SESSION IX

MICROCOMPUTER PROGRAMMING TECHNIQUES

William L. Palya, *Presider*

Repairing the Microsoft BASIC RND function

DANIEL S. LORDAHL

Brock University, St. Catharines, Ontario, Canada

Difficulties with the Microsoft BASIC function for generating random numbers (RND) are cited and new problems reported. Fixes are suggested that allow use of some of the speed of the original function and that produce greatly improved pseudorandom numbers. A published generator is combined with the RND function as a compromise solution.

Whitney (1984) pointed out that the pseudorandom number generator supplied by Microsoft Corporation in the BASIC language for the IBM PC and similar models has marked spikes in its frequency spectrum, indicating significant subcycles. Modianos, Scott, and Cornwell (1984) also studied generators on microcomputers and found the IBM PC version faulty. This faultiness became evident to me while simulating a contingency chi-square problem. Although the raw frequency counts on a one-way classification of numbers from the RND function pass a basic chi-square test for all of the partitions I have tried, the first-order relations between numbers is quite non-random.

Problem

For example, Table 1 summarizes some tests done with the RND function. The random numbers were converted to integers in the range indicated in the column labeled K in Table 1. These values were summed into a $K \times K$ matrix, where row entries represent the outcome of some number and the column entries show the outcome of the succeeding number in the RND sequence. In each of the tests shown here, a chi-square test was performed using the row totals obtained to compute the expected frequencies for each cell.

The resulting values are based on $K(K-1)$ degrees of freedom. In *all* cases tested, including others not shown here, the total chi-square was smaller than the expected mean. Converting the values to the normal curve approximations for cases having large degrees of freedom, the resulting z scores and corresponding probability values consistently show the frequencies to be too flat.

For example, 400 replications were used with a 50×50 matrix with samples of size 50,000, yielding average expected cell frequencies of 20. The total chi-square obtained was 975,479, whereas the total degrees of freedom (and the mean expected) was 980,000. These data result in a normal z score of -3.23 . Although the expected number of samples in the upper 10% of the chi-square distribution was 40 (out of 400), the actual number of such samples was 27. Such outcomes were common in these tests. Since my interests were in simulating contingency chi-square tests, the vastly reduced probabilities of "significant" outcomes by chance rendered the RND function useless for serious work.

Solution 1

Regularly skipping one, two, or even three numbers in the sequence does not completely eliminate the sequential dependencies in the obtained random integers, again using only the first two or three digits. Also, dropping a digit or two from the front of the random decimal before converting to an integer does not give satisfactory results.

A more complex method was employed with some success. Following initial seeding, which will be discussed later, one RND is used to "randomly" select an integer 1, 2, or 3. Then the corresponding 1, 2, or 3 RNDs are added to the last used result. Any integer part is removed from the sum, and the result is the next decimal value for use. This method will be called RAN and is performed as follows:

```
IL = INT(RND*3) + 1
FOR J = 1 TO IL
  R = R + RND
NEXT J
RAN = R - INT(R)
```

Address correspondence to Daniel S. Lordahl, Department of Psychology, Brock University, St. Catharines, ON L2S 3A1, Canada.

If speed is not a crucial factor, the value of IL, the number of numbers summed, may also be based on a summation with the previous number used. In each case, the use of the carryover value from the last iteration produces an altered sequence even in the rare event of a duplicate reseeding, if initial values for the carryovers are chosen separately.

Table 2 has some characteristic results with the generation method shown above. The same procedures were used as described earlier. Note especially that a number of positive z scores were obtained, indicating that the total chi-square value fell above the expected mean. Moreover, the distribution of "significant" individual chi-squares appears reasonably close to chance levels.

Solution 2

Recently, Wichmann and Hill (1987) published a random number generator for 16-bit machines, which combines three different congruential generators. Their method has the great advantage of a cycle length of $6.95E+12$. Its disadvantage is time.

I have explored another method, which combines the Wichmann and Hill (1987) generators with the RND function in BASIC. Specifically, a number from *one* of the Wichmann and Hill generators is chosen on a rotating basis and is combined with an RND number. This method (called COMB) also has an extremely long cycle, but one not exactly knowable without source code information on RND. The strict cycle of the RND sequence is over 16,800,000, the exact value depending upon the seed used

to start the sequence. Thus the total cycle of the COMB generator would be a minimum of $3 \times 6.95E+12$, attaining that value if and only if the RND repeats in synchrony with the Wichmann and Hill team.

The cost of using any fix is time. However, for serious simulation work, it is highly desirable to use compiled programs. In this work, I used version 2 of the Quick-BASIC compiler by Microsoft on an AT clone at 8 MHz. Approximate times to generate 1,000 random numbers were found in BASICA and in compiled BASIC for each of the four generation methods mentioned here. These times (in seconds), with the compiled times listed second, were: for the RND, .60 and .075; for RAN, 6.07 and .75; for COMB, 7.73 and .83; and for the full Wichmann and Hill (1987), 14.31 and 1.93.

Evaluation

Some standard tests were conducted on the RND, RAN, and COMB generators (Knuth, 1973). The first of these was on the maximum found in a sequence of random numbers; the sequences were of length 100. Results were tabulated in 25 equally likely classes. The second test was a serial test; the leading digit was used to form 10 possible values from each of three consecutive numbers, and the ordered triplets were then tabulated. Finally, permutations of length 5 were studied, with all 120 possible rank orderings forming the categories.

The test statistic was a chi-square in each case. These chi-square values were examined separately in each of 1,000 replications (samples), and the frequencies of chi-

Table 1
Results from the RND BASIC Generator Without Changes

K	Exp.	#	χ^2	df	z	$p(x > z)$	.10	.05	.01
100	10	300*	2,945,652	2,970,000	-10.01	1.0000	5	0	0
100	5	100	986,638	990,000	-2.39	.9916	3	1	0
75	10	200	1,105,690	1,110,000	-2.90	.9981	13	5	0
50	20	400	975,479	980,000	-3.23	.9994	27	11	1
35	10	1100*	1,301,911	1,309,000	-4.39	1.0000	84	38	8
25	20	500	297,669	300,000	-3.02	.9987	33	13	0
25	100	200	118,780	120,000	-2.49	.9937	12	4	1
20	25	1000*	378,667	380,000	-1.53	.9370	84	40	9
20	100	200	75,446	76,000	-1.44	.9247	11	1	1
Totals		4000	8,185,932	8,235,000	-12.11	1.0000	272	113	20

*Combined from more than one run. Exp. is expected frequency. # is replications.

Table 2
Results from the Modified RND BASIC Generator (RAN)

K	Exp.	#	χ^2	df	z	$p(x > z)$	.10	.05	.01
100	10	300*	2,972,456	2,970,000	1.01	.1563	35	18	4
75	10	500*	2,773,195	2,775,000	-.77	.7794	46	27	8
75	20	200	1,111,917	1,110,000	1.29	.0986	27	16	2
50	20	400	979,527	980,000	-.34	.6323	39	21	2
35	10	500	594,454	595,000	-.50	.6915	50	28	6
30	20	500	435,989	435,000	1.06	.1445	62	28	4
25	20	500*	299,149	300,000	-1.10	.8644	44	24	6
20	25	500	189,645	190,000	-.57	.7173	47	24	4
Totals		3400	9,356,332	9,355,000	+.306	.3787	350	186	36

*Combined from more than one run. Exp. is expected frequency. # is replications.

Table 3
Summary of Tests on Three Generators

Test	Function	Classes	<i>p</i>	.10/.90	.05/.95	.01/.99
Max. of	RND	25	.375	97/92	47/54	10/14
100	RAN	25	.880	88/127	58/64	6/13
EV=10	COMB	25	.880	96/113	51/49	9/17
Serial-	RND	1000	.947	93/105	42/53	11/9
length of 3	RAN	1000	.438	101/97	53/54	11/8
EV=5	COMB	1000	.304	103/99	58/49	10/11
Permutations	RND	120	.033	116/75	47/48	6/7
of length 5	RAN	120	.427	88/99	45/49	11/9
EV=10	COMB	120	.730	98/94	54/41	16/9

Note—All tests were replicated 1,000 times. EV is expected class value.

squares in each tail were tallied. The total chi-square across all samples was also found as a type of summary. Table 3 contains the data from these tests. The RND data is given mainly for comparison, since the routine is considered discredited.

Both RAN and COMB functions appear to pass these tests. Since the RAN function is based exclusively on RND, its cycle length is limited. The full 16,000,000+ cycle of RND may not even be available since RAN uses an average of three numbers per output value. Moreover, the less-than-random properties of the library generator

are not well enough understood to fully trust a system based solely on the RND. Finally, the increase in processing time in COMB over RAN is small. I recommend the COMB procedure, which is shown in Listing 1.

Pseudorandom seeding of the generator requires a bit of planning. It is best accomplished by combining a number of variables. For instance, the TIMER function in BASIC gives the number of seconds elapsed since midnight. On my system, this value is given to two decimals of accuracy. Use of the decimal portion *and* the integer can result in seeding that appears quite irregular and unpredictable (i.e., random). The COMB generator has the advantage of four separate seedings, one for each of the Wichmann and Hill (1987) generators and one for the RND function. A simplified version of a seeding procedure is also given in Listing 1.

For improved speed, an assembly language subroutine should be developed for the chosen method of generation. Even compiled programs are somewhat slower, comparatively, since the compiler has no intelligence about statements that follow, and it allows for all possibilities. These gross inefficiencies may be avoided in assembly code.

REFERENCES

- KNUTH, D. E. (1973). *The art of computer programming: Volume 2. Seminumerical algorithms*. Reading, MA: Addison-Wesley.
- MODIANOS, D. T., SCOTT, R. C., & CORNWELL, L. W. (1984). Random number generation on microcomputers. *Interfaces*, 14, 81-87.
- WHITNEY, A. (1984). Generating and testing pseudorandom numbers. *Byte*, 9, 11, 128.
- WICHMANN, B. A., & HILL, I. D. (1987). Building a random-number generator. *Byte*, 3, 127-128.

Listing 1
Combined Congruential Generators and RND Generator

```

101 DEFINT I-N
102 X=TIMER: XI=INT(X): X=X-XI
104 RANDOMIZE TIMER
110 X=RND+X: X=X-INT(X): PRINT X
120 X=INT(X*65536)+1-32769
130 RANDOMIZE X
140 IX=RND*30269: IY=RND*30307: IZ=RND*30323
170 'Wichmann and Hill generators combined with library RND
175 ITT=(ITT+1) MOD 3+1
180 ON ITT GOTO 181, 202, 303
181 IX = 171*(IX MOD 177) - 2*INT(IX/177)
182 IF IX < 0 THEN IX=IX+ 30269
183 R= IX/30269: GOTO 404
202 IY = 172*(IY MOD 176) - 35*INT(IY/176)
204 IF IY < 0 THEN IY=IY+ 30307
206 R = IY/30307: GOTO 404
303 IZ = 170*(IZ MOD 178) - 63*INT(IZ/178)
305 IF IZ < 0 THEN IZ=IZ+ 30323
307 R = IZ/30323
404 R=R+RND: COMB = R - INT(R)

```