in the next section.

6. Ring structure

We use Hall's [8] conventions in discussing the ring structure of V^k. If (r) and (s) are sequences of integers mod Z_M that satisfy (1.1), then we define addition, scalar multiplication, and the shift operator X by:

$$(r+s) = r_1+s_1, r_2+s_2, r_3+s_3, \ldots,$$

$$t(r) = tr_1, tr_2, tr_3, \ldots \quad (\text{where } t \in Z_M),$$

$$X(r) = r_2, r_3, r_4, \ldots.$$

The shift operator has the same effect on vectors in V^k as multiplication by the matrix T. Clearly sequences that satisfy (1.1) or elements of the vector space V^k are closed under all three operations. These operations allow us to identify the set of all sequences satisfying (1.1) with a polynomial ring of operators $R(x)$, and then to identify the ring $R(x)$ with the vector space V^k. We define an isomorphism $\imath : R(x) \to V^k$ by $\imath(1) = w = (0, 0, \dots, 1)$ and

$$\imath(a_n x^n + \cdots + a_1 x + a_0) = a_n X^n w + \cdots a_1 X w + a_0 w.$$

This identifies V^k with the polynomial ring $R(x) = F[x]/f(x)$, where $f(x)$ is the characteristic polynomial of our recurrence relation, and $F = Z_M$. Two polynomials $g(x)$ and $h(x)$ are equivalent in $R(x)$, written $g(x) \equiv h(x)$, if we may find polynomials $q(x)$ and $u(x)$ such that $g(x) = h(x) + f(x)q(x) + Mu(x)$.

Theorem 6. *The unit sequence generates all other sequences satisfying* (1.1), *using addition, scalar product, and the shift operator.*

Proof. (By example) Let $k = 3$ and $M = 3$, so that we have the relation $r_{n+3} \equiv r_{n+2} + r_n \pmod 3$. To produce the sequence generated by the initial vector $(1, 2, 1)$, we first obtain $r_1 = 1$. Take $\imath(x^2) = X^2 w = (1, 1, 1)$. We add $\imath(x) = Xw$ to get the desired r_2,

$$X^2 w + Xw = (1, 1, 1) + (0, 1, 1) = (1, 2, 2),$$

and add $2w$ to get $r_3 = 1$, giving

$$\imath(x^2 + x + 2) = (X^2 + X + 2)w = (1, 1, 1) + (0, 1, 1) + (0, 0, 2) \equiv (1, 2, 1).$$

Note that the sequence represented by the polynomial $(x^2 + x + 2)$ had a characteristic polynomial $(x + 1)$ and that $(x^2 + x + 2)(x + 1) = f(x)$. $\square$

An application of Theorem 6 gives another proof of Corollary 3, that the period of any sequence divides the period of the unit sequence (w).

If M is prime, then $F[x]$ is a field. If $f(x)$ is irreducible in $F[x]$, then $f(x)$ generates a prime ideal, $F[x]/f(x)$ is a field, and the orbits of our generator are well understood. Each (non-zero) orbit has the same size, and any non-zero initial vector will give maximal period. If the characteristic polynomial $f(x)$ is primitive over Z_M, there is a single (non-zero) orbit of size $M^k - 1$. A polynomial of degree k over Z_p is said to be primitive if all its roots have order p^k. Primitive polynomials are irreducible, but the reverse is not true. An example is $f(x) = x^9 - x^8 - 1 \pmod 2$. The polynomial is irreducible over Z_3, but is is not primitive, as the unit sequence $0, 0, 0, 0, 0, 0, 0, 0, 1 \dots$ has length 73. Rather than one large orbit of length $2^9 - 1 = 511$, we have 7 orbits of size 73.

If $f(x)$ factors over $F[x]$, then $R(x)$ is not a field and we may have orbits of different sizes, as we saw in our example. If M is not prime, then Z_M is not a field, and we may again have different periods, as examples of the form $p(v)$ demonstrate.

Consider the period of a (non-unit) sequence (r). Using the isomorphism above, we associate a sequence (r) with a polynomial $r(x) \in R(x)$. The sequence (r) represented by the polynomial $r(x)$ has period P if P is the least integer such that

$x^P r(x) \equiv r(x)$, or $(x^P - 1)r(x) = f(x)q(x) + Mu(x)$. If $f(x)$ factors as $f(x) = s(x)t(x)$, then the sequences that correspond to $s(x)t(x)$ and $t(x)$ will satisfy relations of lower degree, and may have smaller periods.

Theorem 7. *If the modulus is a prime p, and the sequence (r) has a period P' over Z_p which is shorter than the period P of the unit sequence (w), then $r(x)$ satisfies a relation of degree lower than the degree k of the characteristic polynomial $f(x)$.*

Proof. If (w) has period P, then P is the smallest integer such that $f(x) \mid (x^P - 1)$. Since $r(x)$ has period P', we may write $(x^{P'} - 1)r(x) = f(x)q(x) + pu(x)$. If $r(x)$ and $f(x)$ are relatively prime, then $r(x)$ must divide $q(x)$, by unique factorization over the field $Z_p[x]$. So $q(x) \equiv r(x)s(x)$, and $(x^{P'} - 1) \equiv f(x)s(x)$, contradicting the minimality of P. Thus $f(x)$ and $r(x)$ must share a common factor, say $d(x)$, and $f(x) \equiv d(x)e(x)$. This implies that $r(x)e(x) \equiv 0$. Thus $r(x)$ satisfies a relation of degree less than k.

Unique factorization over the field Z_p is crucial. The result is not true over Z_{p^α}, as the example $p(w)$ shows. The converse of this result is also not true: satisfying a relation of lower degree does not force a shorter period. In our example, the unit sequence (4.2) has the same period as the sequence (4.5), which satisfies a relation of lower degree.

If a sequence (v) satisfies a relation of degree less than k, then there is a linear dependence between the vectors $v, Tv, T^2v, T^3v, \dots$. In our examples (4.5) was restricted to a two-dimensional subspace, and (4.4) lay on a line.

In fact, the implication can be reversed: if j consecutive vectors $v_n, v_{n+1}, \dots, v_{n+j-1}$ lie in a subspace of dimension $j-1$, the remaining vectors v_i must also, by the linearity of T. This simple observation was first made by Kronecker, and gives us an elegant, if expensive, way to check for low order relations.

Given the sequence (r), define the function

$$N(r) \overset{\text{def}}{=} \det \begin{vmatrix} r_1 & r_2 & \cdots & r_k \\ r_2 & r_3 & \cdots & r_{k+1} \\ \cdot & \cdot & \cdots & \cdot \\ r_k & r_{k+1} & \cdots & r_{2k-1} \end{vmatrix} . \tag{6.1}$$

Clearly, if (r) is divisible by p, then $N(r) \equiv 0 \pmod p$. Assume that (r) is not divisible by p: then the sequence (r) satisfies a relation of degree less than k if and only if $N(r) \equiv 0$.

This completes our characterization of sequences with short periods. In the next section, we show how to avoid such sequences.

7. How to maximize the period

When creating a Fibonacci random number generator, we must select the modulus, the degree, and the initial vector. We consider each in turn.

7.1. Choice of modulus M

Random number generators often use a modulus $M = 2^a$, where a is the length of a machine register. This gives a large modulus, to allow long periods, and allows a simple way to reduce the random number $r_i \pmod M$. If we have r_i expressed as a result plus overflow, we produce $r_i \pmod M$ by discarding the overflow. As Knuth [12] points out so forcefully, we should not turn to a modulus of 2^a out of sloth, when a modulus of $2^a \pm 1$ is as convenient to work with, and much more 'random' in the least significant bits. As an example, we use a modulus of $2^a - 1$. Since the Fibonacci generator adds integers, the sum can always be held in one register plus an overflow bit. To produce r_i, we simple add the overflow bit back into the register.

Corollary 5 shows that the choice of a power of a prime for a modulus can lead to smaller periods. A new prime p can lengthen the period by a factor of $p^k - 1$, while each additional power of the prime p contributes at most one factor of p to lengthen the period. Consider two examples: first let $M = 2^{16}$ and then let $M = 2^{16} - 1$. If $k = 3$, then the period, mod 2 of (w) is 7, and the period of (w) mod 4 is larger, so the period of (w) over Z_M is $7 \times 2^{15} = 229{,}376$ by Corollary 5. However, $2^{16} - 1 = 3 \times 5 \times 17 \times 257$. If we compute the length of the unit sequence mod each prime, we find that the period of the sequence (w) mod $2^{16} - 1$ is $\operatorname{lcm}(8, 31, 288, 66307) = 591{,}988{,}896$, three orders of magnitude larger. As we increase the degree k, the calculations become harder to make, but the magnitude of the difference increases.

If a power of two must be used, a simple modification will remove one cause of short periods. We pick the first $k - 1$ integers of v using a Lehmer RNG, and then pick an odd integer for the last place. This assures us that $p \nmid v$, but does not prevent v from satisfing a relation of low order. However, if the characteristic polynomial is irreducible, this simple expedient will guarentee a maximal period.

7.2. Choice of degree k

Green, Smith and Klem [7] suggest that k be at least 16, but they are concerned over the amount of memory that this will use. In this time of cheap memory, the user will wish to pick $k \geq 16$.

If it is known that the pseudo-random numbers are to be used modulo a particular prime factor of M, it is wise to be sure that the sequence behaves well with respect to this prime, even at the expense of other primes. We may wish to pick k so that the characteristic polynomial, $f(x)$, is irreducible, or primitive, mod several of the primes in the factorization of M.

Another approach is to follow numerous suggestions, and use a two-term additive congruential generator that has a well behaved characteristic polynomial.

7.3. Choice of initial vector v

Assume that we have selected M and k. To pick an initial vector v with the maximal period, it is sufficient, though not necessary, that $p_i \nmid v$ for each prime factor p_i of M and that (v) satisfies no relation of degree less than k.

One strategy would be to pick a vector v using the system clock and another pseudo-random generator, and calculate the $k \times k$ determinant $N(r)$ to check both conditions. Since k will be large, this is an unattractive technique for a generator designed to eleminate multiplications. We turn to an alternate method of picking 'random' vectors, each of which will satisfy the conditions above.

We assume that there are t distinct prime factors, and that $k \geq t$. The algorithm can be modified quite easily to deal with the general situation. We define a set S of k vectors in the unit sequence, and pick a distinct vector v_i based on some element of S for each prime factor p_i of M. We then build a vector $v \equiv v_i \pmod{p_i^{a_i}}$. This can be done in $O(k+t)$ time and space, using Garner's constructive proof of the Chinese Remainder Theorem (see Knuth [12]). We compute, when designing the generator, constants U_i so that

$$U_i \equiv \delta_{ij} \pmod{p_j^{a_j}},$$

or $U_i \equiv 1 \pmod{p_i^{a_i}}$, and $U_i \equiv 0 \pmod{p_j^{a_j}}$ if $j \neq i$. Then we let

$$v = \sum U_i v_i \pmod{M}.$$

In the algorithm below, we let $S = \{(1,0,\ldots,0),(0,1,\ldots,0),\ldots,(0,0,\ldots,1)\}$, k vectors in the unit sequence[2]. For each prime p_i, we pick an element $s_i \in S$, a random unit $u_i \in Z_{p_i}$, and let $v_i = u_i s_i$. It is easy to see that the sequence generated by v_i has maximal length, mod $p_i^{a_i}$. This allows us

$$\frac{k!}{(k-t)!} \prod_{1 \leq i \leq t} (p_i - 1)$$

distinct initial vectors, ample for most applications. In our case, with $k = 16$, and $M = 2^{16} - 1$, this allows 1.4×10^9 different initial vectors.

Our choice yields a vector with $k-t$ 0's, and thus the first crop of numbers will contain several duplicate numbers. The ability to produce duplicates is one of the strengths of this generator, but it is unpleasant to encounter them so soon. To avoid this, we run the generator k times to prime the pump.

If M contains a square p_i^2, we can modify v by adding a vector of the form $tp_i U_i s_i$ where $t \in Z_{p_i}$, and $s_i \in S$. This does not alter the period, but increases the supply of initial vectors.

It may be preferable to use a vector v_j that satisfies a relation of lower degree over Z_{p_j}, if the characteristic polynomial has a primitive factor of large degree. Primitive polynomials have a better distribution of digits over Z_p. (Compare the unit sequence (w) to the sequence (z) in Table 1: sequence (z) satisfies the primitive polynomial x^2+x+2.) We simply define another set S' to be used to pick v_j, con-

[2] This is the only place that our algorithm uses properties unique to the Fibonacci generator. For general additive generators, we could use the first few terms of the appropriate unit sequence.

taining vectors s that satisfy the primitive polynomial.

We present a program fragment in Pascal that implements our random number generator, with a choice of $M = 2^a - 1$. We assume that a file *params* exists which contains the prime factors of M, as well as the constants U_i. We have written the generator in a high-level language for clarity: in practice, we would code this in machine language or micro code. Since standard Pascal does not provide a random number generator, we assume an implementation of some random number generator: in this example, we have used a Lehmer RNG. Since we cannot trap the overflow condition in Pascal, we assume further a function *mult* that multiplies integers in Pascal. This is needed by the function *Lehmer*, and is also used by *random-init*.

```
const
    modulus = 2^a;
    degree = k;
type
    index = 0··degree;
    vector = array [0··degree] of integer;
var
    seed: integer;          (*Global variable for Lehmer RNG*)
    v;                      (*Last k random numbers*)
    p;                      (*Prime factors of modulus*)
    U: vector;              (*U[i] ≡ δ_ij (mod p_j)*)
    rand-index: index;      (*Index into vector v*)
    params: text;           (*File with parameters*)

function random: integer;
        (*Compute the next random number*)
    var i: integer;
    begin
        i := rand-index;
        rand-index := (rand-index + 1) mod k;
        v[rand-index] := (v[rand-index] + v[i]) mod modulus;
        random := v[rand-index];
    end;

function mult (p,q: integer): integer;
        (*Multiply without overflow*)

function Lehmer (max: integer): integer;
        (*Use a Lehmer random number generator to alter global seed*)
        (*Return a number in the range 0··max-1*)
    begin
        seed := (mult (seed, multiplier) + increment) mod modulus;
        Lehmer := trunc ((seed/modulus) * max);
    end;

procedure random-init (var $v$: vector; var $j$: integer);
            (*Set up the vector $v$, as proposed in [7]*)
    var
        $S$: set of 0··degree
        pick;                   (*index of non-zero entry of $v_i$*)
        $i$;                      (*Loop var*)
        num-factors;            (*Number of prime factors of $m$*)
        temp: integer;          (*Temp variable*)
    begin
        readln (params, num-factors);     (*Read the number of prime
                                            factors*)
        for $i$ := 1 to num-factors do
            readln (params, $p[i]$, $U[i]$);  (*Read prime factors and
                                            constants*)
        seed := clock mod modulus;        (*Set seed to the system clock*)
        $S$ := [0··degree − 1];             (*Set of vectors*)
        for $i$ := to degree do             (*Assume degree > num-factors*)
            $v[i]$ = 0;
        for $i$ := 1 to num-factors do
            begin    (*compute $v = v_i$*)
                pick := Lehmer (degree);  (*Return number in range
                                            0··degree − 1*)
                while not (pick in $S$) do    (*Find new $s_i$*)
                    pick := (pick + 1) mod degree;
                $S$ := $S$ − [pick];
                                    (*$v_i = s_i \times U_i \times$ unit*)
                $v$[pick] := mult($U[i]$, Lehmer($p[i]$ − 1) + 1) mod modulus;
            end;
        for $i$ := 1 to degree do
            temp := random;     (*Spin the wheel $k$ times*)
    end;    (*random-init*)
```

8. Empirical tests for randomness

Green et al. [7] apply the frequency test, the 'poker' test, tests for serial correlations, and the run test to the Fibonacci RNG. To give the reader a feeling for the strengths and weaknesses of the generator, we will present a comparison of the

3-space uniformity of the Lehmer RNG and the Fibonacci RNG, and look at the performance of the Fibonacci RNG on the run test.

8.1. Distribution tests

To demonstrate the n-space non-uniformity of the Lehmer generator, we used the Lehmer and the Fibonacci RNGs to generate vectors in 3-space, and then evaluated the distribution using a standard χ^2 (chi-square) test. In each test, we generated sets of 30,000 numbers. We scaled the numbers to lie within the range 0··9 by division, to use the most significant bits of the integers generated. We then took 10,000 triples of these decimal digits, and tabulated the frequency of each triple. We tested the resulting distribution with a χ^2 test with 999 degrees of freedom, and compare with the expected value of 999. This test does not show the Lehmer generator to any great advantage.

We ran 25 such test of the Fibonacci RNG: none were significant at the 5% level: the extreme values were 912.2 and 1057.0. We also ran 25 trials of the Lehmer RNG: the smallest value was 71,792.4, two orders of magnitude larger than the expected value. We can conclude that the Lehmer RNG is not n-space uniform, and that the Fibonacci RNG may be, for $n < k$.

8.2. Run tests

Of all the standard empirical tests, the run test gives the Fibonacci generator the greatest difficulty. The run test counts the frequency of 'runs' of different lengths. A run up (down) is a sequence of ascending (descending) numbers. The generator of degree 2 produces too many long runs up, and too few long runs down. We have applied the run test to our generator, and we present some typical test results in the following figures.

Our tests consisted of the following steps: a degree was chosen, the RNG initialized, and 10,000 random numbers were generated. In counting runs, we discard the number that ends a run, to avoid interdependence between the number of runs of different lengths. We kept track of the number of runs up (down) of length 1 through 5, and lumped the number of runs of length 6 or greater into one count. These counts were tabulated, and compared to the expected distribution using a χ^2 test with five degrees of freedom. Each set of 10,000 numbers gives us two data points in the later figures: the χ^2 values for runs up and runs down.

We list the degree of the generator on the x-axis, and plot the results of the χ^2 test on the y-axis. First we display (see Fig. 1) the results of the ordinary Fibonacci generator with small degrees and no decimation. The results are two orders of magnitude higher than the expected value.

The results improve as we increase the degree. In order to view the distribution better, we next present (see Fig. 2) the results of 25 tests up and down for even degree from 10 to 24. (Odd degrees produced similar results.) These values are plotted as points in the figures, with the points representing the ascending runs position-

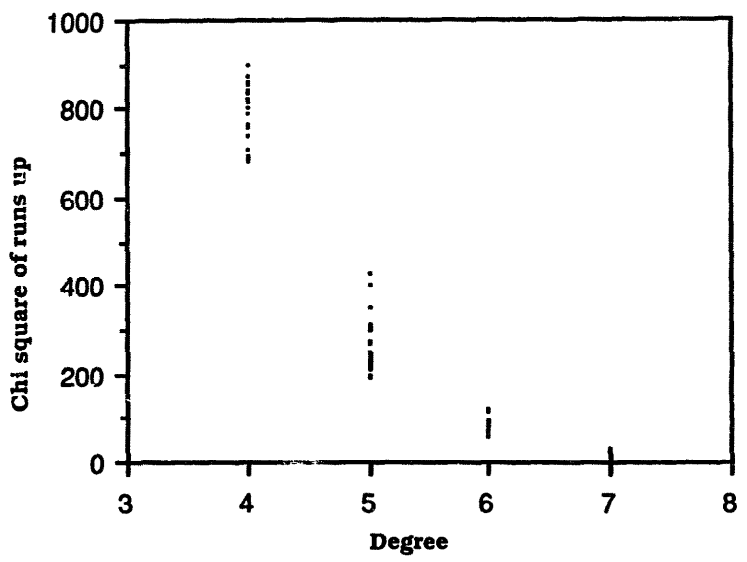

Fig. 1. Fibonacci generator without decimation.

ed the right of the points representing descending runs. A standard table of the χ^2 distribution with five degrees of freedom was used to mark the y-axis in the figures.

In Fig. 3, we plot the results of the generator modified to include decimation. Every other number is discarded. As may be seen, there is an improvement when the degree is large, but the most dramatic improvement is for small degrees.

9. Conclusion

The generalized Fibonacci random number generator is a fast random number generator which may be used to produce sequences with a very large period. The generator may be n-space uniform, if n is less than the degree k. We have presented an algorithm that assures a long period for the generator, while allowing many dif-

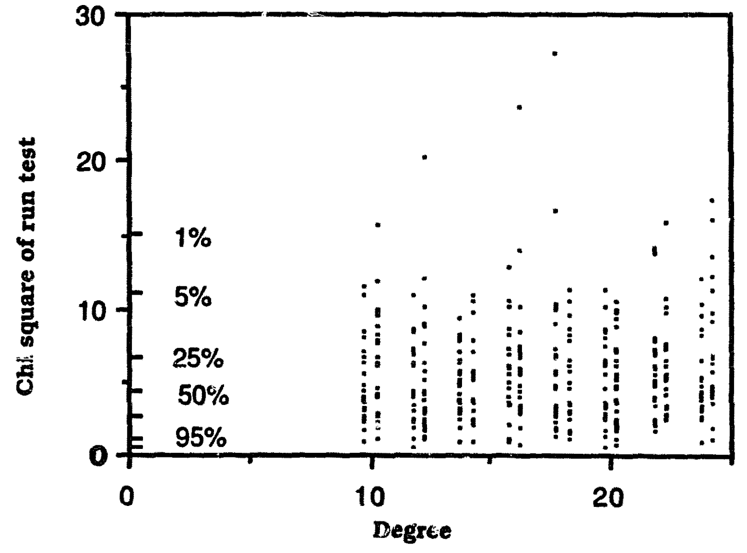

Fig. 2. Fibonacci generator without decimation.

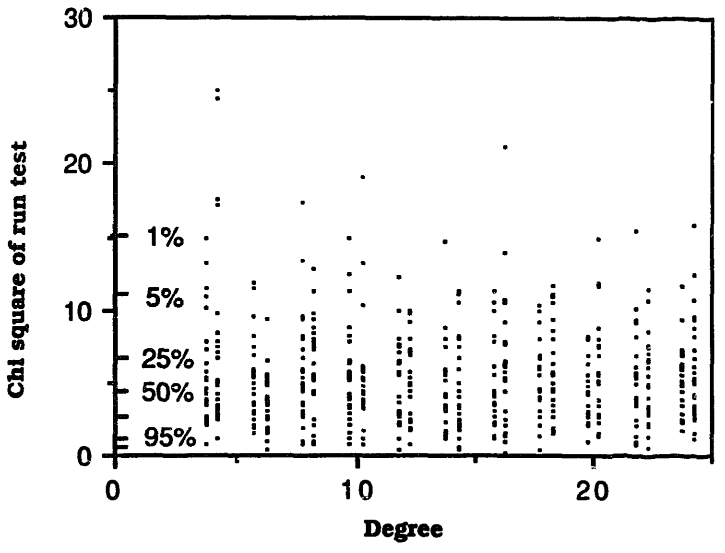

Fig. 3. Fibonacci generator with decimation.

ferent initial values. A long period does not guarantee randomness, but we have shown that a modest increase in the degree of the generator, coupled with decimation, provides greatly improved performance on the run test.

We need not remind the reader that caution is important in the use of any pseudo-random number generator. Empirical testing can discover non-randomness, but cannot prove randomness. Before recommending the adoption of the Fibonacci RNG, we will need to better understand and certify the behavior of tuples $(r_i, r_{i+1}, \ldots r_{i+j-1}) \in V^j$.

In closing, we would like to thank the valiant efforts of the referee, whose comments and questions greatly improved the presentation.

References

[1] P. Bratley, B.L. Fox and L.E. Schrage, A Guide to Simulation (Springer, New York, 1983).

[2] H.S. Bright and R.L. Enison, Quasi-random number sequences, Computing Surveys 11 (4) (1979) 357–370.

[3] J.M. Carrol, and P.M. McLelland, Fast infinite-key privacy transformation for resource-sharing systems, Proc. 1970 AFIPS Fall Jt. Computer Conf., Vol. 37 (AFIPS Press, Arlington, VA) 223–230.

[4] R.R. Coveyou and R.D. MacPherson, Fourier analysis of uniform random number generators, J. ACM 14 (1) (Jan. 1967) 100–119.

[5] U. Dieter, Linear congruential method, in: Applications of Number Theory to Numerical Analysis (Academic Press, New York, 1972).

[6] H.J.A. Duparc, C.G. Lekkerkerker and W. Peremans, Reduced sequences of integers and pseudo-random numbers, Math. Centrum, Amsterdam, Report ZW1953–002, 1953.

[7] B.F. Green, J.E.K. Smith and L. Kiem, Empirical tests of an additive random number generator, J. ACM 6 (1959) 527–537.

[8] M. Hall, An isomorphism between linear recurring sequences and algebraic rings, AMS Trans. 44 (1938) 196–218.
[9] B. Jansson, Random Number Generators (Victor Pettersons Bokindustri Aktiebolag, Stockholm, 1966).
[10] M.G. Kendall and B. Babington Smith, Randomness and random sampling numbers, J.R.S.S. (A) 101 (1938) 167–172.
[11] D.E. Knuth, Construction of a random sequence, BIT 5 (1965) 246–250.
[12] D.E. Knuth, Art of Computer Programming – Vol. 2: Seminumerical Algorithms (Addison-Wesley, Reading, MA, 1969).
[13] D.H. Lehmer, Mathematical methods in large-scale computing units, Proc. 2nd Symposium on Large Scale Digital Calculating Machinery (Harvard Univ. Press, 1949) 141–146, MR 13#146.
[14] T.G. Lewis and W.H. Payne, Generalized feedback shift register pseudorandom number algorithm, J. ACM 20 (3) (1973) 456–468.
[15] G. Marsaglia, Random numbers fall mainly on the planes, Proc. Nat. Acad. Sci. 61 (1) (Sept. 1968) 25–28.
[16] E.R. Sowey, A chronoglogical and classified bibliograpy in random number generation and testing, Int. Stat. Rev. 40 (3) (1972) 355–371.
[17] R. Sedgewick, Algorithms (Addison-Wesley, Reading, MA, 1983).
[18] O. Tausky and J. Todd, Generation of random numbers on computers, in: H.A. Meyer, ed., Symposium on Monte Carlo Methods (Wiley, New York, 1956) 323–337.
[19] R.C. Tausworthe, Random numbers generated by linear recurrence modulo two, Math. Comput. 19 (1965) 201–209.
[20] L.H.C. Tippett, Random sampling numbers, 1927. Reprinted in 1952, Tracts for Computers, no. 15, (Cambridge Univ. Press, Cambridge).
[21] N. Zierler, Linear recurring sequences, SIAM J. 7 (1959) 31–48.