The Generation of Pseudo-Random Numbers on Electronic Digital Computers

by A. R. Edmonds

Summary: Different methods of generation of sequences of pseudo-random numbers on electronic digital computers are discussed. Statistical tests of randomness which can be applied to such sequences are considered, and an account is given of pseudo-random number subroutines which have been written for the Pegasus and Mercury computers.

1 INTRODUCTION

In the last few years there has been an increasing demand from users of electronic digital computers for long sequences of random numbers. There are, broadly speaking, three ways of supplying such sequences.

(i) Tables of random numbers may be recorded on paper tape or punched cards. (Tippett, 1927; Kendall and Babington Smith, 1939; The Rand Corporation, 1955.) The tapes or cards are fed into the computer at suitable stages of the calculation. This method has found little favour, since many problems require very long sequences of random numbers (e.g. Monte Carlo calculations) and the time taken to read in the tables may soon become excessive. The labour needed to prepare the tapes or cards must also be taken into consideration.

(ii) Random numbers may be generated by physical processes such as radioactivity or discharges in gases. (Cf. Dodd, 1953; Thomson, 1959.) Extra equipment is required, the cost of which may be an appreciable fraction of the cost of the computer. The chief objection is a rather paradoxical one; the number sequences cannot be repeated and so it is very difficult to check methods of calculation or programs. For it is not always possible to distinguish between variations in results due to genuine random fluctuations and those due to changes in the program or even to the faulty running of the computer.

These methods have therefore been rejected in favour of (iii) computer subroutines which produce, by recursion, sequences of so-called *pseudo-random* numbers. The computational procedures used in the subroutines are such that these sequences satisfy to a required degree some statistical tests of randomness.

We may represent the typical recursion relation of these subroutines by

$$x_{n+1} = R(x_n) \qquad (1)$$

where the x_n are computer numbers of r digits, the value of r depending on the computer being used. We shall assume without much loss of generality that the numbers are represented in the binary scale.

The relation (1) symbolizes a finite number of computer operations carried out on the digits of x_n; these operations are not necessarily restricted to arithmetical processes such as addition or multiplication.

Now the x_n can have only 2^r possible different values, and (assuming the usual properties of digital computers) $R(x)$ is a single-valued function of x. The relation (1) and an initial value x_0 of x define a sequence x_0, x_1, x_2, . . . As we go through this sequence we must eventually arrive at an x_n with a value identical with that of some previous element x_i $(i < n)$ of the sequence. If $i > 0$ we have a non-cyclic sequence x_0, x_1, . . . x_{i-1} entering a cycle at x_i. If $i = 0$ the whole sequence is cyclic.

There are three criteria which should be satisfied by any method of generation of pseudo-random numbers.

(i) The numbers produced should satisfy the tests of randomness prescribed by the would-be user.

(ii) The rate of production of pseudo-random numbers should compare favourably with the rate at which they may be used up in typical computations. In certain problems, e.g. many of those arising in operational research, the amount of computation between the input of successive random numbers is quite small. A method which makes good use of the characteristics of the computer to carry out each recursion in the shortest time possible is needed in such cases. It will be clear that the best method for one computer is not necessarily the best for another.

(iii) The recursion relation should produce a sufficiently long sequence of pseudo-random numbers. As has been mentioned already, all sequences produced by recursion relations of the type represented by equation (1) are either cyclic or enter a cycle if prolonged far enough. A cyclic sequence will not be a satisfactory source of pseudo-random numbers unless the period is so great that it is of no consequence in practical computations.

2 THE TESTING OF SEQUENCES FOR RANDOMNESS

Let us consider a finite sequence of symbols, each symbol being taken from a set D of t distinct symbols. We shall follow Good (1953) and say that such a sequence S of symbols a_1, a_2, a_3, . . . a_N is *random* if (i) given α and j, the probability that $a_j = \alpha$ $(j = 1, 2, \ldots N; \alpha \epsilon D)$ is p_α, where p_α is independent of j, and

$$\sum_{\alpha \epsilon D} p_\alpha = 1$$

and (ii) the probability is p_α even if some or all of the other symbols in the sequence S are known.

Good defines a *perfectly random sequence* as one in which p_α is independent of α and therefore is equal to t^{-1}. We shall be concerned in this paper with sequences which approximate to perfectly random sequences.

Downloaded from https://academic.oup.com/comjnl/article/2/4/181/470572 by guest on 13 March 2024

The first task in testing the output of a generator of alleged random numbers is to establish that there are no systematic biases or correlations in the digits produced. At this stage we are not particularly concerned with the way in which the numbers will be used; we carry out tests on long sequences of binary digits to ascertain whether these sequences are in a general sense perfectly random.

Suppose that we have established that the output of a generator is perfectly random in the sense that very long sequences of digits have passed tests of randomness. For the purposes of the following argument it is irrelevant whether the generator is a physical process or a computer subroutine. In a particular application a block of digits of a definite length may be required to be random when considered in isolation. The remark of Kendall and Babington Smith (1938) is relevant here, namely (p. 155), "if a series S is locally random in a Domain, it does not follow that any part of S is locally random in that Domain." They conclude that a set of random numbers which is adequate for all requirements is impossible, and the only solution is to carry out tests on blocks of numbers and give the results of these tests, so that the prospective user can choose from the tables those blocks which are suitable for his problem.

This procedure can in principle be applied to sequences of pseudo-random numbers produced on digital computers; however, numbers can be produced by the million and different users may need blocks of widely differing sizes. It is thus not at all easy to decide how to present the numbers describing the statistical properties of the output of a pseudo-random number generator.

If we are considering a sequence of *numbers* rather than of the symbols referred to at the beginning of this section, the symbols in the set D will be replaced by all the r-digit binary numbers and t will be equal to 2^r. The statistical tests will be carried out with these numbers, usually taken to be either the integers $0, 1, 2, \ldots 2^r - 1$ or the fractions $0, 1.2^{-r}, 2.2^{-r}, \ldots 1 - 2^{-r}$. However, a sequence of numbers may satisfy tests of randomness to an apparently high degree without all the individual digits being random. This is due to the fact that the significance of the digits composing the numbers decreases as we move to the right of each number, and lack of randomness in the digits on the extreme right of the numbers may thus not be detected.

A computer programmer might be tempted for reasons of convenience to make use of supposedly random digits appearing in the right-hand ends of the registers of the computer, possibly throwing away the rest, not realizing that statistical tests had been carried out effectively only on the more significant digits.

This circumstance is likely to arise, for example, with the Mercury computer, where the 40-bit register which would contain the output of the pseudo-random number generator can be addressed as four 10-bit registers.

There are, in fact, methods of generation of pseudo-random numbers which give quite good randomness to the digits on the left, while those on the right cannot be said to be random at all; these methods will be discussed in the next section.

The tests of Kendall and Babington Smith (1938) have been used frequently on sequences of decimal digits, although there seems to be no reason to suppose them to be superior to other possible tests. They may be adapted in various ways for testing sequences of binary digits produced by pseudo-random number generators.

These tests are four in number. In the *frequency* test, the frequencies of the different digits in the sequence are recorded. In the *serial* test, a bivariate table is prepared showing the distribution of pairs of digits in the sequence, arranged in rows according to the first digit and in columns according to the second. The digits may also be considered in blocks of, say, five, and the number of blocks in which all digits are the same, the number with four of a kind, etc., are counted. This is called the *poker* test. The *gap* test is concerned with the gaps between successive digits of the same kind in the sequence, say, between the zeros. The number of gaps of length zero, of length one, etc., are counted.

In all these tests the deviations of the observed counts from those expected from a perfectly random sequence are studied. Chi-squared tests are used to give a measure of the permissible divergence from expectation.

Some care may be necessary in adapting these tests to binary sequences. Good (1953) has considered the generalized serial test, in which the frequencies of occurrence of sub-sequences of two or more digits are studied. He has pointed out that the mode of use of the chi-squared test by Kendall *et al.* (1938) in the serial test is incorrect, and has particularly serious consequences when binary sequences are being investigated.

3 METHODS OF GENERATION OF PSEUDO-RANDOM SEQUENCES

It is a futile occupation to carry out detailed statistical tests of randomness on the output of a generator unless we have some knowledge of the *cycle structure* of the sequence as a whole. The proof of local randomness is of no account if it is the case that the digits tested lie in a cycle of which the period is so short that the sequence may be repeated during the running of a computation, or if the digits lie in a non-cyclic sequence which collapses all too soon into a small loop.

From this point of view methods of generation fall into two classes.

(*a*) Those for which there is, or appears to be, no way of determining the cycle structure other than the brute force procedure of investigating the sequence by actual computation. A moment's thought will show that this is ideally not merely a matter of computing, say, hundreds of thousands of recursions of the basic relation; intermediate terms should be stored and compared with the current terms of the sequence.

(*b*) Those for which mathematical analysis can determine the cycle structure, and even suggest suitable parameters in the recursion relation to give the longest period and most satisfactory output.

Downloaded from https://academic.oup.com/comjnl/article/2/4/181/470572 by guest on 13 March 2024

The best-known method of type (*a*) is the so-called *middle-squares* method of Von Neumann and Metropolis (Metropolis, 1956). The number x_{n+1} is obtained from the r-digit x_n by selecting the middle r digits from the $2r$-digit number got by squaring x_n. This procedure can be carried out rapidly only on a computer which has shift operations which link the two halves of a $2r$-digit product register. Metropolis (1956) has examined the behaviour of sequences for values of r appreciably smaller than those normally used in computers, but nevertheless has demonstrated graphically the unpredictable qualities of the sequences generated by the middle-squares method.

Other methods of type (*a*) have been proposed, but do not seem to have found much favour.

Sequences of type (*b*) may be obtained by the reduction of certain sequences of integers. These sequences are defined by the relations

$$u_0 = 0, u_1 = 1; u_{n+1} = u_n + u_{n-1} \quad (n = 0, 1, 2, \ldots) \quad (2)$$

$$u_0 = a; u_{n+1} = ku_n \quad (n = 0, 1, 2, \ldots) \quad (3)$$

where a and k are positive integers.

The sequence (2) is known as the *Fibonacci* series.

If M is a positive integer, which in the case of the sequence (3) is supposed prime to a, then for both (2) and (3) the least non-negative residues mod M of the elements $u_0, u_1, \ldots$ form a periodic sequence. It was first suggested by Lehmer (1951) that such reduced sequences might be used as sets of pseudo-random numbers.

These sequences have the virtue that, given the values of M and in case (3) of a and k, the period may be computed by the use of number theory.

The reduction of a sequence of integers for general integer M will involve division operations, which on most computers are very time-consuming. It has therefore been the practice to choose values of M which make explicit division unnecessary. The most obvious value to take is 2^h, where h is chosen to correspond to the number of binary positions in the accumulator of the computer. The reduction consists of simply keeping the h least-significant digits in a $2h$-digit product, or ignoring the overflow on addition in an h-digit accumulator. (See Thomson (1958) for a useful modification to this method.)

This procedure has the merit of simplicity, but, as Lehmer (1949) has pointed out, the digits on the right-hand ends of the numbers produced are subject to short periods. Values of M of the type 2^h may, however, be satisfactory when only the left-hand or more-significant digits are employed.

Such a generator will, of course, be perfectly safe in the hands of someone acquainted with its limitations; but if we have the task of writing a random-number subroutine for general use we must bear in mind some of the remarks made in Section 2 of this paper. It is, in fact, desirable to have a procedure which produces, in every position of the register, digits which are individually random. This property in a generator must entail a certain sacrifice in speed.

From this point of view a suitable choice of M has been suggested by Lehmer (1949), namely $M = 2^h \pm 1$. The residues mod M are computed in quite a simple fashion, as a result of the following considerations. We may represent a positive integer A less than 2^{2h} by the contents a and α of two registers each containing h bits, e.g. the two halves of the product register. Thus $A = a + \alpha \cdot 2^h$. Then the (possibly negative) residue of A mod $(2^h + 1)$ will be $a - \alpha$ and the not necessarily least positive residue of A mod $(2^h - 1)$ will be $a + \alpha$. For clearly $A = a - \alpha + \alpha(2^h + 1) = a + \alpha + \alpha(2^h - 1)$. It is a straightforward matter to program a computer to give the desired least non-negative residue in either case without the necessity of carrying out a division.

The reduction of both the sequences (2) and (3) can give us periodic sequences of numbers with periods sufficiently large to make them otherwise suitable as sequences of pseudo-random numbers. However, it has been found by the application of statistical tests that sequences derived from the Fibonacci series are not very satisfactory from the point of view of randomness (see Taussky and Todd, 1956). Attempts have been made to remedy this failing by modifying the procedure somewhat (Neovius, 1955), but these modifications are only effective on certain computers.

If the degree of randomness required is not high, this method is attractive, since numbers are generated much faster than by other procedures.

We consider now the choice of the parameters M, a and k for the reduced sequence of type (3). The necessary theory is given by Duparc, Lekkerkerker and Peremans (1953) or, in a more general form, by Hardy and Wright (1956).

The maximum period possible for the sequence is determined by the value of M.

We consider first the case of $M = 2^h$. Duparc *et al.* (1953) show that the maximum period is 2^{h-2}, and that the period of the sequence has this value only when the multiplier k takes a value which is an odd power of one of the integers 3, 5, 13, 19, 21, . . .

In the case of $M = 2^n \pm 1$ we assume

$$M = 2^h \pm 1 = p_1 p_2 p_3 \ldots$$

where $p_1, p_2, p_3 \ldots$ are different primes. The case of repeated prime factors is not important, since this corresponds to a relatively short period, and we may reject such an M outright.

The maximum period for an M of the above form is equal to the L.C.M. of the numbers $p_1 - 1$, $p_2 - 1$, $p_3 - 1$, . . . and we denote it by $L(M)$. This maximum period is obtained only for certain values of the multiplier k. Such a value of k can be found by a tentative procedure which is described by Duparc *et al.* (1953). It is easiest to search for a suitable k through the small integers. However, it seems likely that putting a small k into the recursion relation may give an undesirable correlation between successive x_n. This difficulty is

Downloaded from https://academic.oup.com/comjnl/article/2/4/181/470572 by guest on 13 March 2024

overcome by replacing the k obtained by the Duparc procedure by a power of that k; the exponent must be prime to $L(M)$ to give a period equal to $L(M)$.

In choosing M we must consider three points.

(*a*) For speed of production of pseudo-random numbers M should be small enough for double-length arithmetic to be unnecessary.

(*b*) The largest integer in the sequence produced will be $M - 1$; thus the larger M the greater the number of pseudo-random digits produced at each recursion.

(*c*) M should be chosen so that a long period is possible.

It might be thought that the maximum period $L(M)$ would in general increase with M, so that one could satisfy (*b*) above while also satisfying (*c*). However, an example will show that things are not so simple. Suppose we have a computer with 39-bit registers, e.g. the Ferranti Pegasus machine. It is natural to consider first the values $2^{38} + 1$ and $2^{38} - 1$ for M. Since $2^{38} + 1 = 5.229.457.525313$, $L(2^{38} + 1)$ is equal to 525 312, and $L(2^{38} - 1)$ is even less. On the other hand, if we take the Mersenne prime $2^{31} - 1$ we get $L(M) = 2^{31} - 2 = 2\,147\,483\,646$. This choice has the disadvantages that only thirty positions in the computer register are filled with pseudo-random digits, and since shift operations are necessary to carry out the arithmetic, the resulting subroutine is somewhat slower than that for $M = 2^{38} + 1$ would be. There is no value of h between 38 and 31 which gives a period significantly longer than $L(2^{38} + 1)$.

With $M = 2^{31} - 1$ the maximum period can be obtained with $k = 13$. A suitable value in the recursion relation is thus $k = 455\,470\,314 \equiv 13^{13} \bmod (2^{31} - 1)$, since the exponent 13 is prime to the period $2^{31} - 2$.

Since this M is prime, any starting value u_0 may be taken. The above values of M and k have been used in the Pegasus library routine R 980. This routine takes $5\frac{1}{2}$ milliseconds to generate each 30-bit pseudo-random number.

A pseudo-random number generator of similar type has been devised for the Ferranti Mercury computer. The problem of choosing suitable M and k values is in this case rather more complicated, since Mercury is a floating-point machine with a 30-bit register for the numerical part, and a 10-bit register for the exponent. Thus a smaller M had to be chosen, namely $M = 2^{29} + 1$. This M gives a maximum period $L(M) = 3\,033\,168$, much less than that for the Pegasus routine, but probably satisfactory for most purposes.

The multiplier k was taken to be

$$k = 366\,714\,004 \equiv 7^{11} \bmod (2^{29} + 1).$$

The routine was written so as to give 29 pseudo-random bits in the 30-bit number register (the left-hand or sign digit being always zero). De-standardization instructions are incorporated in the routine so that the exponent is always zero. Since the 30-bit number register can be addressed as three 10-bit registers, a simple programming trick will give three positive numbers, each with nine significant bits, from each recursion. This number of significant digits is quite sufficient for a large class of computations which use random numbers.

With this routine each recursion takes 1·5 to 2 milliseconds.

General statistical tests (cf. Section 2) have been carried out on very long sequences of binary digits produced by the Pegasus routine described above. The results have shown that the routine may be considered for most practical purposes to be a generator of perfectly random sequences.

A programme of tests is now being conducted at the University of London Computer Unit which will give information about the randomness of individual blocks of numbers produced by the Mercury routine. The results obtained will be published together with starting values (u_0) for each block, so that users will be able to choose sets of numbers with the statistical properties appropriate to their requirements.

ACKNOWLEDGEMENTS

I should like to thank Mr. G. E. Felton, Professor D. H. Lehmer and Mr. E. Nixon for useful discussions. The work on the Pegasus routine was carried out while I was at the London Computer Centre of Ferranti Ltd. I am grateful to Mr. B. B. Swann and my former colleagues there for encouragement and assistance.

REFERENCES

DODD, K. N. (1953). *Armament Research Establishment Report* 11/53 (the Ferranti Mark 1 and Mark 1* Computers), p. 90.

DUPARC, H. J., LEKKERKERKER, C. G., and PEREMANS, W. (1953). "Reduced Sequences of Integers and Pseudo-random Numbers," Report ZW 1953—002, Mathematisch Centrum, Amsterdam.

GOOD, I. J. (1953). "The Serial Test for Sampling Numbers and other Tests for Randomness," *Proc. Camb. Phil. Soc.*, Vol. 49, p. 276.

HARDY, G. H., and WRIGHT, E. M. (1956). *Introduction to the Theory of Numbers*, Oxford U.P.

JUNCOSA, M. L. (1953). "Random Number Generation on the B.R.L. High-Speed Computing Machines," Report No. 855, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland.

KENDALL, M. G., and BABINGTON SMITH, B. (1938). "Randomness and Random Sampling Numbers," *J. Roy. Stat. Soc.*, Vol. 101, p. 147.

KENDALL, M. G., and BABINGTON SMITH, B. (1939). "Random Sampling Numbers," Tracts for Computers, 24, *Cambridge U.P.*

LEHMER, D. H. (1951). "Mathematical Methods in Large-Scale Computing Units," Second Symposium on Large-Scale Digital Calculating Machinery, 1949, Harvard U.P., Cambridge, Mass., pp. 141–6.

LEHMER, D. H. (1949). "Review of Juncosa" (1953), *Mathematical Reviews*, Vol. 15, p. 559.

METROPOLIS, N. (1956). "Phase Shifts—Middle Squares—Wave Equation," Symposium on Monte Carlo Methods, University of Florida, 1954, Wiley, p. 29.

Downloaded from https://academic.oup.com/comjnl/article/2/4/181/470572 by guest on 13 March 2024

NEOVIUS, G. (1955). "Artificial Traffic Trials using Digital Computers," *Ericsson Technics*, Vol. 2, p. 279.
THE RAND CORPORATION (1955). "One Million Random Digits and 100,000 Normal Deviates," The Free Press, Glencoe, Illinois.
TAUSSKY, O., and TODD, J. (1956). "Generation and Testing of Pseudo-random Numbers," Symposium on Monte Carlo Methods, University of Florida, 1954, Wiley, p. 15.
THOMSON, W. E. (1958). "A Modified Congruence Method of Generating Pseudo-random Numbers," *The Computer Journal*, Vol. 1, p. 83.
THOMSON, W. E. (1959). "ERNIE—A Mathematical and Statistical Analysis," *J. Roy. Stat. Soc., Ser. A*, Vol. 122, p. 301.
TIPPETT, L. H. C. (1927). *Random Sampling Numbers*, Cambridge U.P.

Time-Sharing on the National-Elliott 802

by R. L. Cook

Summary: This article discusses some of the programming techniques that have been evolved at Elliott Brothers for dealing with on-line applications of the 802 computer in the process-control field. Time-sharing or program interruption methods are described with particular reference to a straightforward data-logging application.

THE 802 COMPUTER

The 802 was designed for two purposes: to perform as a small general-purpose computer, and also to act as the computing centre for on-line process-control systems.

The logical elements of the 802 consist of a junction transistor-magnetic core element shown in Fig. 1. By this means, the high reliability and the little or no maintenance that are required in on-line applications can be achieved.

The 802 has a magnetic-core store capable of holding 1,024 words, each of 33 binary digits. The size of the store can be extended to 4,096 words if necessary. The order code comprises 64 basic orders, most of which refer to the single accumulator and one specified store address. Great care has been taken to ensure that the order code is simple to learn, consistent, and without exceptions. Each 33-bit word can hold two 16-bit orders, together with a B digit. If the B digit is present, the first order is obeyed in the normal manner, but the (new) contents of the store address specified by this order are added to the second order before it is obeyed.

In this way any location of the store may be used as a B-modifier. If the B digit is absent, the two orders are obeyed sequentially without B modification.

There are three independent input channels on the 802: channel 1 is normally attached to a paper-tape input device; channel 2 can receive information, via a switch, from any number of different input devices; and the third channel consists of a set of manually-operated keys, called the *Number Generator*, that enables a single word to be entered. The switch on channel 2 is operated by program control, and by this means, any number of different devices, each capable of giving digital information, may be simultaneously attached to the 802 and individually switched. Two output channels are provided from the computer: channel 1 is fixed and normally operates a paper-tape punch, and channel 2 can be attached by means of a switch to any device capable of accepting digital information.

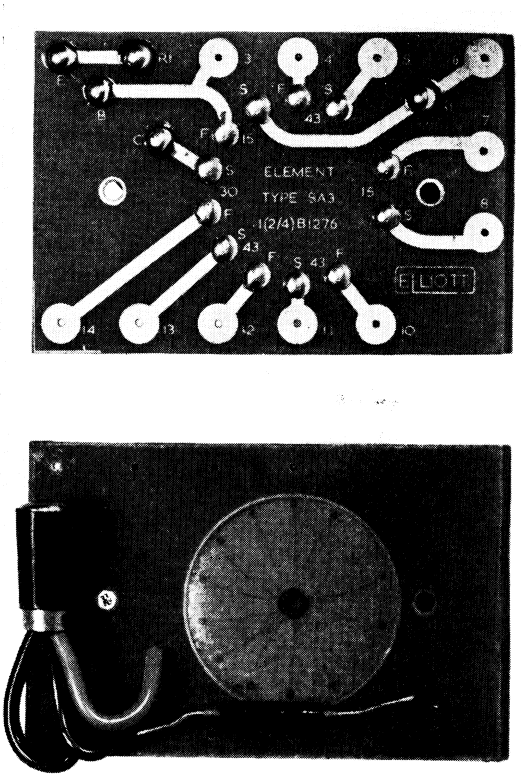

FIG. 1.—The junction transistor and magnetic core which make up the logical element of the 802. The use of solid state devices ensures the high reliability required for on-line working, in addition to minimizing the space and power required for the computer.

The completely transistorized 803 computer, which is functionally identical to the 802 except that it has a 4,096-word core store and a 39-digit word, is also used for process-control applications in the manner described

Downloaded from https://academic.oup.com/comjnl/article/2/4/181/470572 by guest on 13 March 2024