Behavior Research Methods & Instrumentation
1977, Vol. 9 (6), 559

FORTRAN subroutines for random sampling without replacement

BERT F. GREEN, JR.
Johns Hopkins University, Baltimore, Maryland 21218

Robertson (1977) recently reported an algorithm and BASIC program for selecting K things from N symbols without replacement. His algorithm was an improvement on that of Schierer (1976). Actually, algorithms reported by Green (1963) and Knuth (1969) are considerably shorter and more efficient than either. All of these programs except that of Knuth solve the more general problem of producing a random permutation of K symbols out of N. The user may of course ignore the fact that the symbols are randomly ordered as well as randomly selected, but the random ordering is a natural by-product of the Green, Robertson, and Schierer algorithms. Since the symbols themselves do not enter the computation, they can be any numerical or other binary codes, such as ASCII characters (provided that each character is in a separate word).

A FORTRAN subroutine for Green's (1963) algorithm is shown in Table 1. The input is an array, X, of the N symbols, X(1), X(2), . . . , X(N), plus the two parameters, N and K. The output is the rearranged array, X, in which the first K symbols, X(1) . . . X(K), are the required permutation. The process is a loop that is traversed K times. On the Ith cycle, a symbol is selected at random from the (N − I + 1) symbols X(I) . . . X(N). The selected symbol is exchanged with X(I). Since the subroutine is so simple, some users might prefer to use it in-line.

If the original order of symbols in the array is arbitrary, or if the original order can be recovered when needed, then the subroutine can be used repeatedly to generate many permutations, without restoring X. The output X from one CALL can be the input X on the next CALL with no loss. All of the symbols are still in X, though they are partially permuted. That is not true of the Robertson (1977) and the Schierer (1976) algorithms. For those, X must be restored before the process can be repeated.

Table 1
FORTRAN Subroutine for a Random Permutation of K Out of N Objects (Green, 1963)

```
05000        SUBROUTINE PERMUT (X,N,K)
05100        DIMENSION X(N)
05200        DO 50 J=1,K
05300        I=INT ((FLOAT (N-J+1)) *RAN (0))+J
05400        T=X(J)
05500        X(J)=X(I)
05600    50  X(I)=T
05700        RETURN
05800        END
```

Table 2
FORTRAN Subroutine for a Random Selection of K Objects Out of N (Knuth, 1969)

```
07500        SUBROUTINE RANSAM (X,A,N,K)
07600        DIMENSION X(N),A(K)
07700        M=0
07800        DO 50 J=1,N
07900        L=INT ((FLOAT (N-J+1)) *RAN (0))+1
08000        IF (L.GT. (K-M)) GO TO 50
08100        M=M+1
08200        A(M) = X(J)
08300        IF (M.GE.K) GO TO 99
08400    50  CONTINUE
08500    99  RETURN
08600        END
```

Note—X is an input vector of values; it is unaltered by the subroutine. A is an output vector of selected values.

If the original array cannot be modified, and if only a random selection of K out of N is needed, without random permutation, Knuth's (1969) algorithm is efficient. It uses a single pass through the array, which is convenient if the array is on tape. In the routine, the probability of selecting the Ith object in the array is adjusted depending on how many objects have already been selected. Knuth gives a carefully detailed analysis of the algorithm, which establishes the propriety of this suspicious step. For reference, a FORTRAN subroutine is shown in Table 2. A handy feature of the Knuth algorithm is that the output symbols in A are in the same order as the input items in X.

Each routine was timed on a DEC system-1090 as it generated 1,000 selections of 50 out of 100. The results were 29.5 sec for Green (1963), 57.8 sec for Knuth (1969), and 137.2 sec for a FORTRAN equivalent of Robertson (1977). Of course, if each selection is to be printed out, or used in some other slow-moving event, such as a trial in a computer-controlled psychological experiment, efficiency is of little importance. However, if the routine is part of a Monte Carlo study, or a simulation, efficiency is vital.

REFERENCES

GREEN, B. F. *Digital computers in research.* New York: McGraw-Hill, 1963.

KNUTH, D. *The art of computer programming. Vol II. Seminumerical Algorithms.* Reading, Mass: Addison-Wesley, 1969.

ROBERTSON, S. A. A BASIC program to produce random samples without replacement. *Behavior Research Methods & Instrumentation*, 1977, **9**, 363-364.

SCHIERER, C. J. Routine to provide a random order for counterbalanced variables. *Behavior Research Methods & Instrumentation*, 1976, **8**, 468.

(Accepted for publication October 3, 1977.)