URAND

A UNIVERSAL RANDOM NUMBER GENERATOR

BY

MICHAEL A. MALCOLM

CLEVE B. MOLER

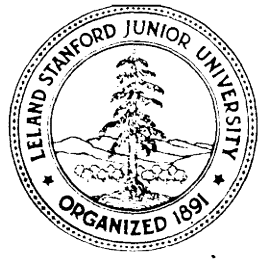
STAN-CS-73-334

JANUARY 1973

COMPUTER SCIENCE DEPARTMENT

School of Humanities and Sciences

STANFORD UNIVERSITY



URAND

A UNIVERSAL RANDOM NUMBER GENERATOR

by

Michael A. Malcolm and Cleve B. Moler

ABSTRACT

A subroutine for generating uniformly-distributed floating-point numbers in the interval [0,1) is presented in ANSI standard Fortran. The subroutine, URAND, is designed to be relatively machine independent. URAND has undergone minimal testing on various machines and is thought to work properly on any machine having binary integer number representation, integer multiplication modulo m and integer addition either modulo m or yielding at least $\log_2(m)$ significant bits, where m is some integral power of 2 .

Upon the first call of URAND, the value of m is automatically determined anti appropriate constants for a linear congruential generator are computed following the suggestions of D. E. Knuth, volume 2. URAND is guaranteed to have a full-length cycle. Readers arc invited to apply their favorite statistical tests to URAND, using any binary machine, and report the results to the authors.

The project was supported by the Office of Naval Research, Contract N00014-67-A-0112-0029.

URAND -- A Universal Random Number Generator

Michael A. Malcolm and Cleve B. Moler

The Fortran subroutine for computing random numbers which we describe in this brief report is intended for publication in a forthcoming Prentice-Hall textbook: Computer Methods for Mathematical Computations, by G. E. Forsythe, M. A. Malcolm and C. B. Moler. Other Fortran subroutines in this book (e.g. the linear equation solver, O.D.E. solver, etc.) are somewhat novel in that they are coded in a relatively machine-independent style. Among other things, this means that each subroutine, if necessary, deduces necessary parameters of the computer arithmetic system at the time it is executed. Techniques related to those given in Malcolm (1972) are used for obtaining floating-point parameters. In the same spirit we have attempted to program a relatively machine independent random number generator which we modestly call URAND which stands for "universal random number generator," and fortuitously for "uniform random number generator." To date, URAND has undergone only minimal testing on an IBM 360, CDC 6600, PDP 10 and SIGMA 7. Since it is purported to work properly on most computers in use, URAND must be tested on many more computers using a variety of statistical tests. We encourage readers to try URAND on their computers and test it using their favorite statistical tests. Feedback from our readers will be greatly appreciated. WC are particularly interested in learning of results of the "spectral test" described in Knuth, vol. 2, p. 82.

A source listing of URAND in ANSI standard Fortran is included at the end of this report. We will briefly describe the rationale which led

to some of the seemingly "random" statements in URAND.

A linear congruential sequence of integers is obtained by setting

$$Y_{n+1} = aY_n + c \text{ (modulo m)}, \quad n \geq 1 , \qquad (*)$$

on the n-th call of URAND. These are converted into floating-point numbers in the interval [0,1) and returned as the value of URAND. The resulting value of Y_{n+1} is returned through the parameter IY and should be used for the actual parameter in the subsequent call. On the first call of URAND, IY should be initialized to an arbitrary integer value.

The values of m, a and c are computed automatically upon the initial entry. The main assumption here is that the machine uses binary integer number representation and multiplication is performed modulo m where m is a power of 2 . This assumption simplifies the computation of (*) . URAND discovers the value of m/2 by testing successive powers of 2 until a multiplication by 2 produces no increase in magnitude. It is also assumed that integer addition is either modulo m , or at least $\log_2(m)$ significant bits are returned. The values of a and c are computed fallowing the advice of Knuth which he summarizes (see p. 78 and p. 155, vol. 2):

i) Pick a to have three properties:

$$a \bmod 8 = 5$$

$$m/100 < a < m - \sqrt{m}$$

The binary digits of a have no obvious pattern.

ii) Pick c as an odd integer with

$$\frac{c}{m} \approx \frac{1}{2} - \frac{1}{6}\sqrt{3} .$$

In the source code, a is called IA , and c is called IC . The random

bit pattern of a is achieved by calling DATAN(1.DO) which returns the double-precision value of $\pi/4$ which, on a binary machine, is the shifted bit pattern of π. The division by 8.DO and multiplication by m/2 is hopefully accomplished without unduly altering this pattern. The double-precision value is finally converted to an integer, multiplied by 8 and incremented by 5 to insure a mod 8 = 5. The resulting value of a is roughly $\frac{m}{8}\pi \approx \frac{m}{2}$. This satisfies the inequality constraints. The value of c is computed directly from the definition (ii). We realize that some Fortran compilers don't convert constants like 8.DO to exact floating-point representations, but this problem will probably be of little consequence.

The sequence $[Y_n]$ is guaranteed to have maximum period length m by Theorem A given in Knuth, p. 15. However, one must remember that the least significant binary digits of the Y_n will not be very random. When the Y_n are converted to floating-point numbers, the least significant digits are usually not important. To compute a random integer between 0 and k-1, one should multiply the result of URAND by k and truncate the result.

We wish to thank Fred Fritsch and Neil Goldman for testing earlier versions of URAND.

```
      FUNCTION URAND(IY)
      INTEGER IA,IC,ITWO,IY,M2,M
      DOUBLE PRECISION HALFM
      DOUBLE PRECISION DATAN,DSQRT
      DATA M2/0/,ITWO/2/
      IF (M2 .NE. 0) GO TO 20
C
c  IF FIRST ENTRY, COMPUTE MACHINE INTEGER WORD LENGTH
C
      M = 1
   10 M2 = M
      M = ITWO*M2
      IF (M .GT. M2) GO TO 10
      HALFM = M2
c
C  COMPUTE MULTIPLIER AND INCREMENT -FOR LINEAR CONGRUENTIAL METHOD
C
      IA = 8*IDINT(HALFM*DATAN(1.D0)/8.D0) + 5
      IC = 2*IDINT(HALFM*(0.5D0-DSQRT(3.D0)/6.D0)) + 1
C
C  S IS THE SCALE FACTOR FOR CONVERTING TO FLOATING POINT
c
      S = 0.5/HALFM
c
C  COMPUTE NEXT RANDOM NUMBER
C
   20 IY = IY*IA + IC
C
c  THE FOLLOWING STATEMENT IS FOR COMPUTERS WHERE THE
C  WORD LENGTH FOR ADDITION IS GREATER THAN FOR MULTIPLICATION
c
      IF (IY/2 .GT. M2) IY = (IY - M2) - M2
c
C  THE FOLLOWING STATEMENT IS FOR COMPUTERS WHERE INTEGER
C  OVERFLOW AFFECTS THE SIGN BIT
C
      IF (IY .LT. 0) IY = (IY + M2) + M2
      URAND = FLOAT(IY)*S
      RETURN
      END
```

REFERENCES

Knuth, D. E. (1969), "Seminumerical algorithms," The Art of Computer Programming. Vol. 2, Reading, Mass.: Addison Wesley.

Malcolm, M. A. (1972), "Algorithms to reveal properties of floating-point arithmetic," Comm. ACM, vol. 15, no. 11, November, 949-951.