The use of time-shared terminals in psychology*

BERT F. GREEN, JR.

The Johns Hopkins University, Baltimore, Maryland 21218

Not every psychology laboratory can have its own computer; but every department *could*, for much smaller cost, have a time-shared computer console, with service provided commercially or by their institution. Time-shared consoles can do most of the things that a laboratory computer can do, including numerical calculations, data analysis, stimulus preparation, running of experiments, and computer simulation. Indeed, equipping a small psychology department requires only a typewriter, a duplicating machine, a pencil sharpener, and a time-shared computer console. For a department with no computer experience, the time-shared console is the way to start. The major difficulties encountered with laboratory computers (programming, interface design, and maintenance) are easily overcome. You write your own programs, the console itself is the only interface, and the time-sharing service maintains the console, the computer, and the system's software.

Experienced computer users will always advise you to do your own programming. With a laboratory computer this often means learning machine language, whereas a time-shared console provides a simple programming language easily learned by the novice. Many time-shared services offer FORTRAN for those old-timers who were weaned on batch processing, but other languages especially designed for time sharing are also offered. JOSS, FOCAL, and TELCOMP are sometimes found, but BASIC is the standard language. BASIC is ideally suited to beginners writing small programs on time-shared consoles and, in my view, is clearly the first language to learn. BASIC has some disadvantages (all variables must be named by a single letter or a letter-numeral pair, all variables in subroutines are global), but it also has advantages (no distinction between fixed- and floating-point numbers, no restrictions on how indices can be indicated). Mainly, BASIC is clear and simple. All the examples in this paper will use BASIC; its transparency should be obvious.

*This work was supported by a research grant, BG 7605X, from the National Science Foundation.

DESK CALCULATOR

The console can serve admirably in place of a desk calculator. And without doubt, it is much preferable to an electronic calculator, programmable or not. The cost of a programmable calculator will pay the rent for the time-shared console for 2 or 3 years, and the console is more powerful and more flexible. Professors who do not know programming will balk, but if they can learn to put two numbers into a calculator and hit the divide button, they can certainly learn to type "PRINT 153.25/15." To add a column of numbers is more of a problem, requiring five lines of code, but the task is simple. Indeed, the time-sharing service we use at Hopkins provides a special program that permits the console to be used precisely like a desk calculator, avoiding all programming.

The computer's advantage over the calculator becomes clear when a calculation of modest complexity must be done repeatedly. Last week, for example, I needed an example of partial correlation for my class in multivariate data analysis. A slide rule would have served for one or two sample calculations, but in order to explore a variety of examples, I chose to write the following little program:

```
10 INPUT R1, R2, R3
20 LET R = (R1-R2*R3)/
   SQR((1-R2↑2)*(1-R3↑2))
30 PRINT R, R1, R2, R3
40 GO TO 10
50 END
```

When each set of three correlations is typed in, the partial correlation pops out immediately and the computer asks for another set.

DATA ANALYSIS

Standard methods of data analysis can generally be done easily on any computer, time-shared or not. The time-shared console has the advantage of being handy and of producing results immediately. The apparent ease of getting results may help to encourage careful testing and checking to insure that the data were entered correctly. Time-shared services usually have "canned" programs for the typical analyses, including correlations, *t* tests, equal-*n* analysis of variance, and stepwise regression.

Some analyses are so lengthy that they should probably be reserved for a big machine, e.g., factor analysis and multivariate analysis of variance, but this depends clearly on the power of the service available.

Multiple correlation is easily done on the time-shared console and a canned program will usually be available, but it is a good example for demonstrating the matrix-handling features of BASIC. The program for multiple correlation is shown in Fig. 1, together with a special-purpose addendum and a sample run. The program reads *N*, the number of predictors, then reads the matrix of predictor intercorrelations, *R*, and the vector of validity correlations, *V*. In Lines 32-38, some matrices are set to their working dimensions before being used in the calculations, so the program is more lengthy than we might have expected. In Line 40, *R* is inverted and in Line 50 the beta weights are calculated. Lines 60 and 65 calculate the squared multiple correlation and Lines 70, 80, and 90 print out the results. The program is short and can take any number of variables. (For more than 10 variables, a dimension statement is needed.) The purpose of the rest of the program lines, 110-180, is to demonstrate the effect of suboptimal weighting. The user inputs a vector of arbitrary weights and obtains the correlation of the criterion with his weighted composite.

Computer systems have been devised for more elaborate interactive data analysis (Press, Rogers, & Shure, 1969). Generally, these systems are not available on commercial time-shared facilities, since they require very large computers to run.

A study by Young and Cliff (1971) combines data analysis with data gathering in an interactive way. The problem is to reduce the enormous number of judgments required in multidimensional scaling procedures. In the Young-Cliff system, a small subset of the stimuli is scaled first, then stimuli are added a few at a time, and the *S* is asked only for those judgments that are likely to yield useful information. The procedure requires from 1/3 to 1/2 the usual number of judgments, with little loss in accuracy. At present the procedure requires either a very powerful computer or a long wait between sets of trials, but eventually that can be reduced. We can expect more of this type of interactive computer use in the future.

PREPARING STIMULI

A computer of any sort can be used to prepare stimuli for use in standard psychological experiments. The main

```

LIST
RMULT

10 HEAD N
20 MAT HEAD R(N,N)
30 MAT READ V(N)
32 MAT S=ZER(N,N)
33 MAT A=ZER(1,1)
34 MAT B=ZER(N,1)
38 MAT C=ZER(1,N)
40 MAT S=INV(R)
50 MAT B=S*V
60 MAT C=TRN(B)
65 MAT A=C*V
70 PRINT "MULTIPLE R =" ; SQR(A(1,1))
80 PRINT "BETA WEIGHTS "
90 MAT PRINT B
110 MAT INPUT W(N)
120 MAT C=TRN(W)
130 MAT A=C*V
140 LET C1=A(1,1)
150 MAT B=H*W
160 MAT A=C*B
170 PRINT "CRITERION R=" ; C1/SQR(A(1,1))
180 GOTO 110
900 DATA 2,1,.4,.4,1,.6,.5
999 END

```

```

RUN
RMULT 0949

MULTIPLE R = .663684
BETA WEIGHTS
.476191

.309524

71,1
CRITERION R = .657376
72,1
CRITERION R = .661724
71,2
CRITERION R = .682799
73,1
CRITERION R = .653156
71,3
CRITERION R = .59636
?
DONE AT 0950

```

Fig. 1. Listing and sample run for RMULT, a BASIC program that computes multiple correlation and correlations of arbitrary input composites.

advantage of the time-shared console, again, is its immediate response and the ease of programming. Many of the list-making possibilities of computers were discussed by Green (1963), together with randomization techniques.

A colleague of mine, Stewart Hulse, recently wanted the computer to generate a sequence of binary events (1s and 0s) in which the probability of a 1 changed regularly throughout the series. As a start, he decided to use a linear change in probability, so the program required three parameters—the number of events, N, the probability at the beginning of the series, P1, and the probability at the end of the series, P2. The program, shown in Fig. 2, first calculates the change from one event to the next, D, presets P, and presets the random-number generator by the

mundane procedure of throwing away M numbers. Then, for each of N events, the computer chooses E, either 1 or 0, according to the current P, prints both P and E, and changes P. Alternative ways of changing P, such as geometrically or logarithmically, would be equally simple, as would generating blocks of N1 trials at each level of P.

RUNNING EXPERIMENTS

Laboratory computers are usually touted as unexcelled for running psychological experiments, but the time-shared console does nearly as well in many applications. Huesman (1970) and Jones, Johnson, and Young (1969) have given several examples of experiments that could be run by sitting a S at a console, presenting stimuli on the typewriter, and requiring the S to type his responses.

These experiments do not have precise timing requirements; response latencies are not required, or can be obtained by manual use of a stopwatch. This eliminates paired-associate-learning experiments and many perceptual experiments; we shall stay in the realm of problem solving, concept formation, and decision making.

As a simple example, consider the old warhorse of human learning and decision making, the binary guessing experiment. On each of a long series of trials, the S guesses which of two alternative events will occur, say "1" or "2." In fact, the events are controlled by a stationary random process. On each trial with probability P, the event is "2"; elsewhere it is "1." The typical result is that Ss tend to guess "2" with probability P—the "probability match" phenomenon.

A program to run this experiment is shown in Fig. 3. The parameters are M, a reset number for the random-number generator, N, the number of trials, P, the probability of a "2," and S, the S number. The program records the results on a file named BG1 for later analysis. The details of the file system are not relevant; the main fact is that three lines (10, 40, and 280) suffice to record the file. Lines 60-80 preset the random number generator, Line 100 presets the counter, Line 110 presets the S. Some programs print out all the instructions to the S (Freeman, 1971) and there is nothing difficult about doing so, but there is nothing interesting either, so we shall ignore it. Whether the E or the computer gives the instructions is mainly a matter of taste.

The loop for N trials is 200-300. On each trial, the S's guess, G, is input and checked; then the "true" event, T, is determined by using the defined function FNB(P) which yields 2 with probability P; elsewhere it yields 1. Then the appropriate counters are incremented, G and T are recorded on

```

LIST
HULSE

10 READ M,N,P1,P2
20 LET D=(P2-P1)/(N-1)
30 FOR I=1 TO M
40 LET X=RND(0)
50 NEXT I
60 LET P=P1
70 FOR I=1 TO N
80 LET E=INT(1+P-RND(0))
90 PRINT P;E
100 LET P=P+D
110 NEXT I
120 DATA 387,20,1,0
130 END

```

Fig. 2. BASIC program to find and print a series of binary events with linearly changing probability.

LIST
BINGES

```

10 FILES BG1
20 READ M,N,P,S
30 REM 3 RECORDS PER SUBJECT IN FILE
40 PRINT #1,1+3*(S-1);M,N,P,S
50 REM DISCARD M RANDOM NUMBERS
60 FOR I=1 TO M
70 LET X=RND(0)
80 NEXT I
90 REM C(1,J) COUNTS RESPONSES J TO STIMULI I
100 MAT C=ZER(2,2)
110 PRINT "WHEN ? APPEARS, TYPE YOUR GUESS(1 OR 2)"
120 DEF FNB(P)=INT(2+P-RND(0))
130 REM FNB(P) YIELDS 2 WITH PROBABILITY P,1 WITH PROB 1-P
200 REM LOOP FOR N TRIALS
210 FOR I=1 TO N
220 INPUT G
230 IF G=1 OR G=2 THEN 260
240 PRINT "INPUT ERROR, ONLY 1 OR 2 ALLOWED; RETYPE."
250 GOTO 220
260 LET T=FNB(P)
270 LET C(I,G)=C(I,G)+1
280 PRINT #1,G,T
290 PRINT TAB(7);T
300 NEXT I
310 PRINT 100*(C(1,1)+C(2,2))/N;" PERCENT CORRECT"
320 STOP
900 DATA 500,20,.75,1
999 END

```

RUN

BINGES 0942

WHEN ? APPEARS, TYPE YOUR GUESS(1 OR 2)

```

?1
1
?1
1
.
.
.
?2
.
2
?2
1
90 PERCENT CORRECT

```

DONE AT 0943

Fig. 3. BASIC program to run the binary guessing experiment. A sample is shown partially.

the file, T is printed out for the S to see, and the next trial begins. At the end of N trials, the S is told his percentage of hits and the program stops. The timing is such that the computer prints the value of T within 2 sec of receiving the input of a trial. Twenty trials can be run in less than a minute if the S guesses swiftly.

One interesting feature of this program is the ease with which it can be modified. As a simple example, consider adding a condition in which negative recency is a correct hypothesis. That is, exactly $P*N$ of the N events are "2," as if N cards were shuffled and turned over one at a time. The events are a random permutation of $P*N$ 2s and $N-P*N$ 1s. We must preset two additional counters, alter the statement obtaining T, and add one statement to decrement a counter:

```

140 LET N(1) = INT(P*N)
150 LET N(2) = N-N(1)
250 LET T = FNB(N(1)/(N(1)+N(2)))
255 LET N(T) = N(T)-1

```

The algorithm is described by Green (1963) and by Walsh (1971).

Freeman (1971) actually did this experiment, with $P = .75$, obtaining positive recency with the strictly random group and negative recency with the random permutation group.

A somewhat similar experimental design is found in studies of concept formation. As a concrete example, consider an object to be a string of six symbols, each either 1 or 0. Some are ZAPS, and the rest are DUDS. A ZAP has a specific two of the six symbols, both 1 or possibly both 0. The S's task is to discover which pair of symbols governs and whether they are to be 0

or 1. In the computer-driven experiment, the E may supply the problems or the computer may generate them at random. For each problem, the computer must allow up to N trials, on each of which the S may present any single object that he wishes to be classified as a ZAP or DUD or he may offer his solution to the problem. The computer's main value is in doing the bookkeeping and in accurately classifying the objects presented.

Johnson has added an interesting feature to this experiment that absolutely requires a computer. In his version, the computer never sets the problem initially but makes it up as it goes along. Specifically, it considers all 30 hypotheses initially. As each object is presented by the S for classification, the computer sees which class is implied by each hypothesis and picks the more frequent class. The hypotheses disconfirmed by the computer response are marked as untenable by the program, and the next trial occurs. Thus, the S can never identify the correct answer until he has ruled out all the other alternatives, simply because there is no right answer until all but one of the possibilities have been marked untenable.

The program to carry out the Johnson experiment requires over 100 lines to express and will not be shown. This is the only one of our examples that is not child's play to a professional programmer.

As a final example, let us consider an experiment in social psychology, the common-target game. There are three players who cannot communicate with each other but who must jointly develop a strategy for solving the problem. On each trial, the E presents a target number; each of the three players must respond with a number from 0 to 10, in the hope that the sum of the three responses will equal the target. At the end of each trial, each S learns the responses of all three Ss. In one version of the game (Joyner, 1970), the S must respond not only with his own number, but also with his guess as to what his two partners will say.

It would be possible, if difficult and cumbersome, to run this experiment with three consoles simultaneously, but most social psychologists use stooges these days; we shall use one real player and two stooges, both of whom are merely variables in the computer program.

There are many possible strategies in the common-target game. One is to divide the target number by three, with one player adding the remainder. Another is to divide by two, with the third player absorbing only the remainder. A third, seldom found by

LIST
COMTRG

```

10 REM N TRIALS,H-STRATEGY
20 REM M PLAYER ROLE
30 READ N,H,M
40 DEF FNN(N)=INT(N*RND(0))+1
50 REM ADD OTHER PRESETS WHEN SIMULATING
60 FOR I=1 TO N
70 READ T
80 PRINT "TARGET IS";T
90 INPUT R(1),R(2),R(3)
100 PRINT "SUBJECT 1: ";R(1),R(2),R(3)
110 GOSUB H OF 300,400,500
120 LET Z=S(M)
130 LET S(M)=S(1)
140 LET S(1)=S(M)
150 LET S=R(1)+S(2)+S(3)
160 PRINT " RESULT: ";R(1),S(2),S(3)
170 IF S=T THEN 200
180 PRINT "MISS"
190 GOTO 210
200 PRINT " HIT"
210 NEXT I
220 STOP
300 REM TWOS REMAINDER STRATEGY
310 LET S(1)=10 MIN INT(T/2)
320 LET S(2)=S(1)
330 LET S(3)=T-S(1)-S(2)
340 RETURN
400 REM THREES REMAINDER STRATEGY
410 LET S(1)=10 MIN INT(T/3)
420 GOTO 320
500 REM TENS STRATEGY
510 LET S(1)=T MIN 10
520 LET S(2)=(0 MAX (T-10)) MIN 10
530 LET S(3)=0 MAX (T-20)
540 RETURN
900 DATA 5,3,2
910 DATA 6,24,15,28,1,7,17,27,14,9
920 DATA 10,3,26,12,18,30,26,25,2
999 END

```

Fig. 4. BASIC program to run the common-target game with two hypothetical stooges.

scientifically oriented players, is the 10s strategy: Player 1 says 10 only when the target is 20 or more; otherwise he says 0. Player 2 says 10 only when the target is 10 or more; otherwise he says 0. Player 3 gives the units digit. Let us use only these three strategies in our sample program in Fig. 4. The parameters for a run are which strategy the stooges are to use, which roles they are to play (actually specified by which role the S is to be "trained" to play), and how many trials are to be run. Line 40 defines a useful function that yields random integers from the range 1 to n. The trial loop starts with the reading of the target from the list supplied by the E, presentation of it to the S, and his response input. After printing his response, the program determines what the responses would be if all were following the given strategy and switches the responses of the first S and the response for the specified role so that the specified role is always in first position. Then it prints out the S's actual response along with the stooge's responses. It then redundantly prints HIT or MISS, depending on

whether or not the "team" hit their target. The subroutines for the various strategies are very simply stated by use of minimum and maximum operators.

In this version of COMTRG, the stooges are very rigid. A variant would permit the stooges to comply with the S's strategy. The stooges would

```

2000 REM SIMULATED SUBJECT RESPONDER
2005 REM H2 IS S'S HYPOTHESIS;M2=S'S ROLE
2010 GOSUB H2 OF 300,400,500
2020 LET R(1)=S(M2)
2030 LET R(2)=S(2)
2040 LET R(3)=S(3)
2050 LET R(M2)=S(1)
2060 RETURN
2100 REM SIMULATED SUBJECT HYPOTHESIS OR ROLE CHANGE
2110 FOR J=2 TO 3
2120 IF S(1)=R(J) THEN 2180
2130 NEXT J
2140 LET H2=FNN(H)
2150 LET M2=FNN(3)
2160 RETURN
2180 LET M2=J
2190 RETURN
9999 END

```

Fig. 5. Subroutines to simulate a S's behavior in the common-target experiment.

determine what role in what strategy would lead to the S's last response and respond accordingly. The program would be somewhat more complex, and the computer would be indispensable.

COMPUTER SIMULATION

An interesting feature of the common-target program, as of any program for running an experiment, is that simulation becomes very easy to add (Fig. 5). The input statement is replaced by a subroutine that simulates the S's responses (GOSUB 2000 at Line 90). Some presets must be inserted (GOSUB 2140 at Line 50), and a procedure for changing the S's strategy must be included (GOSUB 2100 at Line 185). The particular simulation shown here merely assumes that, when the team misses, the S checks to see if one of his partners matched his response, in which case he merely switches roles. Otherwise, he selects a new strategy and a new role at random.

Joyner (1969) proposed a much more elaborate simulation involving lists for long-term and short-term memory. His algorithms could be stated in BASIC, but many more lines would be needed. However, it is important to understand that simulation does not require a special list-processing programming language.

OTHER CONSOLE USES

The console can be useful in areas other than the laboratory. Computer-based instruction, computer-based testing, and computer-based guidance are all worthy pursuits (Holtzman, 1970). Also of considerable interest to human factors experts is the nature of man-computer interaction as typified by the time-shared console. Chapanis (1971) has argued strongly that current methods of man-computer

communications are too restricting and stultifying. Studies of time sharing vs batch processing have tended to find time sharing more popular but less efficient (Sachman, 1970). One of the most fascinating results (Seven, Boehm, & Watson, 1971) concerns the beneficial effect of an enforced time-out period after each computer request in an interactive problem-solving situation. In short, people sitting at the console feel constrained to use it, so they type things without enough prior thought, perhaps to give the impression that they are working. How to change this pattern of behavior is worth more study.

COSTS

One difficulty with the time-shared console is its cost. At minimum it will cost \$1000 per year and, if it is popular, perhaps as much as \$3000 per year. This is still substantially below the cost of amortizing and supporting an in-house laboratory computer, but it is not cheap. One of the crosses time sharing has had to bear is the per-hour charge. No other psychological equipment is rented by the hour; students and professors alike should try to avoid thinking of the timed charges, though of course they should be careful.

Will programs written for the time-shared console transfer easily to a laboratory computer? Yes. Most minicomputers provide BASIC; the others provide a close cousin such as FOCAL or TELCOMP. Thus the laboratory computer will typically be able to run the programs with little change.

My purpose in displaying several sample programs has been to show how very simple and straightforward the programs are and, thus, to imply that they are not difficult to learn how to write. There is an added implication that the computer permits much more intricate experiments than have been the fashion in the human experimental laboratory.

REFERENCES

- CHAPANIS, A. Prelude to 2001: Exploration in human communication. *American Psychologist*, 1971, 26, 949-961.
- FREEMAN, L. C. A computer controlled experiment on recency in probability learning. *Behavioral Science*, 1971, 16, 174-179.
- GREEN, B. F. *Digital computers in research*. New York: McGraw-Hill, 1963.
- HOLTZMAN, W. (Ed.) *Computer-assisted instruction, testing, and guidance*. New York: Harper & Row, 1970.
- HUESMANN, L. R. Controlling experiments with a general-purpose time-sharing system. *Behavior Research Methods & Instrumentation*, 1969, 1, 276-280.
- JONES, L. V., JOHNSON, E. S., & YOUNG, F. W. *Computer control of psychological experiments*. Chapel Hill, N.C.: L.L. Thurstone Psychometric

Laboratory, University of North Carolina, No. 80, November 1969.

JOYNER, R. C. Computer simulation of individual concept learning in the three-person common target game. *Journal of Mathematical Psychology*, 1970, 7, 478-514.

PRESS, L. I., ROGERS, M. S., & SHURE, G. H. An interactive technique for the analysis of multivariate data. *Behavioral Science*, 1969, 14, 364-370.

SACHMAN, H. *Man-computer problem solving*. Princeton, N.J.: Auerbach, 1970.

SEVEN, M. J., BOEHM, B. W., & WATSON, R. A. A study of user behavior in problem solving with an interactive computer. Santa Monica, Calif: Rand Corporation, R-513-NASA, April 1971.

WALSH, J. F. An efficient procedure for selecting equal probability samples. *Behavior Research Methods & Instrumentation*, 1971, 3, 212-213.

YOUNG, F. W., & CLIFF, N. *Interactive scaling with individual subjects*. Chapel Hill, N.C.: L. L. Thurstone Psychometric Laboratory, University of North Carolina, No. 94, May 1971.

Misuse, abuse, overuse, and unuse of on-line computer facilities by psychologists

WILLIAM R. UTTAL

University of Michigan, Ann Arbor, Michigan 48104

When Dr. Tepas asked me to discuss some of the negative aspects of the psychological use of on-line computers, my first thought was, "Oh boy, here's a chance to view with concern and point with alarm at what must be a morass of one or another sort of computer application evil." After a little thought, though, it appeared to me that this was not exactly the case and such a perspective could not be entirely accurate.

In the last 7 or 8 years since the announcement of the first computers of a size that made possible almost personal facilities, there has been a flowering of installations, applications, and I think, in the main, wholesome productivity. Computers now cost inconsequential amounts compared to the other expenses of running any sort of a laboratory operation, and thus the economic issues are no longer of primary concern. The most important aspect of laboratory computers is still their ability to open new theoretical and empirical doors and to provide us with new techniques for the analysis of problems which could not possibly have been examined previously. I hope it is not too trite to repeat the oft-quoted analogy that the computer is as important to psychology as the microscope was to biology. Of course, we will have to wait a few decades or longer to see if what seem to be the likely long-range contributions of this instrument do develop.

As I look around now, I see that there are many, many on-line computer installations. I know of a number of individual psychologists who are simultaneously using two or even three small computers in a productive fashion. The many positive reasons for this increasingly important

contribution of computers to experimental psychology are clear. The other two speakers this morning have spoken to these points, and I shall not reiterate the argument for computerization. I would like to note that the fact that laboratory control computers are so ubiquitous is itself an argument of considerable consequence for their general utility and importance. Unfortunately, there are also instances of unfulfilled needs that seem so obvious among some of our colleagues that they must be acknowledged. I am convinced that there are many more of my colleagues in need of computer assistance in their laboratories than there are those who are misusing them.

Before I start with my assigned task of a broad critique, I do want to make it clear that I believe that the positive side of on-line computer applications does very much outweigh the negative aspect. But my assigned role today is that of the critic, and I must to some degree view with alarm and point with concern; as usual, it is always possible to do so. Some issues I will raise will be of considerable importance. Others will seem, to some at least, to be of only minimum consequence. All, I hasten to warn you, are based upon my personal perspectives and prejudices, and I am sure there are reasonable men among us who will find ample and perhaps even valid reason to differ with me on certain points. If there is any theme which I can see to all of these comments, it is that they will be concerned mostly with strategies and global approaches to the use of control computers rather than to any specific details of their use.