A Portable FORTH Random Number Generator

William T. Doyle

Physics Department
Dartmouth College
Hanover, NH 03755

Abstract

A portable high-level FORTH random number generator is described. It is based upon Knuth's FORTRAN function IRN55 and uses the tabular method of Mitchell and Moore. The FORTH word and the FORTRAN subroutine yield identical long sequences of random double numbers.

Introduction

Sequences of random numbers are needed for many important applications, including numerical analysis, simulation, sampling, modelling, program testing, and, of course, games. For all of these, a reliable random number generator is indispensible. A good random number generator should be fast, compact, and portable. It must have a very long repeat period and produce sequences of numbers that closely approximate the statistical behavior of a physical random variable. An excellent discussion of random number generators may be found in Knuth [1].

Random Number Algorithms

Linear Congruential Method

The popular linear congruential method, generates each new number X as a linear function of the preceding number,

$$X_{n+1} = (aX_n+c) \bmod m$$

where a and c are constants, chosen to produce long sequences of 'good' numbers, and m is usually the word length of the machine. The linear congruential method has been widely used and thoroughly studied. However, it can be slow in microcomputers and, more seriously, it performs poorly in sequential tests. This means that if one takes sequences of two or three successive random numbers and plots them as points in two or three dimensional space, the points tend to lie on lines or planes, in an obviously non-random way. This tendency is conspicuous in single-number versions of the linear congruential method, even when the parameters are well-chosen. Single-number linear congruential generators should be used only for short sequences of random numbers, where sequential correlations are not of concern.

Tabular Methods

An alternative method particularly well-adapted to micro-computers was introduced by Mitchell and Moore in 1958 and has been widely used and tested since its introduction (Ref 1, p. 26). In this 'additive' method an array of random numbers is first created. Then each random variable is formed as the sum of two widely separated values in the array. As each new value is found, the lower array element is reset to the new value of the variable. The additive method avoids the sequential

problems of the linear congruential method and has the further advantage of yielding the longest sequences of numbers yet produced while not requiring any time-consuming multiplication or division. We shall use a variant of the Mitchell and Moore algorithm called the 'subtractive' method (Ref 1, p. 171).

In the subtractive method a table of random numbers is first created. Successive random numbers are obtained by calling the array elements cyclically. As each element is used, the array index is incremented and the element itself is updated by subtracting from it the current value of a carefully chosen cyclically higher element in the array. Whenever a negative number occurs, a large positive constant is added to it to force the numbers to be positive. This constant also determines the range of possible numbers.

Knuth describes a portable high-level version of the subtractive random number generator that may be implemented in any higher level language. Since it makes use only of integer arithmetic, it lends itself particularly well to high-level FORTH. We follow Knuth in choosing an array size of 55 and an index offset of 31, so successive random numbers are obtained from the array according to the rule

$$X_{n+55} = X_n - X_{n+31}$$

Where X_{n+55} is the new value of X_n. Other index pairs leading to the maximal length sequences are tabulated by Knuth, but this pair permits us to take advantage of the full range of FORTH double numbers and to compare our values directly with those given by Knuth's program using the same seed and the same number range.

FORTH Implementation

The FORTH implementation is shown in screens 111 through 113. It is written entirely in high-level FORTH-79, so it is portable. In screen 111 we first set up a 55 element double number array RND to hold our random variables, and a counter variable to keep track of the cyclic counter. CYCLE and STEP create the cyclic counter TALLY for the index range 1-55.

RND.INIT is based on Knuth's initialization subroutine IN55. It initializes the entire array from a single seed by computing a Fibonacci-like sequence and then scattering the values around to improve the initial randomness of the array. This method of initialization permits a direct comparison with results obtained with the FORTRAN routines. Simpler methods could also be used, for example, by using a simple single-number random number generator, if one is already at hand.

The word RANDOM on screen 112 does the work. It is equivalent to Knuth's function IRN55. To be consistent with the size of FORTH signed double numbers, we use a range of 0-1000000000 here. Rather than following Knuth and exhausting the entire array before resetting it, RND is continually updated as the numbers are used. This is no more time-consuming, and it avoids an extra delay at every 55th call.

If RND.INIT (or any other non-random seeding) is used, the generator must be 'warmed up' by resetting the array a number of times before using the numbers. The word WARMUP cycles the array four times to randomize the elements. This corresponds to the warmup procedure in Knuth's FORTRAN routine. SETUP resets the cyclic counter TALLY, initializes the double number array RND from the single seed SEED, and calls WARMUP.

To use the generator LOAD screens 111-113 and call SETUP once. Subsequent calls to RANDOM will return random double numbers in the range of 0-1000000000. The sequence of numbers is, of course, identical to those obtained with the same algorithm in other high-level languages, so a direct comparison is possible when desired. The FORTH generator has been checked against versions in FORTRAN, BASIC, and 6502 machine language. The high-level FORTH version is comparable in speed with the 6502 machine language version on an Apple-II.

Conclusion

Many important applications require reliable long sequences of random numbers with statistical properties approximating those of a physical random variable. The popular linear congruential method performs poorly on sequential tests. Tabular methods, on the other hand, suppress sequential correlations while yielding the longest known repeat periods. The high-level Forth-79 word RANDOM (SCR #112) is based upon Mitchell and Moore's tabular algorithm. It returns FORTH double-numbers in the range 0-1000000000. When seeded with the word RND.INIT (SCR #112) the sequences produced are identical to those produced by Knuth's FORTRAN subroutine with the same seed and number range. With a repetition period of 2^{55}, sequences produced by RANDOM should be useful in many applications.

References

1. *The Art of Computer Programming, Seminumerical Algorithms*, Knuth, D. E., Vol. 2, p. 1-177, Addison-Wesley (1981).

Manuscript received July 1983.

Dr. Doyle received his B.A. from Brown University, and his M.A. and PhD. from Yale University. He is a professor of Physics and Astronomy at the Wilder Laboratory, Dartmouth College, Hanover, New Hampshire. He became interested in using Forth for laboratory applications, and employs it for testbenches in the physics laboratory.

```
SCR# 111
( RANDOM NUMBER GENERATOR  1                    FORTH-79              WTD )

: DARRAY                                    ( CREATES DBL NR ARRAY )
    CREATE 4 * ALLOT DOES> SWAP
    1- 4 * + ;
55 DARRAY RND                               ( 55 RANDOM NRS IN RND )

VARIABLE TALLY 0 TALLY !                    ( SET UP TALLY COUNTER )
2VARIABLE SEED 314159296. SEED 2!           ( SET SEED )

: CYCLE                    ( N1 A --- N2 )  ( SETS UP CYCLIC INDEX )
    DUP >R @ 1+ DUP ROT > IF DROP           ( GIVEN LIMIT, ADDRESS )
    1 THEN DUP R> ! ;                       ( LEAVES NEXT VALUE )

: STEP                       ( A --- N+1 )  ( CYCLIC INDEX MOD 55 )
    55 TALLY CYCLE ;                                              -->


SCR# 112
( RANDOM NUMBER GENERATOR  2                                      WTD )

: RND.INIT                          ( --- )  ( INITIALIZE ARRAY )
    0 TALLY ! SEED 2@ 2DUP 55 RND 2!        ( ZERO TALLY SET SEED )
    1. 2SWAP 55 1 DO  2OVER 2DUP  21        ( SCATTER VALUES BY )
    I * 55 MOD RND 2! D- DUP  0 < IF        ( SUBTRACTION )
    1000000000. D+  THEN 2SWAP LOOP         ( MAKE THEM POSITIVE )
    2DROP 2DROP ;                           ( CLEAN UP THE STACK )

: RANDOM                            ( --- D )  ( NEXT RANDOM NUMBER )
    STEP DUP >R RND 2@ 2DUP R@ DUP          ( SUBTRACT I+31 TERM )
    25 < IF 31 + ELSE 24 - THEN RND         ( FROM I TERM [CYCLIC] )
    2@ D- DUP 0 < IF 1000000000. D+         ( MAKE RESULT POSITIVE )
    THEN R> RND 2! ;

                                                                  -->


SCR# 113
( RANDOM NUMBER GENERATOR  3                                      WTD )

: WARMUP                            ( --- )
    221 1 DO RANDOM 2DROP LOOP ;            ( INIT TALLY & RND )

: SETUP                             ( --- )
    0 TALLY ! RND.INIT WARMUP ;             ( INIT AND RANDOMIZE )


( TO USE:  RUN SETUP. THEN CALL RANDOM DOUBLE NUMBERS USING RANDOM. )
( WITH THE SAME SEED AND RANGE [1E9.], RANDOM RETURNS THE SAME )
( NUMBERS AS KNUTH'S FORTRAN ROUTINE IRN55[1A]. )
```