Behavior Research Methods & Instrumentation
1981, Vol. 13 (3), 374-376

Bringing APL down to earth on the DECsystem-10: Standard characters and a standard editor

FRED A. MASTERSON
University of Delaware, Newark, Delaware 19711

The APL programming language provides an unprecedented combination of simplicity and power. Simplicity characterizes program preparation. Since APL code is interpreted, programs ("functions," as they are called in APL) can be executed as soon as they are written, without the frustrating delays commonly interpolated by compilers. Furthermore, any APL program (function) can pass control to another APL program (function) without a preliminary time-consuming linking procedure. Finally, individual lines of APL code can be interpreted outside an official function definition, a feature that encourages trial-and-error learning. In all these respects, APL shares or improves upon the breezy informality of interactive BASIC, but, unlike BASIC, APL does not disappoint advanced students of programming.

APL is a superior programming system for statistical data analysis. The interactive flexibility of APL facilitates exploratory data analysis. In addition, APL provides powerful operators for the manipulation and transformation of tabular (matrix) data structures. Formulas for univariate or multivariate analysis can be transcribed into APL on a line-to-line basis (i.e., one line of mathematical formula converts to one line of APL code), instead of the one-line-to-many-line basis required by ALGOL, FORTRAN, or PASCAL-type languages. This one-to-one correspondence between mathematics and APL is hardly surprising, since APL was originally created not as a computer programming language, but as an improved notation for algebraic formula (Iverson, 1962).

It is regrettable that a language with so many positive features has not gained wider acceptance among researchers in the behavioral sciences. Potential users are discouraged by three major drawbacks. The first drawback is that APL requires terminals with a special keyboard. Such terminals are relatively scarce at many computing installations, so users have to wait in line. Furthermore, once a user gains access to an APL terminal, he or she is likely to be annoyed by the novel positions of such standard characters as left and right parentheses. Learning the positions of special APL characters is bad enough: There is no reasonable justification for requiring users to learn new key locations for standard characters.

The second drawback to current APL implementations is the required use of a special APL editor. Most computer systems feature a multipurpose editor that serves to edit data files, word processing files, and program source code files. Why, then, should APL require users to master a different editor? This question is particularly important in light of the obvious inferiority of many APL editors to the general-purpose editor of the typical host computer system.

The third drawback to APL is its popular reputation as a "write-only language." This reputation has resulted more from the style of seasoned APL programmers than from APL itself. Expressions in APL are fantastically economical, and it is a huge temptation to push this economy to the extreme by writing highly compressed code. While this may be a defensible practice for experienced APL programmers, it should be resisted by beginning and occasional users of APL, for whom lengthier, easier to read code is desirable.

The extreme compactness of pure APL is illustrated in Figures 1a and 1b. Both examples compute the chi-square statistic for testing the reliability of a relationship between two categorical variables. In Figure 1a,

(a)
```
CHISQ←+/+/[1]((O-E)*2)÷E←((+/O)∘.×+/[1]O)÷(⍴ O)⍴+/+/[1]O
```

(b)
```
NMAT←(⍴ OBSMAT)⍴+/+/[1]OBSMAT
EXPMAT←((+/OBSMAT)∘.×(+/[1]OBSMAT))÷NMAT
CHISQ←+/+/[1]((OBSMAT-EXPMAT)*2)÷EXPMAT
```

(c)
```
NMAT_(SIZE OBSMAT)SHAPE SUM COLSUM OBSMAT
OPERATOR1_'#'
EXPMAT_((SUM OBSMAT)OUTPROD(COLSUM OBSMAT))%NMAT
CHISQ_SUM COLSUM((OBSMAT-EXPMAT)*2)%EXPMAT
```

Figure 1. (a) A one-line APL program for the chi-square computation. O is the matrix of observed frequencies, E is the matrix of expected frequencies, and CHISQ is the value of the chi-square statistic. (b) The chi-square computation spread out over three lines of APL code. The names for the matrices of observed and expected frequencies have been extended for greater readability. NMAT is a matrix in which every element is equal to the total number of observations. NMAT has the same dimensions as OBSMAT and EXPMAT. (c) The code for the chi-square computation in the present system. The difference between this and the listing in Figure 1b lies in the use of standard characters and the use of more suggestive names for operators. (The second line stores the operator–in this case, multiplication–that is to be used in the outer product in Line 3.)

this is accomplished in a single line of APL code. Figure 1b is more practical, if less entertaining: The computation is spread out over three lines, and longer names are used for variables. Even so, many of the operator names lack good mnemonic value. In addition, the special characters demand a special terminal.

To summarize, behavioral scientists who would be enthralled by APL's simplicity and power are nonetheless repelled by unreadable code, unappealing editors, and unavailable terminals. What follows is a description of a system that avoids these negative features. It is based on Digital Equipment Corporation (DEC) APLSF, an extended version of APL for the DECsystem-10. The system was designed primarily for beginning students, as well as for experienced computer users who move back and forth between APL and other languages and systems and would like to use the same character set and text editor for all their applications.

Implementation. DEC's APLSF allows the use of a standard keyboard if the user desires. In some cases, a single standard character can be used in place of an APL character. Luckily, the representation of many common operators follows conventional algebraic notation (for example, addition is represented by "+" and subtraction by "–".) Other common operators are represented by single characters that are reasonably similar to those employed in conventional algebraic and computer notation (thus, multiplication is represented by "#," division by "%," and the value assignment operator by "–").

In other cases, DEC allows the users to substitute strings of standard characters for APL characters. As shown in Table 1, ".RO" may substitute for "ρ" (the rho or shape operator) and ".DA" may be substituted for "↓" (the down-arrow or drop operator). One problem with these substitutions is that they lack adequate mnemonic value. In addition, APLSF will not honor spaces between these operator names and the names of the surrounding arguments, resulting in such monstrosities as "N.ROSCORES" or "12.DASCORES" (where SCORES is a data vector or array.) Our solution is to define these operations as user-defined functions. In our system the monadic form of "ρ" is called "SIZE," whereas the dyadic form is named "SHAPE." Similarly, "DROP" is the name of a function that performs the "↓" operation. These names have good mnemonic value. In addition, APL respects spaces on either side of the names of user-defined functions. Hence, the jumbled-together examples given above are replaced by the more readable "N SHAPE SCORES" and "12 DROP SCORES."

These and other user-defined functions are shown in Table 1. Some user-defined functions have been created specifically for beginners. Thus, "SUM," "ROWSUM,"

Table 1
APL Operators Expressed in APL characters, in APLSF Equivalent Standard Characters, and as User-Defined Functions in the Present System

Operation	APL Keyboard	Standard Keyboard	User Defined Functions
Remove the part of Y specified in X	X↓Y	X.DAY	X DROP Y
Take the part of Y specified in X	X↑Y	X^Y	X TAKE Y
Find the dimensions of Y	⍴Y	.ROY	SIZE Y
Restructure Y according to the dimensions specified in X	X⍴Y	X.ROY	X SHAPE Y
Generate a vector of the first N integers	⍳N	.ION	GENINTS N
Get input from keyboard	⎕	.BX	KEYB
Base 10 logarithm of y	10⍟Y	10.LGY	LOG Y
Transpose array Y	⍉Y	.TRY	TRANSP Y
Invert matrix Y	⌹Y	.DQY	INVERT Y
Execute string S as APL code	⍎S	.XQS	EVAL S
Sum across a vector Y	+/Y	+/Y	SUM Y
Sum across the rows of a matrix Y	+/Y	+/Y	ROWSUM Y
Sum across the columns of a matrix Y	+/[1]Y	+/[1]Y	COLSUM Y

and "COLSUM" in Table 1 provide easy to understand alternatives to "+/" and "+/[1]." The latter notations utilize APL's compression operator ("/") and can be reserved for more advanced users.

Our system permits the use of the DECsystem-10 general-purpose editor, SOS, for creating and modifying APL function definitions. The transfer of control between APL and the SOS editor is completely transparent. The user merely invokes an APL function named EDIT, supplying as argument the name of the function to be created or modified. Thus, typing

EDIT 'MEAN'

automatically transports the user into SOS. Furthermore, if MEAN has been defined previously, the old definition is automatically loaded into the SOS work file. The user then can use standard SOS commands to modify the definition. When finished editing, the user types

G

which has the automatic effect of transporting the user back to APLSF along with the new definition. The net result is a smooth and simple flow between APLSF execution mode and SOS-mediated function-definition mode.

This APLSF-SOS interaction capitalizes on some features that are peculiar to DECsystem-10 APLSF and SOS: (1) APLSF can read and write standard ASCII disk files. This is essential for the EDIT function to be able to write an old definition on the SOS working disk file (which must be in ASCII) and, later, to read back the new definition. (2) APLSF has the special system command,)CALL, which temporarily suspends APLSF and transfers control to another program (SOS, in the present case). (3) SOS can be preset to attend to a particular disk file. In the current case, this is the temporary work file used for modifying the function definition. (4) SOS recognizes "G" as a command to exit SOS and then execute a program predefined by the user. In the present application, this is a machine language program that executes APLSF with an address offset of 1 (the address offset serves to avoid the opening messages that occur when one enters APLSF at the very top).

A documented listing of the definition of EDIT, as well as a description of the use of our total system, is available upon request.

Discussion. The system described preserves the interactive simplicity and powerful operations of DECsystem-10 APLSF while permitting the use of standard characters and the standard DECsystem-10 SOS editor. Introductory statistics students find this system easy to learn. Experienced users of the DECsystem-10 appreciate being able to use the same text editor and character set as they use in other DECsystem-10 applications.

The use of character strings with good mnemonic value leads, in practice, to more readable APL code. Besides aiding memory, the strings are long enough (three characters or more) to avoid the mania for compacted code that typically occurs when the operators are named by single, typographically distinctive characters. I am easily persuaded that pure APL, special characters and "one-line" definitions, can provide hours of exciting intellectual entertainment. For the average user and the average application, however, a more prolix APL seems preferable.

Figure 1c shows how the present system represents the chi-square computation discussed earlier. Figure 1c is similar to Figure 1b. The difference lies in the use of standard characters and more suggestive operator names. APL programmers will note the use of the outer product operator, here represented as "OUTPROD," to produce a multiplication table of row sums times column sums, which enters into the computations of the table of expected frequencies. The operator used in the outer product, in this case, multiplication, has been predefined in the second line of Figure 1c.

At the University of Delaware, we have a strong interest in the human engineering of software in general and programming languages in particular. From this perspective, a "demystified" APL with standard characters, a standard editor, and meaningful operator names represents a more approachable APL, at least for beginners and occasional users of the language.

REFERENCE

Iverson, K. *A programming language.* New York: Wiley, 1962.

(Accepted for publication June 12, 1981.)