GENERATING RANDOM NUMBERS IN APL

by

Thomas N. Herzog

Loyola College of Baltimore

APL is a vector-matrix based computer programming language which is a powerful tool for solving actuarial problems. One such class of problems is Monte Carlo or simulation studies. In the present paper we suggest improvements in the APL primitive function roll (denoted by ?), which is a uniform pseudorandom number generator. We also discuss procedures for generating pseudorandom normal deviates in APL.

The paper consists of two parts. The first part deals with uniform pseudorandom numbers, the second with pseudorandom normal deviates. Because a large portion of ARCH readers are probably not familiar with APL, we have not included much APL notation or coding in this paper. Such material is, however, to be found in Freiden and Herzog [1979] and Herzog [1981].

1. Uniform Pseudorandom Numbers

1.1 The APL Uniform Random Number Generator

The APL primitive function roll (denoted by ?) is a multiplicative congruential generator of the form

$x_{n+1} = 7^5 \cdot x_n \pmod{2^{31} - 1}$

1.2 Inadequacies of the Roll Function

Marsaglia [1968] in a paper entitled "Random numbers fall mainly in the planes" has shown that for many applications the use of multiplicative congruential generators, such as the roll operator, is inappropriate. Specifically, "the principal defect is that certain simple functions of n-tuples of uniform random numbers do not have the distribution that probability theory predicts." "The problem lies in the 'crystalline' nature of multiplicative generators; if n-tuples

$(u_1, u_2, \ldots, u_n)$, $(u_2, u_3, \ldots, u_{n+1})$,...of uniform variates produced by the generator are viewed as points in the unit cube of n-dimensions, then all the points (n-tuples) will be found to lie in a relatively small number of parallel hyperplanes."

Knuth [1980] reports two schemes for circumventing this difficulty. The first such scheme, suggested by MacLaren and Marsaglia [1965], entails the use of one random sequence to permute the elements of another. The other scheme, suggested by Bays and Durham [1976], can give surprisingly better results even though it requires only one random sequence instead of two. Specifically, the Bays-Durham scheme results in a lengthened period and a decrease in local nonrandomness; in addition, this latter scheme requires less computer time than the MacLaren-Marsaglia scheme provided the procedure is carried out in assembly language or in a computer language such as BASIC or FORTRAN which is not a vector-matrix oriented language. Unfortunately, the Bays-Durham scheme seems quite inefficient to perform in APL, at least in terms of computer time.

For APL, the following widely used scheme appears to be more efficient in this respect. To generate N pseudorandom integers uniformly distributed between 1 and M, inclusive, (with M no more than the underlying modulus $2^{31}-1$) use the APL roll operator to first generate N pseudorandom numbers over the desired range and then to reorder the sequence using a random permutation of the integers from 1 to N. This procedure keeps the length of the period of the sequence at $2^{31}-2$.

The recommended procedure is in the spirit of MacLaren and Marsaglia [1965] as well as a procedure which Learmonth and Lewis [1973] attribute to John W. Tukey. The Tukey scheme is to generate blocks of, say 1024, pseudorandom numbers and then shuffle the elements of each block according to some random permutation.

1.3 Conclusions and Recommendations

1. APL'ers using the roll operator to generate pseudorandom numbers should be well-aware of the serious defect identified by Marsaglia [1968], and should, therefore, probably consider "shuffling" all random sequences produced by the roll operator.

2. A new APL primitive ?̈ (? overstruck with ¨ , the dieresis) should be introduced to return a "shuffled" sequence of uniform pseudorandom numbers. Such a primitive would implement an optimal algorithm in the interpreter. It would also be a constant reminder to all APL'ers that the roll operator, if not modified, may produce bad, but unrecognized, results in simulation studies. Alternatively, the roll operator could be redefined as a "better" random number generator such as in McGill University's Random Number Package SUPER DUPER (Marsaglia, Ananthanarayanan, and Paul [1976]).

2. Generating Normal Random Deviates in APL

2.1 Three Methods

Knuth [1969] describes three methods for generating pseudorandom normal deviates. One method, is said to be considerably faster than the others. In fact, Knuth regards its relative speed sufficient to compensate for its larger memory requirement. We will show that this analysis is not valid when the basic machine instruction set is unavailable to programmers, as in APL.

The three methods treated by Knuth [1969] are the polar method, Teichroew's method, and Marsaglia's rectangle-wedge-tail method. Each of these methods requires as input at least one pseudorandom number uniformly distributed over the interval (0,1).

In the polar method, two uniform deviates are required as input. This requirement does not, however, decrease the relative speed of the polar method because two independent normal deviates are generated. The polar method may be described as follows:

First, generate an ordered pair (u_1,u_2) where u_1 and u_2 are independent random variables uniformly distributed between ¯1 and 1. If (u_1,u_2) is in the interior of the unit circle centered at the origin (of a two-dimensional Cartesian coordinate system), form x_1 and x_2 as:

$$x_1 = u_1\sqrt{\frac{-2 \ln R}{R}} \quad \text{and} \quad x_2 = u_2\sqrt{\frac{-2 \ln R}{R}}$$

where the radius $R = u_1^2 + u_2^2$.

x_1 and x_2 are the desired independent normally distributed random variables. If (u_1,u_2) is outside the unit circle centered at the origin, generate another ordered pair and repeat the above procedure.

In an APL implementation of Knuth's "Algorithm P" for generating pseudorandom normal deviates by the polar method (see

Freiden and Herzog [1979]) the primitive operations natural logarithm and raise to an arbitrary power can be used. This makes the polar method relatively faster in APL.

Teichroew's method also takes advantage of an APL primitive operator since it is a polynomial approximation of a normal deviate. Teichroew's method may be described as follows: Let U be a vector of 12 independent random variables uniformly distributed between 0 and 1, and define a new random variable R by

$$R = \frac{(\sum_{i=1}^{12} U_i)-6}{4}$$

Then if A is the 10-element vector A = (0, 3.949846238, 0, 0.252408784, 0, 0.076542912, 0, 0.008355968, 0, 0.029899776), it turns out that the polynomial

$$H(R) = \sum_{i=0}^{9} A_i R^i$$

approximates a normal deviate. This polynomial is an approximation of $F^{-1}(G(R))$ where F( ) is the standardized normal distribution function and G( ) is the exact distribution of R. Since each element of the vector U is assumed to have a uniform distribution over the interval (0,1), R is approximately normally distributed with mean 0 and standard deviation 1/4.

Marsaglia's method, which may be applied to any distribution function, is quite different from the other methods. Marsaglia suggests rewriting the desired distribution function F as a linear combination of n distribution functions $F_1, F_2, \ldots, F_n$. Thus, we may write

$$F(x) = \sum_{i=1}^{n} P_i F_i(x)$$

where

$$0 \leq P_i \leq 1 \quad \text{and} \quad \sum_{i=1}^{n} P_i = 1$$

Each F_i is to be computed only $100P_i$ percent of the time, on the average. Since most of the functions F_i are trivial modifications of the uniform distribution, if the P_i and F_i are chosen judiciously, the calculation of $F(x)$ will be trivial most of the time.

The implementation of Marsaglia's method rests on the decomposition of a random bit string which is equivalent to the pseudorandom number

$U = 0.b_0\ b_1 \ldots b_t$

The first bit b_0 determines the sign of the result. The bits b_1 through b_8 are used to select the appropriate F_i as follows:

(a) If $0 \leq b_1b_2b_3b_4 < 10$, then set x to

$A[b_1b_2b_3b_4] + .00\ b_5b_6 \ldots b_t$

(b) If $40 \leq b_1b_2b_3b_4b_5b_6 < 52$, then set x to

$B[b_1b_2b_3b_4b_5b_6] + .00\ b_7b_8 \ldots b_t$

(c) If $208 \leq b_1b_2 \ldots b_8 < 225$, then set x to

$C[b_1b_2 \ldots b_8] + .00\ b_9b_{10} \ldots b_t$

where A, B, and C are stored tables of values.

(d) Otherwise, use a more complicated technique, which we will not describe here.

Note that 10 times out of 16, or 62.5 percent of the time, on the average, only step (a) is required, whereas step (d) is required only 12.5 percent of the time, on the average.

The decomposition of U into bit strings of length 4, 6, or 8 is best done in machine language using logical "AND" and "SHIFT" operations within high-speed registers. The indexing of the tables A, B, and C is also fast in machine language. In APL, however, these operations must be simulated using multiplication and division by the appropriate powers of two. Our APL implementation of Marsaglia's method was limited to the first step (step (a) above).

2.2 Timing the Methods

We are interested only* in execution speed and we expected the algorithm making best use of APL primitives to be the fastest. Obviously, execution speed depends on the APL implementation and the instruction set of the host machine. Therefore, we have run timing tests on three different computer systems. Of course, it is not necessary to test the complete Marsaglia algorithm if the first step alone is slower than the complete algorithms of the other methods. In the following table, execution time is an average obtained by generating ten sets of 100 normal random numbers and is measured relative to the execution time of the polar method.

The three computer systems were

- IBM 370 (The Computer Company)
- IBM 5100
- Burroughs 7700 (Data Resources, Inc.)

*According to Knuth [1969, page 113], both the polar and rectangle-wedge-tail methods "give essentially perfect accuracy." However, "Teichroew's method is only approximate, although in most applications its accuracy (an error bounded by 2×10^{-4} when $|R| \leq 1$) is quite satisfactory." In the second edition of his book, Knuth [1980] omits entirely a discussion of Teichroew's method.

Relative Execution Times

Method	System 1	2	3
Polar	1.00	1.00	1.00
Teichroew	1.32	1.19	1.16
Marsaglia	1.64	1.64	1.15

Although the timings are quite different, the polar method requires the smallest amount of execution time regardless of which computer system is used.

2.3 Historical Remarks

The polar method was originally suggested by Box and Muller [1958]. The form of the polar method used in this paper is due to Marsaglia [1962].

The original reference for Marsaglia's rectangle-wedge-tail method is Marsaglia, MacLaren and Bray [1964]. An improved version of this scheme is presented in Marsaglia, Ananthanarayanan, and Paul [1976].

2.4 Further Thoughts

So far, there have been no remarks directed at specific actuarial issues. So I thought that before closing I would at least make one observation. For problems in which the experience of a portfolio of insureds is to be simulated, the experience of the underlying mortality table should be assumed to have arisen from a "random process." In particular, the mortality table itself should usually be considered to be the mean of a probability distribution. Consequently, unless the amount of experience underlying all age groups represented in the mortality table is exceedingly large (so that all variances may be considered to be negligible), the distribution of the mortality table as well as the portfolio under consideration should both be simulated in such problems.

BIBLIOGRAPHY

Bays, C. and Durham, S. D., "Improving a poor random number generator," ACM Transactions on Mathematical Software, Vol. 2, No. 1, March 1976, pages 59-64.

Box, G. E. P. and Muller, M. E., "A Note on the Generation of Random Normal Deviates," Annals of Statistics, Vol. 29, 1958, pages 610-611.

Freiden, A. and Herzog, T. N., "Generating normal random deviates in APL," APL QUOTE QUAD, Vol. 9, No. 3, March 1979, pages 49-51.

Gebhardt, F., "Generating pseudo-random numbers by shuffling a Fibonacci sequence," Mathematics of Computation, Vol. 21, No. 100, October 1967, pages 708-709.

Herzog, T. N., Generating Uniform Pseudorandom Numbers in APL, Actuarial Division, HUD, 1981.

Hoaglin, D. C., Theoretical Properties of Congruential Random Number Generators: An Empirical View, Department of Statistics, Harvard University, November 1976.

Knuth, D. E., The Art of Computer Programming, Vol.2, 2nd edition, Addison-Wesley, 1980. (The first edition was published in 1969.)

Learmonth, G. P. and Lewis, P. A. W., "Statistical tests of some widely used and recently proposed random number generators," in Proceedings of Computer Science and Statistics: 7th Annual Symposium on the Interface, W. J. Kennedy, E., Ames, Iowa: Iowa State University Press, 1973.

MacLaren, M. D. and Marsaglia, G., "Uniform Random Number Generators," Journal of the Association for Computing Machinery, Vol. 12, No. 1, January 1965, pages 83-89.

Marsaglia, G., "Random Variables and Computers," Transactions of the Third Prague Conference on Information Theory, Statistical Decision Functions, Random Processes, June 1962, Prague, Publishing House of Czechoslovak Academy of Sciences, 1964, pages 499-512.

Marsaglia, G., "Random numbers fall mainly in the planes," Proceedings of the National Academy of Science, Vol. 60, No. 5, September 1968, pages 25-28.

Marsaglia, G., Ananthanarayanan, K., and Paul, N. J., "Improvement on Fast Methods for Generating Normal Random Variables," Information Processing Letters, Vol. 5, pages 27-30.

Marsaglia, G., MacLaren, M. P., and Bray, T. A., "A Fast Procedure for Generating Normal Random Variables," Communications of the ACM, Vol. 7, No. 1, January 1964, pages 4-10.

Discussion of Preceding Paper

AL CHRISTIANS: With the APL roll function, do you know the number of planes in hyperspace that are actually involved?

TOM HERZOG: No. I have, unfortunately, not looked into this. Of course, this would depend on the dimension of the hyperspace under consideration, and the latter would, in turn, depend upon your specific application.

VOICE FROM FLOOR: It's not clear to me why you want to introduce a new operator. Why not simply improve the old one?

TOM HERZOG: I agree with your suggestion and made that recommendation myself in my article in the December 1981 QUOTE QUAD.

KATHRYN PLANTE: Do your results also apply to congruential generators of the form

$$x_{n+1} = ax_n + b \pmod{c}$$

TOM HERZOG: Yes. On page 28 of his National Academy of Sciences paper Marsaglia states that similar results can be established for this type of generator.