JOURNAL OF COMPUTATIONAL PHYSICS 83, 16–31 (1989)

ACORN—A New Method for Generating Sequences of Uniformly Distributed Pseudo-random Numbers

R. S. WIKRAMARATNA

Petroleum Reservoir Technology Division, Atomic Energy Establishment, Winfrith, Dorchester, Dorset DT2 8DH, United Kingdom

Received January 8, 1988; revised September 2, 1988

A new family of pseudo-random number generators, the ACORN (*a*dditive *c*ongruential *r*andom *n*umber) generators, is proposed. The resulting numbers are distributed uniformly in the interval [0, 1). The ACORN generators are defined recursively, and the $(k+1)$th order generator is easily derived from the kth order generator. Some theorems concerning the period length are presented and compared with existing results for linear congruential generators. A range of statistical tests are applied to the ACORN generators, and their performance is compared with that of the linear congruential generators and the Chebyshev generators. The tests show the ACORN generators to be statistically superior to the Chebyshev generators, while being statistically similar to the linear congruential generators. However, the ACORN generators execute faster than linear congruential generators for the same statistical faithfulness. The main advantages of the ACORN generator are speed of execution, long period length, and simplicity of coding.

1. INTRODUCTION

In this paper we consider the problem of generating a sequence of real numbers, distributed uniformly between zero and one. The primary motivation in this work was to find a method which would produce statistically uniform and uncorrelated sequences of numbers and which could easily be implemented on any computer ranging from micro to mainframe.

Knuth [1] considers at some length the class of linear congruential generators, which can be defined by

$$U_n = (aU_{n-1} + c)_{\mathrm{mod}\,1}, \qquad n > 1 \tag{1}$$

(where $(X)_{\mathrm{mod}\,1}$ means the fractional part of X) with suitably chosen values for the parameters U_0 (the initial value; $0 < U_0 < 1$), a (an integer valued multiplier; $a > 0$) and c (the increment; $0 \leqslant c < 1$). The special case $c = 0$, which was proposed originally by Lehmer [2], is generally known as the multiplicative congruential method. The more general case of non-zero c which is due independantly to Thompson [3] and Rotenberg [4] is often called the mixed congruential method. Congruential generators are among the most popular methods used today [1]. An

0021-9991/89 $3.00

example of such a generator is the subroutine G05CAF from the NAG subroutine library [5], which will be used in this paper to provide a benchmark for testing the new generator which is being proposed.

A second comparison is provided by the Chebyshev mixing method proposed by Erber, Everett, and Johnson [6]. In this generator, the random numbers U_n are determined from

$$U_n = (1/\pi)\cos^{-1}(Z_n/2), \tag{2}$$

where $Z_n = Z_{n-1}^2/2$ and where the initial value Z_0 is chosen to lie in the range $-2 < Z_0 < 2$. Superficially this generator may appear to satisfy the major objectives as described above; as will be shown, however, it possesses undesirable qualities which make it unsuitable as a source of random numbers.

2. The ACORN Generator

Knuth [1] states that we can immediately reject the cases of linear congruential generators with $a = 0$ or $a = 1$ as leading to sequences which are obviously non-random and goes on to consider only those generators for which $a \geqslant 2$. We will demonstrate below that there is merit in considering the two cases $a = 0$ and $a = 1$ as the basis for a more general recursive generator which performs as well as the mixed congruential generator while being considerably simpler to implement and faster to execute.

We define the kth order ACORN (*a*dditive *co*ngruential *r*andom *n*umber) generator X_j^k recursively from a seed X_0^0 $(0 < X_0^0 < 1)$ and a set of k initial values X_0^m, $m = 1, ..., k$ each satisfying $0 \leqslant X_0^m < 1$ by

$$X_n^0 = X_{n-1}^0, \qquad n \geqslant 1 \tag{3}$$

$$X_n^m = (X_n^{m-1} + X_{n-1}^m)_{\mathrm{mod}\,1}, \qquad n \geqslant 1,\ m = 1, ..., k. \tag{4}$$

Other additive generators have been defined previously (see, for example, Tausworthe [7] or the survey by Halton [8]). In general, such generators calculate each number as some additive combination of the previous n numbers in the sequence; the ACORN generator is unique in that each number in the kth order sequence is derived by combining the previous number in the sequence with a corresponding number from the $(k-1)$th order sequence.

The way in which these equations are applied in calculating the random numbers is illustrated in Fig. 1. The arrows indicate the numbers which are combined in calculating each particular value X_n^m; thus, for example, X_3^2 is derived from X_3^1 and X_2^2 according to Eq. (4) and so

$$X_3^2 = (X_3^1 + X_2^2)_{\mathrm{mod}\,1}. \tag{5}$$

Equation (3) simply states that all the numbers X_n^0 in the top row of Fig. 1 are set

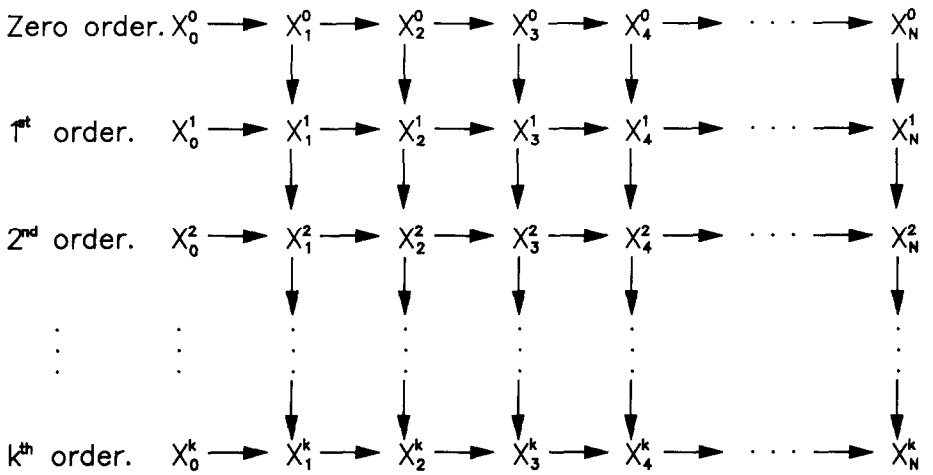

FIG. 1. Method of calculation of ACORN numbers.

equal to the seed X_0^0. It should be noted that the performance of the ACORN generator is closely related to an appropriate choice of seed. Tests have shown that the best performance is obtained by choosing X_0^0 to be a number which has no obvious regularities or repetitions of digits; a convenient method in practice was to set X_0^0 equal to the machine representation of an irrational number, for example, a suitable multiple of the square root of 2. The initial values of X_0^m in each subsequent row $m \geqslant 1$ can be assigned any value in the range $0 \leqslant X_0^m < 1$; for the particular example considered in the results section of this paper all the initial values were set equal to zero, but similar results are obtained with other choices. It should be noted also that this choice of initial values X_0^m has no effect on the execution times for the ACORN generator.

Clearly, the zero-order ACORN generator (3) is a special case of a mixed congruential generator with $a = 0$, while the first-order generator (Eq. (4) with $m = 1$) is a mixed congruential generator with $a = 1$. As such, these low-order ACORN generators are of little direct use as sources of random numbers. However, their usefulness stems from the recursive definition of the general kth-order ACORN generators.

The ACORN generator can conveniently be implemented in one of two ways.

(i) In testing the generator for randomness, we wish to generate a fixed number N of pseudo-random numbers and apply statistical tests to these numbers. To minimise the storage requirements and to allow the tests to be applied efficiently to successively higher orders of generators we calculate the N values X_n^m, $n = 1, ..., N$ and apply the statistical tests to these numbers; we then use (4) to calculate X_n^{m+1}, $n = 1, ..., N$ and apply the tests again. This process can be repeated up to any desired order of the generator and corresponds to calculating the array in Fig. 1, one row at a time.

```
      DOUBLE PRECISION FUNCTION ACORN(XDUMMY)
C
C        FORTRAN IMPLEMENTATION OF ACORN RANDOM NUMBER GENERATOR
C        OF ORDER LESS THAN OR EQUAL TO 12 (HIGHER ORDERS CAN BE
C        OBTAINED BY INCREASING THE PARAMETER VALUE MAXORD).
C
C        R.S.WIKRAMARATNA
C        AEE WINFRITH, DORCHESTER, DORSET DT2 8DH, UNITED KINGDOM.
C
C        THE VARIABLE XDUMMY IS A DUMMY VARIABLE. THE COMMON BLOCK
C        ACO IS USED TO TRANSFER DATA INTO THE FUNCTION.
C
C        BEFORE THE FIRST CALL TO ACORN THE COMMON BLOCK ACO MUST
C        BE INITIALISED BY THE USER, AS FOLLOWS. THE VALUES OF
C        VARIABLES IN THE COMMON BLOCK MUST NOT SUBSEQUENTLY BE
C        CHANGED BY THE USER.
C
C          KORDER  - ORDER OF GENERATOR REQUIRED ( MUST BE =< MAXORD )
C
C          XV(1)   - SEED FOR RANDOM NUMBER GENERATOR
C                    REQUIRE    0. < XV(1) < 1.
C                    XV(1) SHOULD BE CHOSEN TO APPROXIMATE AN
C                    IRRATIONAL NUMBER, EG IF 0. < EPS < 1., THEN SET
C                       XV(1)=EPS*DSQRT(2.0D0)/1.42
C                    THE EXAMPLE FOR THE STATISTICAL TESTS USED THIS
C                    FORM FOR THE SEED, WITH EPS=0.1
C
C          (XV(I+1),I=1,KORDER)
C                  - KORDER INITIAL VALUES FOR GENERATOR
C                    REQUIRE    0. =< XV(I+1) < 1.
C                    THE EXAMPLE FOR THE STATISTICAL TESTS USED
C                    ZERO FOR ALL THE INITIAL VALUES.
C
C        AFTER INITIALISATION, EACH CALL TO ACORN GENERATES A SINGLE
C        RANDOM NUMBER BETWEEN 0 AND 1.
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (MAXORD=12,MAXOP1=MAXORD+1)
      COMMON /ACO/KORDER,XV(MAXOP1)
      DO 7 I=1,KORDER
        XV(I+1)=(XV(I+1)+XV(I))-INT(XV(I+1)+XV(I))
    7 CONTINUE
      ACORN=XV(KORDER+1)
      RETURN
      END
```

FIG. 2. Fortran implementation of the ACORN generator.

(ii) In a practical application of the generator, we wish to compute the kth order values, for some fixed value k. We do not know a priori how many such values will be required, and we want to avoid storing large arrays of random numbers. Starting with the seed X_0^0 and the k initial values X_0^m, $m = 1, ..., k$ we determine X_1^m, $m = 0, ..., k$ from (3) and (4). The first random number is given by X_1^k and the values of X_1^m, $m = 0, ..., k$ are stored until the next random number is required. The process is repeated with each successive call to the generator, the nth call updating the values of X_{n-1}^m, $m = 0, ..., k$ to give values of X_n^m, $m = 0, ..., k$ and returning the value X_n^k as the nth random number. This process corresponds to calculating the array in Fig. 1 one column at a time; note that it is only necessary to store the $(k+1)$ values in the most recently computed column. Figure 2 is an example of a Fortran 77 implementation of the ACORN random number generator.

3. Comparison with Existing Generators

In interpreting the results of the statistical tests, it is instructive to consider the way in which the nth random number in each sequence is derived from its predecessor.

The class of linear congruential generators has been defined in (1). The relationship between U_n and U_{n-1} is illustrated in Fig. 3a, for values of $a = 5$ and $c = 0.1$. Similar plots could be drawn for other values of a and c. In a scatter plot of pairs of points (U_{2k}, U_{2k-1}), all the points would lie on the lines shown. Such a plot is closely linked to the serial test for successive pairs discussed below; for a generator to pass the serial test, the points (U_{2k}, U_{2k-1}) should be equidistributed in the rectangular region $[0, 1) \times [0, 1)$. Clearly a linear congruential generator with a small value of a would have no chance of passing the serial test, whereas with larger values of a it is more likely to pass.

The Chebyshev generator is defined by (2). We can rewrite this as

$$U_n = (1/\pi)\cos^{-1}[(z_{n-1}^2 - 2)/2] \tag{6}$$

and hence

$$\begin{aligned}\cos(\pi U_n) &= 2[(z_{n-1}/2)^2 - 1] \\ &= 2[\cos^2(\pi U_{n-1}) - 1] \\ &= \cos(2\pi U_{n-1}).\end{aligned} \tag{7}$$

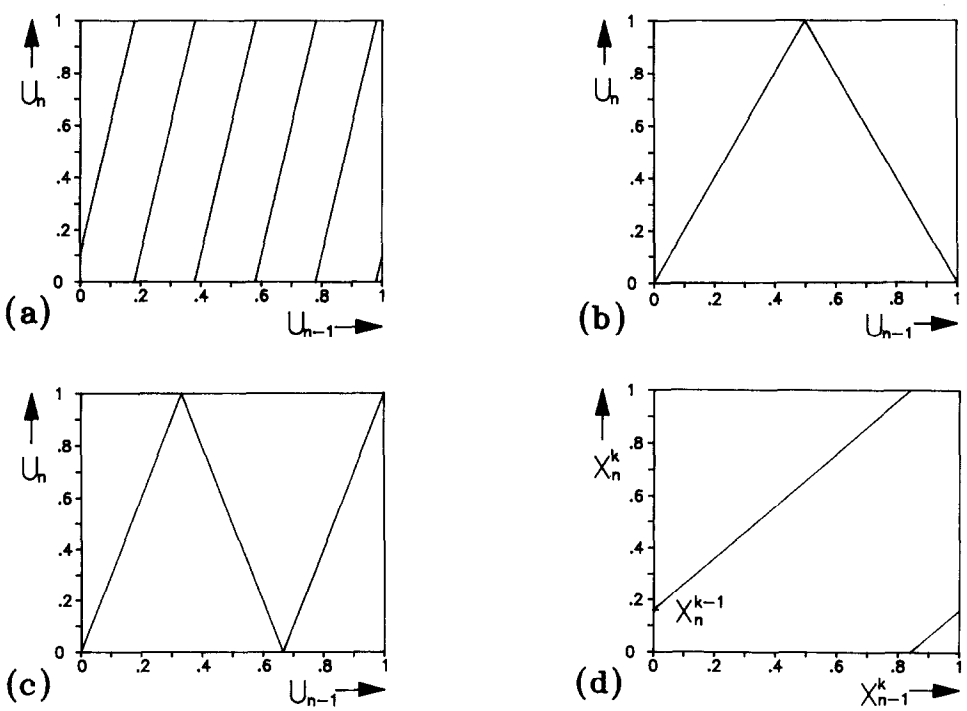

FIG. 3. Relationship between the $(n-1)$th and nth random numbers generated for different generators: (a) mixed congruential generator ($a = 5$, $c = 0.1$); (b) standard Chebyshev generator, based on expansion for $\cos(2z)$; (c) Chebyshev generator, based on expansion for $\cos(3z)$; (d) ACORN generator, order k.

Thus the generator can be re-written in the form

$$\begin{aligned} U_n &= 2U_{n-1} \qquad U_{n-1} < 0.5 \\ U_n &= 2 - 2U_{n-1} \qquad U_{n-1} \geqslant 0.5 \end{aligned} \tag{8}$$

and this relationship between U_n and U_{n-1} is shown in Fig. 3b.

If the generator is implemented in this form on a finite precision binary computer, then it will not work at all. Suppose that U_n has k significant binary digits; then each successive number $U_{n+1}, U_{n+2}, \ldots$ would have one fewer non-zero digit, until the number U_{n+k} would have no significant digits and the sequence would collapse to zero. This property of the Chebyshev generators has been previously noted by Erber *et al.* [9]. The reason that the generator works at all is due entirely to the rounding error which is introduced at each step in calculating the inverse cosine, which prevents the sequence from collapsing to zero. However, it is clear from Fig. 3b that this particular generator will certainly not pass the serial test for successive pairs (since every pair of points will fall on the line shown, and the distribution of points on the plane will clearly be non-uniform. We could define more complicated generators based on the same ideas as the Chebyshev generator, making use of expressions for $\cos az$ in powers of $\cos z$ and $\sin z$. For example, for $a = 3$,

$$\cos 3z = 4\cos^3 z - 3\cos z \tag{9}$$

and we could define

$$U_n = (1/\pi)\cos^{-1}(z_n), \tag{10}$$

where $z_n = 4z_{n-1}^3 - 3z_{n-1}$, with an initial value in the range $-1 < z_0 < 1$. For this particular case, the relationship between U_n and U_{n-1} is shown in Fig. 3c. Clearly such a generator will still perform very badly on the serial test, unless the value of a were very large. Thus any improvements along these lines would be at the expense of an enormous increase in computational effort.

For the kth order ACORN generator, the situation is somewhat different. The relationship between X_n^k and X_{n-1}^k is very simple, as shown in Fig. 3*d*. However, the intercept on the y-axis has a value which varies with n, being simply the value X_n^{k-1} of the corresponding number from the $(k-1)$th order generator. The fact that the X_n^{k-1} are themselves equidistributed, combined with the fact that the method increases the randomness of the sequences with increasing order k is a key to the success of the ACORN generator.

4. Period Length for ACORN Generator

Equations (3) and (4) can be re-written in the form

$$Y_n^0 = Y_{n-1}^0, \qquad n \geqslant 1 \tag{11}$$

$$Y_n^m = (Y_n^{m-1} + Y_{n-1}^m)_{\mathrm{mod}\, M} \qquad n \geqslant 1, m = 1, \ldots, k, \tag{12}$$

where the Y_n^m take integer values between zero and $(M-1)$, and by $(Y)_{\mathrm{mod}\,M}$ we mean the remainder on dividing Y by M. The Y_n^m can be related to the X_n^m in Eqs. (3) and (4) by defining

$$X_n^m = Y_n^m/M. \tag{13}$$

This formulation has the advantage that it is more amenable to theoretical analysis, since it only requires integer arithmetic to calculate the Y_n^m and the calculations can be carried out exactly without any rounding errors. In practice, however, it is more convenient to use the formulation of Eqs. (3) and (4) to implement the generator, using real arithmetic. The appropriate value of M can be related to the precision of the real arithmetic on any particular machine by setting $M = 1/\varepsilon$, where ε is the smallest real number such that the machine representations of 1 and $(1+\varepsilon)$ are different. For a binary computer, $M = 2^{\alpha-1}$, where α is the number of significant digits in the internal representation of a real number (typically, $\alpha = 23$ in single precision and $\alpha = 52$ in double precision Fortran on an IBM PC-XT).

In the following sections we establish bounds on the period lengths of the first- and second-order ACORN generators and derive conditions under which the bounds are attained. We further show that the period length for the $(k+1)$th-order generator must be an integer multiple of the period length for the kth-order generator, and that the period lengths are equal only in special circumstances. Finally we make a comparison with existing theoretical results concerning the linear congruential generator and show that the period lengths in this case are no better than for the first-order ACORN generator, for a given precision of arithmetic.

One approach to increasing the period length is to increase the precision of the arithmetic. This is particularly simple to do for the ACORN generators: since the only arithmetic operation required is addition modulo 1, it would be very straightforward to code an implementation of the ACORN generator which uses any specified precision of arithmetic (albeit at the expense of an increase in computational effort).

4.1. *Period Lengths for the First-Order ACORN Generator*

THEOREM 1. *The maximum possible period length for the first-order ACORN generator, defined by* (12) *with* $m = 1$, *is equal to* M. *For any given values of* M *and* Y_0^0, *the period length actually obtained is equal to* M/H, *where* H *is the highest common factor of* M *and* Y_0^0; *the maximal period length is attained if and only if* M *and* Y_0^0 *are relatively prime.*

Proof. Y_n^1 can take only the M values $0, 1, \ldots, (M-1)$, so there must be at least two of the $(M+1)$ values $Y_0^1, Y_1^1, \ldots, Y_M^1$ which are equal. Suppose that $Y_i^1 = Y_j^1$, where $0 \leqslant i < j \leqslant M$ and suppose further than $Y_i^1 \neq Y_k^1$ for any k s.t. $i < k < j$. Then from Eqs. (11) and (12),

$$\begin{aligned} Y_{i+1}^1 &= (Y_i^1 + Y_{i+1}^0)_{\mathrm{mod}\,M} \\ &= (Y_j^1 + Y_{j+1}^0)_{\mathrm{mod}\,M} = Y_{j+1}^1. \end{aligned} \tag{14}$$

Hence, by induction, $Y^1_{(n+j-i)} = Y^1_n$ for all $n \geqslant 0$, and so the sequence Y^1_n is periodic with period $(j-i) \leqslant M$.

Suppose that H is the highest common factor of Y^0_0 and M. Then defining $Z^0_0 = Y^0_0/H$, $\mu = M/H$, and $Z^1_n = ((Y^1_n - Y^1_0)/H)_{\mathrm{mod}\,\mu}$, we obtain

$$\begin{aligned} Z^1_n &= (Z^1_{n-1} + Z^0_0)_{\mathrm{mod}\,\mu} \\ &= (nZ^0_0)_{\mathrm{mod}\,\mu}, \end{aligned} \tag{15}$$

since by definition $Z^1_0 = 0$.

Then for any n, $Z^1_{n+m} = (nZ^0_0 + mZ^0_0)_{\mathrm{mod}\,\mu}$ and, since Z^0_0 and μ are relatively prime, it follows that $Z^1_{n+m} = Z^1_n$ if and only if $m = k\mu$ for some integer k. Thus the period length for Z^1_n is equal to $\mu = M/H$; by definition of the Z^1_n, the sequence Y^1_n must also have a period of length μ.

If Y^0_0 and M are relatively prime then $H = 1$ and so the sequence Y^1_n will have the maximal period length of M.

COROLLARY TO THEOREM 1. *If $M = 2^k$, then the first-order ACORN generator will have a period length M if and only if Y^0_0 is odd.*

4.2. *Period Length for the Higher Order ACORN Generators*

THEOREM 2. *The period length P_{k+1} for the $(k+1)$th-order ACORN generator is an integer multiple of the period length P_k for the kth order generator. If we define the integer α_k to be the sum, modulo M, of the Y^k_i over one entire period, then $0 \leqslant \alpha_k < M$, and for any value of n we can write*

$$\alpha_k = \left(\sum_{i=1}^{P_k} Y^k_{n+i} \right)_{\mathrm{mod}\,M}, \tag{16}$$

Then, if β_k is the smallest strictly positive integer s.t. $(\alpha_k \beta_k)_{\mathrm{mod}\,M} = 0$, the period lengths are related by $P_{k+1} = \beta_k P_k$. The period lengths are equal if and only if $\alpha_k = 0$.

Proof. Suppose that the $(k+1)$th order generator has period length $P_{k+1} = P$. Then

$$Y^{k+1}_{n+P} = Y^{k+1}_n \qquad \text{for all} \quad n \geqslant 0. \tag{17}$$

Making use of Eq. (12), we get that

$$(Y^k_{n+P} + Y^{k+1}_{n+P-1})_{\mathrm{mod}\,M} = \left(Y^k_n + Y^{k+1}_{n-1} \right)_{\mathrm{mod}\,M}. \tag{18}$$

Since Y^{k+1}_n is periodic, with period P, this implies

$$Y^k_{n+P} = Y^k_n \qquad \text{for all} \quad n \geqslant 0. \tag{19}$$

Hence the period length P_k for the kth order generator must divide $P = P_{k+1}$ exactly.

Combining (16) with (12), we obtain

$$Y_{n+P_k}^{k+1} = \left(Y_n^{k+1} + \sum_{i=1}^{P_k} Y_{n+i}^k \right)_{\text{mod } M}$$
$$= (Y_n^{k+1} + \alpha_k)_{\text{mod } M} \qquad \text{for any } n. \tag{20}$$

It is clear from (20), together with the definition (16), that $Y_{n+P_k}^{k+1}$ is equal to Y_n^{k+1} for all n if and only if $\alpha_k = 0$; in this case the period length P_{k+1} will be equal to P_k.

Suppose now that $\alpha_k \neq 0$. By repeated application of (20),

$$Y_{n+\beta P_k}^{k+1} = (Y_n^{k+1} + \alpha_k \beta)_{\text{mod } M} \tag{21}$$

for any μ and for any strictly positive integer β; if β_k is the smallest such β which satisfies $(\alpha_k \beta_k)_{\text{mod } M} = 0$, then clearly $P_{k+1} = \beta_k P_k$.

THEOREM 3. *If Y_0^1 is a multiple of H (possibly zero) then the second-order ACORN generator has a period of length μ or 2μ according to whether μ is odd or even (where, as before, H is the highest common factor of M and Y_0^0 and $\mu = M/H$).*

Proof. Theorem 1 tells us that the period length for the first-order ACORN generator is equal to μ, while Theorem 2 tells us that the period length for the second-order ACORN generator must be an integer multiple of this.

Suppose that the period length is $k\mu$, where k is an integer. By repeated application of Eq. (12) we obtain

$$Y_{n+k\mu}^2 = \left(Y_n^2 + \sum_{i=1}^{k\mu} Y_{n+i}^1 \right)_{\text{mod } M}. \tag{22}$$

If Y_0^1 is a multiple of H, then Y_{n+i}^1 must take each of the values $H(i-1)$ exactly once in some order as i ranges from 1 to μ and exactly k times as i ranges from 1 to $k\mu$. Thus

$$Y_{n+k\mu}^2 = \left(Y_n^2 + kH \sum_{i=1}^{\mu} (i-1) \right)_{\text{mod } M}$$
$$= (Y_n^2 + kH\mu(\mu-1)/2)_{\text{mod } M}. \tag{23}$$

Thus $Y_{n+\mu}^2$ is equal to Y_n^2 if and only if

$$(kH\mu(\mu-1)/2)_{\text{mod } M} = 0, \tag{24}$$

or, equivalently,

$$(k\mu(\mu-1)/2)_{\text{mod } \mu} = 0. \tag{25}$$

The period length for the second-order ACORN generator is then equal to $k\mu$ for the smallest value of $k \geqslant 1$ such that (25) holds.

If μ is odd, then $(\mu - 1)$ is divisible by 2 and so (25) holds for $k = 1$. Thus in this case the period length is μ.

If μ is even, then $(\mu - 1)$ is not divisible by 2, and so (25) only holds if k is divisible by 2. This gives a period of length 2μ.

COROLLARY TO THEOREM 3. *If $M = 2^k$ and Y_0^0 is odd, then the second-order ACORN generator has a period of length $2M$.*

4.3. *Period Length for the Linear Congruential Generators*

Equation (1) can be written in the form

$$V_n = (aV_{n-1} + d)_{\mathrm{mod}\, M}, \qquad n > 1, \tag{26}$$

where V_n takes integer values in the range $0 \leqslant V_n < M$ and d is an integer-valued constant. Defining

$$U_n = V_n / M,$$

the above equations are related to (1) in the same way as (11)–(13) are related to (3) and (4).

Knuth [1, pp. 16–20] discusses two theorems concerning the period length for linear congruential generators. The first, the proof of which was due to Hull and Doble [10], states that for the linear congruential generator given by (26) with non-zero d the maximum period length is equal to M and gives conditions for this maximum period length to be achieved. The second, due to Carmichael [11], considers the case of multiplicative generators ($d = 0$) and gives the maximum period lengths for different values of M, together with conditions for achieving the maximum period; in particular, if M is prime we obtain a period length of $M - 1$ (the maximum achievable for any purely multiplicative generator), while for $M = 2^k$, the maximum period is $M/4$.

4.4. *Comparison of Period Lengths*

In the preceding sections we have shown that for a given value of M, it is simple to choose a seed Y_0^0 such that the first-order ACORN generator has a period length equal to M, while the period length of the $(k+1)$th-order ACORN generator is an integer multiple of the period length for the kth order generator; further, the multiplier is only equal to 1 in special cases.

For the linear congruential generator, the maximum possible period length is also equal to M, but this will only be achieved by certain special choices of the seed, multiplier, and additive constant.

Thus it is clear that for the higher order ACORN generators, the period lengths which can be achieved are many times longer than for the linear congruential generators with the same value of M.

5. Empirical Tests

The ACORN generator was subjected to various empirical tests of randomness. A full description of each of these tests is given in reference [1, Section 3.3.2]. The brief notes given below are intended merely to clarify the exact way in which the tests have been implemented here.

(i) Mean and variance.

(ii) Mean values of $\mu_2 = 4X_{2n-1}X_{2n}$, $\mu_3 = 8X_{3n-2}X_{3n-1}X_{3n}$, and $\mu_4 = 16X_{4n-3}X_{4n-2}X_{4n-1}X_{4n}$. For independent random numbers, the mean of the product would equal the product of the means, and so we would expect a value of 1 in each case.

(iii) Frequency distribution test. The interval [0, 1) was divided into 100 equal sub-intervals and a χ-square test applied to the frequency of occurrence of the random numbers in each of the sub-intervals.

(iv) Serial test for successive pairs. The unit square was divided into 10×10 equal intervals and a χ-square test applied to the frequency of occurrence of the pairs (X_{2n-1}, X_{2n}) in each of these sub-intervals.

(v) Permutation test. The sequence of random numbers was divided into groups of 3 elements, and a χ-square test applied to the frequency of occurrence of each of the $3! = 6$ possible orderings of the 3 elements.

(vi) Gap test. We consider the length of gaps between occurrences of random numbers in the range $0.3 \leqslant X_i \leqslant 0.6$. The frequency of gaps of length, 0, 1, 2, 3, 4, 5, 6, 7, and 8 or more was counted, and a χ-square test applied to the results.

(vii) Poker test. The interval [0, 1) was divided into five equal sub-intervals; for groups of five successive numbers the number of distinct sub-intervals occurring were counted, and a χ-square test applied to the frequency of occurrence 5, 4, 3, and less than 2 sub-intervals in a group of 5 successive numbers.

(viii) Maximum of t test. Let $V_j = \max(U_{tj+1}, U_{tj+2}, \ldots, U_{tj+t})$. The frequency distribution test was applied to the sequence V_j, $j = 0, 1, 2, \ldots$. It should be noted that for $t = 1$ this reduces simply to the equidistribution test and so was only applied for values of t in the range $2 \leqslant t \leqslant 10$.

For comparison purposes, the ACORN generator was implemented in FORTRAN on an IBM PC-XT, and the results were compared with those obtained using the subroutine G05CAF from the NAG subroutine library [5] and with the Chebyshev mixing method of Erber, Everett, and Johnson [6].

The subroutine G05CAF uses the multiplicative congruential method (1) with multiplier $a = 13^{13}$ implemented in the form

$$U_n = N_n / 2^{59},$$

where $N_n = (13^{13} \times N_{n-1}) \bmod 2^{59}$ and the initial seed $N_0 = 123456789 \times (2^{32} + 1)$.

The Chebyshev mixing method (2) was implemented in double precision using the initial seed $Z_0 = \pi - 3$.

In a recent paper, Hosack [12] applied the above tests, with the exception of the poker test, to the Chebyshev mixing method as defined above. Hosack also applied a number of other tests for randomness. In general, the results for the Chebyshev mixing method reproduce those of Hosack and are included here to provide a further comparison for the ACORN generator. However, there is one anomaly, which is discussed in the results.

The ACORN generator of order (k) was implemented for values of $k \leqslant 10$ in double precision using the method in 2(i) to generate the numbers for the statistical testing and the method in 2(ii) for the comparison of execution times with a seed $X_0^0 = \varepsilon\sqrt{2}/1.42$ where $\varepsilon = 0.1$ and initial values $X_0^i = 0$, $i = 1, ..., k$. Similar results have been obtained using other values for the seed, provided that it is chosen to have no obvious regularities or repetitions of digits, as well as for non-zero initial values.

In each case, a sequence of 10,000 random numbers was generated, and the empirical tests of randomness (i)–(viii) above applied to the results. The tests were all applied at the 0.05 level of significance.

6. Results

Table I summarises the results obtained; asterisks represent failures at the 0.05 significance level. The first-order ACORN generator performs very poorly as a source of uniform random numbers. This is to be expected, since the first-order ACORN generator is simply a mixed congruential generator with $a = 1$.

As the order of the ACORN generator is increased, there are fewer and fewer failures, until for $k = 5$ the generator fails only on the gap test, and then only marginally (a value for χ^2 of 17.55 compared with a 0.05 significance limit of 17.54). For values of $k \geqslant 6$, the ACORN generator passes all the test applied, with one exception for $k = 7$ (the generator failed the serial test at the 0.05 level) and two exceptions for $k = 8$ (the maximum of t tests for $t = 3$ and $t = 9$ failed at the 0.05 level in this case). The same tests have been applied to the ACORN generator using different seeds (e.g., X_0^0 defined as above with $\varepsilon = 0.01$ and $\varepsilon = 0.001$) with very similar results: the ACORN generators for $k \geqslant 5$ appear to pass all the tests, with a few isolated exceptions. There was no discernible pattern to the particular tests which were failed for the different seeds and different orders of generator.

The performance of the ACORN generators for $k \geqslant 5$ is comparable with that of the NAG routine G05CAF, which passed all the tests at the 0.05 level.

The results for the Chebyshev generator reproduced those of Hosack [12]. Its performance is very variable, giving reasonable results for the mean and variance and passing the frequency distribution test, but performing poorly on the remaining tests. An anomalous result for this generator is that it fails the maximum of t tests

TABLE I

Results of Applying Statistical Tests to the Random Number Generators

TEST	DEGR. OF F'DOM	5% SIGNIF. LIMITS FOR CHI-SQUARED	ACORN GENERATOR										MIXED CONGR. METHOD	CEBYSEV METHOD
			K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10		
MEAN			.5002	.4977	.4990	.4954	.4996	.5012	.4992	.4951	.4999	.5012	.4976	.4938
VARIANCE			.0833	.0821	.0844	.0833	.0832	.0844	.0830	.0829	.0842	.0842	.0842	.0854
M2			1.158	.9873	1.000	.9855	1.005	.9984	1.001	.9892	.9947	1.011	.9922	.9794
M3			1.325	.9559	.9847	.9766	1.018	.9894	1.013	.9779	.9761	1.013	.9792	.9821
M4			1.423	.9407	.9720	1.030	.9783	.9783	.9880	.9588	1.002	1.010	.9752	.9580
FREQUENCY DISTR. TEST	99	73.36-128.4	1.440%	76.56	103.0	90.44	109.1	109.7	122.2	89.00	110.9	81.52	88.65	107.8
SERIAL TEST	99	73.36-128.4	>1000%	98.48	135.6%	127.2	95.08	109.5	145.2%	117.6	90.64	97.24	100.0	>1000%
PERMUTATION TEST	5	0.831-12.83	>1000%	119.0%	4.608	4.752	2.674	3.711	1.587	8.932	3.599	4.381	4.395	>1000%
GAP TEST	8	2.180-17.54	>1000%	188.3%	6.120	6.026	17.55%	9.178	3.148	11.48	14.14	13.89	3.229	>1000%
POKER TEST	3	0.216-9.348	>1000%	3.383	18.92%	10.05%	2.130	2.820	7.114	3.343	2.002	0.337	3.313	725.4%
MAX. T= 2	99	73.36-128.4	>1000%	86.24	128.2	89.20	107.5	111.4	108.9	107.4	81.12	126.2	73.74	>1000%
OF T T= 3			>1000%	141.1%	93.55	93.85	105.7	94.75	111.1	138.3%	93.49	111.1	90.36	>1000%
TEST T= 4			>1000%	134.2%	105.9	90.72	96.24	105.9	115.4	111.0	90.96	114.2	89.80	866.8%
T= 5			>1000%	143.3%	116.8	100.3	84.10	101.5	123.5	99.70	77.10	125.9	106.0	632.6%
T= 6			569.5%	115.8	109.0	92.94	94.98	92.46	115.4	102.8	92.70	106.9	104.7	498.1%
T= 7			388.5%	135.7%	101.6	96.09	93.57	115.8	98.61	120.5	122.3	111.8	95.78	477.2%
T= 8			380.4%	132.7%	103.0	98.00	89.20	124.2	97.36	107.4	89.84	97.68	107.8	428.1%
T= 9			483.0%	132.8%	98.99	95.39	98.27	111.6	112.3	128.7%	101.5	110.5	104.0	396.9%
T=10			639.6%	119.2	140.0%	99.20	80.80	106.6	102.2	77.60	87.80	128.0	117.5	341.0%

for all values of t between 2 and 10, whereas Hosack [10] found that it passed this test; however, no details were given concerning his implementation of the test.

Table II shows a comparison of execution times on an IBM PC-XT for the ACORN generators for $k = 1, 2, ..., 10$, the NAG routine G05CAF and the Chebyshev generator. The table gives the cpu time (in milliseconds) per random number generated, calculated in each case as the average over 10,000 calls to the appropriate function. Figure 4 shows the cpu time per random number generated; the corresponding times for the Chebyshev generator and the NAG routine are shown as horizontal lines. It is clear that the execution time for the ACORN generator increases linearly with the order k. The execution time for the ACORN generator with $k = 4$ is comparable with that for the Chebyshev generator. The NAG routine G05CAF is considerably slower than either of the other generators; by extrapolation of the line on Fig. 4 , we can estimate that the execution time for G05CAF is comparable with that of a 34th order ACORN generator. The same comparisons have also been made on other machines and there is some variation in the relative speeds of the NAG routine G05CAF and the ACORN generators: on an IBM PC-AT all routines executed faster but the time for the NAG routine was comparable with the 10th-order ACORN generator, while on a VAX 8600 the time compared with that for the 4th-order ACORN generator. There is a need for a more detailed comparison of execution times on different machines to resolve these anomalies.

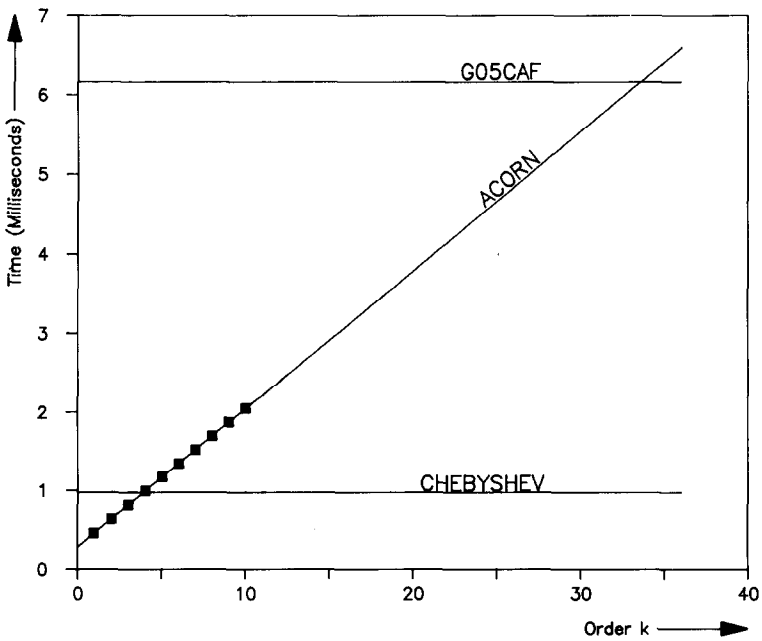

FIG. 4. Comparison of execution times per random number generated for the kth order ACORN generator for various values of k, the NAG routine G05CAF and the Chebyshev generator.

TABLE II

Comparison of Timing of the Different Generators

Generator	Time (ms/function call)
ACORN	
$K=1$	0.461
$K=2$	0.637
$K=3$	0.812
$K=4$	0.988
$K=5$	1.158
$K=6$	1.334
$K=7$	1.510
$K=8$	1.686
$K=9$	1.861
$K=10$	2.037
Chebyshev	0.977
NAG Routine G05CAF	6.168

7. Conclusions

A new family of pseudo-random number generators, the ACORN (*a*dditive *co*ngruential *r*andom *n*umber) generators, has been proposed. The resulting random numbers are uniformly distributed in the interval [0, 1). The ACORN generators are defined recursively, and the $(k+1)$th order generator is easily derived from the kth order generator.

A range of statistical tests has been applied to the ACORN generators for $k \leqslant 10$, to the NAG implementation of the linear congruential generator, and to the Chebyshev generator. While the performance of the Chebyshev generator and the low-order ACORN generators was mixed, the performance of the ACORN generators was seen to improve as the order increased up to $k=5$. For values of $k \geqslant 5$ the performance was comparable with that of the linear congruential method, which passed all the tests at the 0.05 level of significance.

On an IBM PC-XT the execution time for the ACORN generators was considerably faster than for the linear congruential method, while the period lengths which can be achieved are longer even for relatively small values of k, and they increase with k. Thus the ACORN generator is of particular value in applications where very long sequences of random numbers are required, when period length and execution times both become significant factors. There was considerable variation in the relative speeds on different machines, and further work is required to resolve these anomalies. The ACORN generator can be very simply coded in-line

rather than as a function call, leading to further saving in execution time. Where period length is the overriding consideration, the ACORN generator can be readily implemented in arbitrary precision arithmetic, since the only operation required is addition modulo 1; by a suitable choice of seed and precision, a period length in excess of any desired value can be obtained for the first-order ACORN generator, and a period length considerably in excess of this for the kth order generator for any $k \geqslant 2$.

ACKNOWLEDGMENT

The author would like to thank Dr. C. L. Farmer for introducing him to some of the problems of generating random numbers and for some very useful discussions concerning this paper.

REFERENCES

1. D. KNUTH, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms* (Addison–Wesley, Reading, MA, 1981).
2. D. H. LEHMER, in *Proceedings, 2nd Symposium on Large-Scale Digital Calculating Machinery, Cambridge* (Harvard Univ. Press, Cambridge, MA, 1951), p. 141.
3. W. E. THOMPSON, *Comput. J.* **1**, 83 (1958).
4. A. ROTENBERG, *J. Assoc. Comput. Mach.* **7**, 75 (1960).
5. NAG, *Fortran PC50 Library Handbook—Release 1* (Numerical Algorithms Group Ltd., Oxford, UK. 1983) (unpublished).
6. T. ERBER, P. EVERETT, AND P. W. JOHNSON, *J. Comput. Phys.* **32**, 168 (1979).
7. R. C. TAUSWORTHE, *Math. Comput.* **19**, 201 (1965).
8. J. H. HALTON, *SIAM Rev.* **12**, 1 (1970).
9. T. ERBER, T. M. RYNNE, W. F. DARSOW, AND M. J. FRANK, *J. Comput. Phys.* **49**, 394 (1983).
10. T. E. HULL AND A. R. DOBLE, *SIAM Rev.* **4**, 230 (1962).
11. R. D. CARMICHAEL, *Bull. Amer. Math. Soc.* **16**, 232 (1910).
12. J. M. HOSACK, *J. Comput. Phys.* **67**, 482 (1986).