

Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator



B. A. Wichmann; I. D. Hill

Applied Statistics, Vol. 31, No. 2 (1982), 188-190.

Stable URL:

<http://links.jstor.org/sici?sici=0035-9254%281982%2931%3A2%3C188%3AAA1AEA%3E2.0.CO%3B2-K>

Applied Statistics is currently published by Royal Statistical Society.

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at <http://www.jstor.org/about/terms.html>. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at <http://www.jstor.org/journals/rss.html>.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

JSTOR is an independent not-for-profit organization dedicated to creating and preserving a digital archive of scholarly journals. For more information regarding JSTOR, please contact support@jstor.org.

Algorithm AS 183**An Efficient and Portable Pseudo-random Number Generator**By **B. A. WICHMANN**

and

I. D. HILL*National Physical Laboratory,
Teddington, Middx, TW11 0LW, UK**Clinical Research Centre,
Harrow, Middx, HA1 3UJ, UK*

[Received April 1981. Revised February 1982]

Keywords: PSEUDO-RANDOM NUMBERS; EFFICIENCY; PORTABILITY

LANGUAGE

Fortran 66

DESCRIPTION AND PURPOSE

Schrage (1979) has pointed out the advantages of pseudo-random generators that can be written in a high-level language and produce the same results on any machine. The generator that he presents, however, has the disadvantages: (1) like all simple multiplicative congruential generators, it does not work well at the extremes of the distribution—for any number produced that is less than 5.9499×10^{-5} the next number will simply be 16 807 times as much, and similarly at the top end; (2) on a 16-bit machine it has to use double precision arithmetic instead of integer arithmetic, which makes it very slow, and also uncertain that rounding errors could not occur.

Our algorithm does not have these difficulties. We claim that it is reasonably short, reasonably fast, machine-independent, easily programmed in any language, and statistically sound. It has a cycle length exceeding 2.78×10^{13} so that even using 1000 random numbers per second continuously, the sequence would not repeat for over 880 years. Consequently we have tested only small parts of it, consisting of many millions of numbers nevertheless. However, there are theoretical grounds for expecting good results, and the results of the tests we have made have been so satisfactory, that we are prepared to extrapolate our experience and infer that the sequence is satisfactory throughout.

The algorithm produces numbers rectangularly distributed between 0 and 1, excluding the end points.

METHOD

Three simple multiplicative congruential generators are used. Each uses a prime number for its modulus and a primitive root for its multiplier, which guarantees a complete cycle. The three results are added, and the fractional part is taken.

It is well known that the sum of n independent rectangular random numbers tends to normality as n increases, but much less well known that the fractional part of such a sum remains rectangular for all values of n . This is most easily seen by considering a simplified case, of two generators that produce numbers of only one digit after the decimal point such as shown in Table 1.

It is immediately clear from this table that if x_2 is equally likely to take any of its ten possible values, then so is the fractional part of $x_1 + x_2$, provided that x_1 and x_2 are independent, whatever the value of x_1 may be. It therefore remains so, whatever the distribution of x_1 . It follows that where we have n such variables, the joint distribution of $n - 1$ of them is immaterial provided that the other one is random rectangular and independent.

In practice, however, no pseudo-random generator is perfect, and adding several in this way leads to the imperfections of each being "ironed out" by the others. Furthermore, because each generator is prime to the others, they are clearly independent and the cycle-length is the product of the individual cycle-lengths. We have found $n = 3$ to be adequate.

TABLE 1
Fractional part of $x_1 + x_2$

<i>Value of x_2</i>	<i>Value of x_1</i>									
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.0	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.0
0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.0	0.1
0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.0	0.1	0.2
0.4	0.4	0.5	0.6	0.7	0.8	0.9	0.0	0.1	0.2	0.3
0.5	0.5	0.6	0.7	0.8	0.9	0.0	0.1	0.2	0.3	0.4
0.6	0.6	0.7	0.8	0.9	0.0	0.1	0.2	0.3	0.4	0.5
0.7	0.7	0.8	0.9	0.0	0.1	0.2	0.3	0.4	0.5	0.6
0.8	0.8	0.9	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.9	0.9	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8

STRUCTURE

FUNCTION RANDOM (L)

Formal parameter

L Integer input: a dummy argument. Any value will do.

Common

COMMON/RAND/IX, IY, IZ

These three integers must be set before the first entry, to values within the range 1–30 000, preferably chosen at random. Where the machine has facilities for accessing the date and the time, suitable values may be derived from these. If the values are printed before the first use of the algorithm, they are available for a repeat run if required. If they are printed after the last use, they give suitable values for entry on another occasion with the certainty of a different sequence.

Once set, these numbers should not be changed, other than by calls of the algorithm.

These values could be passed as arguments instead of through *COMMON*, but on many machines using *COMMON* will be faster.

RESTRICTIONS

Integer arithmetic up to 30 323 is required.

It is assumed, without testing, that the three integer values are within the correct range. It does not seem worth testing at every entry, because if the values are suitable at the first entry they necessarily remain so at subsequent entries.

MODIFICATION

A modified, and simpler, version is mentioned in a comment. This requires integer arithmetic up to 5 212 632 and may be slightly faster on some machines; however, it is much slower on other machines, and should not be used without testing for speed.

TIME

On a PDP-11 computer each call of the algorithm takes 0.85 ms. This compares with 4.4 ms. for Schrage's algorithm (in its double-precision version that is necessary for a 16-bit machine), and with 1.3 ms. for the generator included in the GLIM program.

TESTS

Results of the serial test, poker test, coupon collector's test and runs-up-and-down test performed on this generator are given in Wichmann and Hill (1982). This reference also contains further discussion. We are grateful to the referee for additional, very thorough, testing.

ASSOCIATED ALGORITHM

To produce pseudo-random normal deviates, the results of this algorithm may be used as input to AS 111 (Beasley and Springer, 1977).

REFERENCES

- BEASLEY, J. D. and SPRINGER, S. G. (1977). Algorithm AS 111. The percentage points of the normal distribution. *Appl. Statist.*, **26**, 118–121.
 SCHRAGE, L. (1979). A more portable Fortran random number generator. *ACM Trans. Math. Softw.*, **5**, 132–138.
 WICHMANN, B. A. and HILL, I. D. (1982). A pseudo-random number generator. *NPL Report DITC 6/82*.

```

FUNCTION RANDOM(L)
C
C      ALGORITHM AS 183  APPL. STATIST. (1982) VOL.31, NO.2
C
C      RETURNS A PSEUDO-RANDOM NUMBER RECTANGULARLY DISTRIBUTED
C      BETWEEN 0 AND 1.
C
C      IX, IY AND IZ SHOULD BE SET TO INTEGER VALUES BETWEEN
C      1 AND 30000 BEFORE FIRST ENTRY
C
C      INTEGER ARITHMETIC UP TO 30323 IS REQUIRED
C
COMMON /RAND/ IX, IY, IZ
IX = 171 * MOD(IX, 177) - 2 * (IX / 177)
IY = 172 * MOD(IY, 176) - 35 * (IY / 176)
IZ = 170 * MOD(IZ, 178) - 63 * (IZ / 178)
C
IF (IX .LT. 0) IX = IX + 30269
IF (IY .LT. 0) IY = IY + 30307
IF (IZ .LT. 0) IZ = IZ + 30323
C
C      IF INTEGER ARITHMETIC UP TO 5212632 IS AVAILABLE,
C      THE PRECEDING 6 STATEMENTS MAY BE REPLACED BY
C
C      IX = MOD(171 * IX, 30269)
C      IY = MOD(172 * IY, 30307)
C      IZ = MOD(170 * IZ, 30323)
C
C      ON SOME MACHINES, THIS MAY SLIGHTLY INCREASE
C      THE SPEED. THE RESULTS WILL BE IDENTICAL.
C
RANDOM = AMOD(FLOAT(IX) / 30269.0 + FLOAT(IY) / 30307.0 +
*          FLOAT(IZ) / 30323.0, 1.0)
RETURN
END

```

Remark AS R42

A Remark on AS 127. Generation of Random Orthogonal Matrices

By MARTIN A. TANNER and RONALD A. THISTED

The University of Chicago, Chicago, Illinois 60637, USA

[Received May 1980. Revised February 1982]

In testing uniformly distributed orthogonal $n \times n$ matrices generated using Heiberger's (1978) algorithm, we have found two discrepancies which affect both orthogonality and uniformity of distribution.