YALE UNIVERSITY

Department of Statistics

Technical Report No. 4

July, 1968

"Use of Iverson's language APL for statistical computing"

By F. J. ANSCOMBE

This research was supported by the Army, Navy, Air Force and NASA under a contract administered by the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government.

Contract number: Nonr 609(52)

Task number: NR 042-242

[This paper will be offered for publication in a journal. Comments will be welcome.]

Use of Iverson's language APL for statistical computing

F. J. Anscombe

1. STATISTICAL COMPUTING

A symposium on statistical computing was held by IBM in 1963; the proceedings were subsequently published [10]. Another such symposium was held in London in 1966; papers have been published [14]. During the latter symposium a working party was formed to coordinate and stimulate further developments in statistical computing.

Prominent among the topics discussed in the London symposium and elsewhere in the recent literature are

(i) standardization to facilitate exchange of information,

(ii) communication between persons and computers.

Under item (i) come questions of standardizing the layout and coding of tape files of statistical data -- for example, files of material collected in a sociological survey. The aim here is to permit copies of the tape to be studied and used by workers at different centers, using different computing equipment and possibly different languages. Suggestions have also been made for standardizing some features of computer programs. Standard variable names would facilitate reading of programs by others. Good documentation and testing of programs are stressed.

Standardization is not discussed in this paper.

As for item (ii), the computer has not so far had the profound effect on statistics that it has had on some other fields of science and technology. Statistical method as generally practised today was largely developed during the period between the world wars, 1918-1939, and was conditioned by the principal computing resource of that time, the desk calculator. In recent years numerous computer programs have been prepared to reproduce on a large computer the kinds of calculations formerly done on desk calculators. Such programs have greatly aided statistical work. But the potential aid from the computer is vastly greater. If large amounts of computing can be carried out rapidly and with little effort by the investigator, output can be asked for that formerly was not thought of because it was unthinkable. A challenge is offered to statistical theory: if almost any output can be had for the asking, at almost no cost, what output should be asked for -- what questions is the output relevant to, what can be made of it?

There has certainly been some progress toward realizing the potentialities of the computer in statistical work. Many new techniques have been tried. The main reason why more progress has not been made is, presumably, that computers are hard to communicate with. The old desk calculator had some merits not found readily in a large computer. In laboring over the figures, the operator of a desk calculator sometimes obtained unexpected insight, and could at any moment modify his plan of calculation. By contrast a computer program is liable to seem uninformative and inflexible or, if informative and flexible, then confusing.

Statistical analysis is experimental. It has always been so to some extent, it will become more so in the future. When a statistician is presented with some data, he may know what analytic steps he would like to see carried out first, but he cannot know until he has explored the consequences of preliminary analyses what he will come to consider a satisfactory final analysis of the data. Good statistical analysis is done in steps. Methods must be adjusted to fit the data; the adequacy of theoretical descriptions or "models" must be assessed. The task of statistical analysis is much less well defined at the outset than the task of, say, tabulating a mathematical function. Interaction between the investigator and the computer is required.

The persons who have occasion to use computing equipment for statistical analysis are very diverse in their knowledge of, and interest in, both statistics and computing. A facility that is attractive to one may be intolerable to another. For example:

(a) Consider a researcher who works in a field that yields statistical problems, but who is not himself primarily a statistician. He wishes to use good statistical tools to better his understanding of his field. He is not interested in statistical methods for their own sake, and does not wish to experiment with them (more than he has to). The less he must be concerned with the technicalities of statistical calculations the better.

(b) At another extreme, consider a statistician, interested in the development of statistical procedures, interested in relevant mathematics, interested in particular bodies of data mainly as examples of statistical problems. He may be happy to use a computer program written by someone else, but only if he fully understands what the program does. If communication with the computer is easy enough, he generally prefers to write his own programs, so that he may do exactly what he wishes, rather than what someone else has wished.

Numerous attempts have been made to accommodate Man (a). Many program packages have been written, for operation in batch mode, in which the user specifies various options in output. The BMD biomedical programs [3] are a well-known example. When operation in conversational mode is possible, the computer can "take charge" of the whole procedure, by interrogating the user about what material he wants analyzed, explaining to him what procedures are available and asking him at each step what he would like done next. Such an interrogatory system, if well designed, can give assistance and guidance to the user, while permitting him considerable freedom of choice. Examples are Schatzoff's system COMB for two-way tables [11, 1] and A.J.T. Colin's statistical system ([14], pp. 111-119).

It seems likely that much further attention will be paid to this type of system. The problem of designing one is not only statistical but psychological. How can the interrogation be conducted, so that misunderstanding is unlikely, guidance is offered to users who need it, and the system does not seem excessively clumsy to experienced users who know how they wish to proceed?

As for Man (b), he needs to be able to write his own programs. An explosive development of statistical science can be expected once programming can really be done by any interested person, without a large preliminary investment of time in mastering a computer language and without much time spent in actual coding. (Time spent in deciding just what to do and how to do it is another matter altogether, for which the computer and its software should not be blamed!) No doubt Man (b) will make use of previously written programs (preferably written by himself), but he needs also to be able to improvise.

What makes programming so tedious in FORTRAN and other commonly used languages is the negotiation of arrays. Arithmetical operations are required, not just on individual numbers, but on whole vectors or matrices; and in these languages such operations must be spelled out in loops -- loops, and loops within loops, initialized, ranged and fussed over. Numerous attempts have been made to lessen the tedium through special computing systems designed for particular types of user. It is noticed that a certain type of user is mainly concerned with a not very broad class of array operations, and a vocabulary of such operations is made available to him as a special system. Thus OMNITAB [5] was based on the fact that much of the work of tabulating mathematical functions and also of processing experimental observations involves a small variety of operations on vectors, and so these are made directly available in the system. STORM [9] provides a library of matrix operations designed for analysis of variance and regression calculations. COSMOS (Schatzoff [12]) offers primarily six basic operations on vectors and matrices in terms of which analysis of variance, factor analysis and other quadratic calculations can be readily expressed.(1)

The capital difficulty in designing any such system is to know when to leave off. As soon as the user is able to perform his standard calculations easily, he finds he sometimes wants to do further things not already provided for. There is pressure for the system to be extended, extended even to the point that it encompases everything that can be done in a general-purpose algorithmic language like FORTRAN. But now the system must be quite rich, because of the diversity of logical operations that occur in programming. What was at first a collection of a few easily remembered commands becomes something much more intricate and difficult to assimilate -- though possibly effective and worth assimilating.

The purpose of this note is to suggest that K. E. Iverson's language known as APL (after the title of his original book on the subject [6]), though not developed specially for statistical work, is in fact very suitable for this purpose. Two salient reasons are:

(i) APL was designed at the outset to handle (almost indifferently) scalars, vectors, matrices and rectangular arrays in any number of dimensions. All the basic arithmetic operations can be performed on arrays just as well as on scalars, without any loop written in the program. Programs in APL therefore tend to contain few loops. The programmer is encouraged to think of array operations as entities without a logically irrelevant internal sequence; this is aesthetically pleasing, even illuminating.

(ii) There is a high degree of consistency in APL, resulting from a high degree of generality in the definitions. Syntax is governed ruthlessly by a very few simple rules. Once the basic vocabulary is learned, the language is easy to remember. There is a remarkable absence of arbitrary features that require frequent reference to the manual. The language therefore has a peculiar dignity and reasonableness. One feels it is worth learning.

What is likely to prove the relative usefulness of APL for statistical computation, compared with special statistical systems? It has already been suggested that some users, including Man (a), will be best served by an interrogatory system in which nothing that would ordinarily be called programming is called for. Other users, with technical interests intermediate between those of Man (a) and Man (b), will be able to make good use of a noninterrogatory statistical system. The more the user resembles Man (b), the more directly he will wish to control his computing and the greater the variety of computing he will wish to do. Man (b) will, as it seems, be best served by the best general-purpose algorithmic language, which at present appears (by an order of magnitude) to be APL.(2)

If this last conclusion ultimately receives general assent, the situation in regard to statistical software will resemble that for hardware. In the later 1940's, before big computers were generally available, discussions were held between statisticians and computer experts about statistical computing. Would it be possible to develop a machine for analysis of variance, having special facilities for summing squares and products? The experts advised against this; the effort would be better expended on a general-purpose machine. We see now that their judgment was right. If a good-enough general-purpose machine is available, no one wants to be handicapped with a special-purpose machine. The situation seems to be similar with software.

2. DESCRIPTION OF APL

Iverson's programming language, as originally published in 1962, was intended for precise and concise expression of algorithms, so that computing procedures could be discussed and communicated. It differed from ALGOL in having a larger vocabulary, a more powerful syntax, and a notation and way of thinking closer to established mathematics.(3) No attention was paid to the existing peculiarities of computer hardware and organization, nor to the existing meagerness of key-punch character sets.

Recently a modified version of the language, called APL-360, has been implemented experimentally at IBM's Thomas J. Watson Research Center, Yorktown Heights, N.Y., as a coding language for computation in conversational mode through typewriter terminals. The computer is an IBM 360 model 50 (a model 65 or 67 is also used). The system is interpretive; programs are not compiled. No language other than APL is accepted in conversational mode (unless a translator into APL is available). Emphasis up to now has been on development and improvement of the system, with only a modest scale of implementation. High-speed input and output devices have not been incorporated; workspaces have been small.(4)

What follows is a partial description of APL, as it can be used in the above-mentioned experimental implementation, for statistical computation. The complete facilities available are described in the manual [4]. This partial description is intended to convey the feel of the language, without introducing the reader too suddenly to too great a flood of unfamiliar notation. Slight previous acquaintance with FORTRAN or other archaic language will be presumed, but for the benefit of the unacquainted there is an appendix on some common computing terms.

(a) Preliminaries, standard scalar functions

The system can be used as a desk calculator. If the operator types "2 × 3" and then touches a key to transmit this message, the computer responds by causing "6" to be typed on the next line. When the keyboard of the terminal is unlocked and ready to receive a new command, the typeball is in such a position that the operator's command is indented six spaces, whereas the machine's reply is usually not indented. Thus a reader can tell which party typed what. See Figure 1. (Still clearer distinguishing can be effected on some terminals by use of a two-color ribbon, arranged so that the operator's instructions are in red and the machine's replies are in black. That refinement is unnecessary, however.)

Usually when a calculation is made the operator wishes to store it for future reference under a name. The name may be a string of one or more letters or digits starting with a letter. Assignment of a value (or other content) to such a name is denoted by a left arrow. If "2 × 3" is assigned to the name X (see line 3 of Figure 1), the machine does not type anything in reply, but when the calculation has been done and the result stored under the name X the carriage is shifted six spaces and the keyboard is unlocked for another command.

`      2×3`	1.	Command typed by operator.
`6`	2.	Reply by computer.
`      X←2×3`	3.	2×3 stored under name *X*.
`      X+2`	4.	Display *X*+2.
`8`	5.	Reply by computer.
`      2+X←2×3`	6.	Two previous orders combined.
`8`	7.	
`      X*2`	8.	*X* squared. Or: *X*×*X*
`36`	9.	
`      X←X+4`	10.	Value of *X* changed to 10.
`      X÷2+2`	11.	Divide 10 by (2+2).
`2.5`	12.	
`      (X÷2)+2`	13.	Or: 2+*X*÷2
`7`	14.	
`      6⌈5`	15.	The greater of 6 and 5.
`6`	16.	
`      6⌊*2`	17.	The lesser of 6 and e^2.
`6`	18.	
`      3!7`	19.	Number of combinations $\binom{7}{3}$.
`35`	20.	
`      1=X÷10`	21.	Is (*X*÷10) equal to 1?
`1`	22.	Yes, that is true.
`      3×X>0`	23.	3 times the truth value of (*X*>0).
`3`	24.	
`      X<!3`	25.	Is *X* less than 6?
`0`	26.	No, that is false.
`      X÷6`	27.	Or: □←*X*÷6
`1.666666667`	28.	
`      0.01×⌊0.5+100×X÷6`	29.	(*X*÷6) correct to two decimals.
`1.67`	30.	

FIGURE 1. Dialog in APL on the left. Comments added on the right.

Whenever the typed command does not assign all calculated quantities to names, the unassigned result is typed out. Thus when we typed (in line 1) just "2 × 3", without assigning the result anywhere, we obtained the typed reply "6". But when 2 × 3 is assigned to X, nothing is typed in reply. We can call for X later by just typing X; since X is not assigned anywhere by that instruction, its value is typed out. In Figure 1, X + 2 is asked for at line 4. (It is possible to refer to the typewriter as though it were a named variable; its name is a square symbol called "quad". We can ask for a result to be typed out by assigning it to quad, as shown in the comment at line 27 of Figure 1. This alternative way of asking for a display is sometimes useful, as we shall see.)

The two commands, to calculate 2 × 3 and store it under X, then add 2 and display the answer, could have been combined in one instruction as shown at line 6.

This illustrates a very simple syntactic rule of APL that saves much trouble in the long run but takes some getting used to. When a command contains more than one operation or function, the right-most operation is performed first, then the next right-most, and so on. Thus in a sense the machine reads from right to left. In the composite instruction of line 6, the machine first computes 2 × 3, then stores that under the name X, then adds X onto 2, then (having nowhere else to put it) types out the result. If we wish to modify this right-to-left execution we must use parentheses -- compare lines 11 and 13.

The four elementary dyadic functions of arithmetic, addition, subtraction, multiplication and division, are denoted by their usual symbols (+, -, ×, ÷). Exponentiation is denoted by a star instead of by raising the exponent -- see line 8. A good many other basic dyadic functions are recognized and given special symbols, the symbol denoting the function always being placed <u>between</u> its two arguments, just as with +. Examples are shown in Figure 1. Note that symbols like =, >, ≤, ... stand for functions taking the truth values 1 or 0 according to whether the corresponding statement is true or false.

Some of these function symbols have a meaning as monadic functions when no argument is placed in front. Thus -A means 0 - A, the negative of A; ÷ A means 1 ÷ A, the reciprocal of A; *A means the natural exponential of A, or e^A; !A means factorial A. The L-like symbol that means "the lesser of" when placed between two quantities (line 17) means "the integer part of" when it has no argument preceding it (line 29).

Numerical calculations are performed by the machine, according to need, either in single-precision integer arithmetic or in floating-point "extended precision" arithmetic with precision equivalent to about 14 decimal digits in the mantissa. Results are ordinarily printed with ten digits, except that trailing zeros after the decimal point are omitted.

There is no vocabulary in the language relating specifically to format, and no requirement that formats be specified. For many purposes there is no need to control the format of typed output, except sometimes by the rounding procedure illustrated at line 29 of Figure 1. The language has facilities for character manipulation, which however are not described here. With these it is possible to control the format of printed output as minutely as is customary with other languages such as FORTRAN, and that should usually be done if the output is to be a table or other matter for photographing. Such format control is not illustrated here.

It so happens that every number shown in Figure 1 is positive. Negative numbers are typed with a "high minus" sign in front of them, regarded as part of the number, not as a function. Compare lines 1 and 3 in Figure 2.

So far we have considered only single numbers (scalars). The peculiar virtue of APL for statistical work will not appear until we come to arrays. But at this point two cleannesses in the language should be noticed. First, there is quite a rich vocabulary of standard functions, each represented by a single symbol (not all of them have been shown in Figure 1). In accordance with ordinary usage for addition (+) and multiplication (×), all functions that have two arguments are invoked by placing the function name between the arguments. Thus we write AfB rather than f(A,B), where f is the name of the function and A and B are the arguments. Similarly, if a function has just one argument, the argument follows the function name, as in fA. These conventions apply also to functions defined by programs, as we shall see below. Thus all functions, whether they are the basic operations built into the language or defined functions that some people would term "macros", are referred to and called in similar style.

It should also be noticed that no rule of inherent precedence among the basic operations (or any other functions) has to be learned. The only precedence comes from position, according to the rule that execution is from right to left. This rule is harder to keep in mind than one might suppose, because it sometimes opposes common usage. Unfortunately, common mathematical usage (and also FORTRAN programming) involves a mixture of left-to-right and right-to-left reading and many arbitrary special conventions. No precedence rule can appear equally natural in all circumstances. APL's right-to-left convention seems pleasantly simple and unambiguous after the first shock.

(b) Arrays

A variable name in APL can just as well stand for an array as for a scalar. The members of an array may be either all numbers or all characters (letters, digits and other symbols of the language); but character arrays will be touched on here only very lightly. No initial dimension statement is made, and the same variable name may stand for arrays of different sizes at different times in an extended computation.

The permitted kinds of array are (i) a vector, (ii) a rectangular matrix, (iii) a rectangular array in three or more dimensions. Sharp distinctions are made between types of array. Whereas a scalar is a single item, not indexed, a vector is a list or concatenation of items indexed by one indexing variable (or subscript) taking consecutive positive integer values; a matrix is a set indexed by two such indexing variables; and so on. A single number is ordinarily taken to be a scalar, though for special purposes it can be made into a vector of one component, or a matrix with one row and one column, etc. (Such a change of status is usually understood if the context demands it.) Similarly, a simple list of numbers is ordinarily taken to be a vector, though if we wish we can make from it a matrix having either one row or one column, these being the "row-vector" or "column-vector" familiar in matrix algebra. Vectors in APL are neither row-vectors nor column-vectors, but just vectors!

Concatenation is denoted by a comma. Thus "1, 2, 3" denotes a vector with three components. If V denotes either a vector or a scalar then "V, 5" denotes the vector consisting of V followed by 5. In ordinary mathematical notation, the individual components of a vector V might be denoted by V_1, V_2, ..., using subscripts. In the present implementation of APL, since subscripting is not conveniently feasible, the indexing variable is placed in square brackets, and the components are written V[1], V[2], ... To select a subset of components, indexed by a vector of index values, we may write V[1, 3, 5], for example, or V[I], where I is any vector of index values.

The two indexing variables of a matrix are separated by a semicolon (a comma would not do, for it would denote concatenation). Thus M[2;1] is the element in the second row and first column of a matrix M. If I and J are vectors of index values, M[I;J] is the submatrix of M consisting of the intersection of rows I and columns J. M[I;] is the submatrix of complete rows I, and M[;J] is that of complete columns J. If I and J are scalars, M[I;] and M[;J] are vectors.

Two special notations for vectors need to be noticed. (i) An iota followed by a positive integer N stands for the vector of the first N positive integers, (1, 2, ..., N). Iota followed by 0 stands for an empty vector. (ii) A vector of numbers may be typed with the numbers separated by spaces instead of commas, in which case the numbers <u>will be understood to be tied</u> as though enclosed in parentheses.

Rho (ρ) stands for a function, either monadic or dyadic, concerned with the size of arrays. As a monadic function, ρ followed by an array name, say A, is a vector expressing the size of A. If A is a vector, ρA is a vector of length 1, whose member is equal to the number of members in A. If A is a matrix, ρA is a vector of length 2, the number of rows of A followed by the number of columns of A. If A is a vector, ρρA is equal to 1. If A is a matrix, ρρA is equal to 2. Similarly for arrays of higher dimension. If A is a scalar, ρA is empty and ρρA is equal to 0 (interpretations that are consistent with the foregoing, though perhaps surprising).

As a dyadic function, ρ preceded by a vector A and followed by a scalar or vector B is an array whose size is A, formed from members of B concatenated with itself if necessary, i.e. (B, B, B, ...). ρAρB is equal to A. See Figure 2, lines 11 and 18. This dyadic ρ permits any matrix or higher-dimensional array to be entered into the workspace from the keyboard. The coefficients are first entered as a vector, and then the vector is restructured into the desired array by the ρ function.

Now at last we come to the application of the standard scalar functions (such as were illustrated in Figure 1) to arrays. There are several different ways in which such functions can be applied to arrays, and these must be distinguished. All the standard scalar functions are treated exactly alike -- a remarkable feature that separates APL from other computer languages and systems.

```
        ¯1+3                          1.  (Negative 1) plus 3.
2                                     2.
        -1+3                          3.  0-(1+3).
¯4                                    4.  Negative 4.
        ⎕←V←2×⍳4                      5.  V is 2×(1,2,3,4).
2  4  6  8                            6.
        V[1 3 1 1]                    7.  Select some members of V.
2  6  2  2                            8.
        (⍴⍴V),⍴V                      9.
1  4                                  10.
        ⎕←M←2 3⍴V                     11. Use members of V to form a 2×3
                                          matrix M.
                                          Or:  ⎕←M←(2,3)⍴V

  2  4  6                             12. Matrix printed with indentation
                                          and line skip.
  8  2  4                             13.
        (⍴⍴M),⍴M                      14.
2  2  3                               15. That is:  2, (2, 3).
        M[;1]                         16. Display first column of M.
2  8                                  17. A vector, not a "column vector".
        10⍴1                          18. Display a vector of ten ones.
1  1  1  1  1  1  1  1  1  1          19.
        ⎕←A←(0.01*N)÷!N←0,⍳5          20. First 6 terms of series for exp(0.01).
1  0.01  5E¯5  1.666666667E¯7  4.166666667E¯10  8.333333333E¯13     21.
        +/A                           22. Sum of these terms.
1.010050167                           23.
        -/A                           24. Sum with alternating signs to
                                          approximate exp(-0.01).
0.9900498337                          25.
        N!5                           26. Binomial coefficients.
1  5  10  10  5  1                    27.
        (⍳3)∘.≤⍳5                     28. An outer product.
 1 1 1 1 1                            29.
 0 1 1 1 1                            30.
 0 0 1 1 1                            31.
```

FIGURE 2. Continuation of Figure 1.

First let us consider applications that leave the array size unchanged. If f is a standard monadic scalar function and A is an array, fA is an array of the same size as A obtained by applying f to each member of A. If f is a standard dyadic scalar function, and if A and B are arrays of the same size (so that ρA is equal to ρB), then AfB is an array of the same size as each of A and B, obtained by applying f to pairs of corresponding members. The expression AfB retains a meaning if one argument is an array but the other is scalar; the scalar is understood to be repeated to form an array of the same size as the other argument, and then f is applied to pairs as before. These usages are all illustrated in line 20 of Figure 2. We see that the vector (0, 1, 2, 3, 4, 5) is labeled N. The factorial sign in front of N gives the vector of factorials of the members of N, that is

$$(1,\ 1,\ 2,\ 6,\ 24,\ 120).$$

The expression in parentheses stands for the scalar 0.01 raised to the power of the members of N, that is the vector

$$(1,\ 10^{-2},\ 10^{-4},\ 10^{-6},\ 10^{-8},\ 10^{-10}).$$

These two vectors are divided, term by term, to give the vector

$$\left(\frac{1}{1},\ \frac{10^{-2}}{1},\ \frac{10^{-4}}{2},\ \frac{10^{-6}}{6},\ \frac{10^{-8}}{24},\ \frac{10^{-10}}{120}\right),$$

which is labeled A and displayed. See also lines 5 and 26.

Another way of applying a standard dyadic scalar function to an array is to compress it, reducing its dimensionality by one. Suppose that A is a vector. Then f/A stands for

A[1]fA[2]f ... fA[ρA] ,

where as usual execution is from right to left. Thus +/A is the sum of all the elements of A, and ×/A is the product. (See lines 22 and 24 of Figure 2.) A familiar example of term-by-term multiplication followed by sum compression (scalar product of vectors) is the evaluation of a one-dimensional integral by a quadrature method such as Simpson's rule or a Gaussian formula. Let Y denote the vector of ordinates and W the vector of corresponding weights. Then the desired result is

+/Y×W.

If A is a matrix or higher-dimensional array, the coordinate to be compressed must be specified, and that is done by "subscripting" the compression symbol /, by adding the coordinate number in square brackets. If A is a matrix, +/[1]A means the vector of sums over the rows, that is, the column totals; whereas +/[2]A means the sums over columns, or the row totals. The latter may also be written +/A, with the understanding that if the / is not subscripted it relates to the last coordinate of the array A.

A standard dyadic scalar function f can be applied to two arrays A and B to form an "outer product" of size (ρA), ρB; f is applied to every possible pair consisting of a member of A and a member of B. When f is multiplication and A and B are vectors we obtain the ordinary matrix outer product. The notation for outer product (small circle followed by period followed by f) is shown at line 28 of Figure 2, where f is the function $\leq$. If f had been ×, this outer product would have been

```
1   2   3   4   5
2   4   6   8  10
3   6   9  12  15
```

and if f had been +, the outer product would have been

```
2   3   4   5   6
3   4   5   6   7
4   5   6   7   8
```

Much use is made of outer products, as we shall see.

The familiar inner product of matrix algebra involves two standard dyadic scalar functions, multiplication and addition, and is denoted by A+.×B. Here A and B are arrays such that the last member of ρA is equal to the first member of ρB. More generally, an "inner product" Af.gB can be formed using any two standard dyadic scalar functions f and g. If A and B are both vectors of the same length, the inner product A+.×B is just the ordinary scalar product, +/A×B.

This is an appropriate point to comment on the generality of definitions in APL. Symbols are introduced to meet common needs, but they are defined broadly, so that many possibilities are brought in "for free", some of which turn out to be useful. Thus any language designed to handle matrices must obviously encompas ordinary matrix addition. In APL we express the sum of two matrices A and B of the same size by writing A + B. But this notation does not relate specially either to <u>matrices</u> or to <u>addition</u>. For as we have seen, A and B may be rectangular arrays of any size and dimensionality (the same for A and B), or one may be an array and the other a scalar; moreover, + may be replaced by any other standard dyadic scalar function. Many of these possibilities are just as useful as the ordinary matrix sum. And similarly for the other array operations just described, especially the "outer product". Because of this generality in the definitions, statements in APL often have a high degree of ambiguity (as, for example, the statement "C left-arrow A + B"). The statement acquires a precise meaning from its context. Superfluous information -- information that has previously been given or implied -- is not required. If the size of A has been previously determined, it need not be mentioned when the sum A + B is called for. In APL, as in ordinary English, reliance on context leads to economy of expression. Occasionally ambiguity is deliberately exploited. For example, an argument in a regression program could be a variable Y, standing <u>either</u> for a vector of observations <u>or</u> for a matrix formed from several observation vectors of equal length set side by side which we wish to analyse simultaneously in parallel. With a little care in writing, the program may run equally well in either case.

We have now met most of the symbols likely to be useful in statistical calculations. Rather than continue to illustrate individual usages as in Figures 1 and 2, it will be more interesting to show a few small examples of statistical calculation. We begin with direct "impromptu" dialog between operator and machine, and then pass on to defined functions, which permit carefully considered programs to be stored and used repeatedly.

These illustrations are <u>not</u> intended to represent the present art of statistical analysis. For one thing, scatterplotting is not shown, although the ease with which this can be done is possibly the most valuable gift of the computer to statistics. The illustrations are intended merely to suggest how a statistician may apply an APL terminal to some common tasks.

3. ILLUSTRATIONS

(a) <u>Dialogs</u>

Figure 3 relates to observations of the number of heads showing when ten coins have been thrown down. Twenty-five such observations were made by students in an introductory statistics course. Let us see whether the readings conform with the usual theoretic idea of independent binomial variables with probability of heads equal to one half.

The twenty-five readings are entered and named Y at lines 1 and 2, and counted (lines 3 and 4). Each reading must necessarily have one of the eleven values (0, 1, 2, ..., 10). The frequencies of each of these values is called for at line 5; an outer product of the value vector and Y, using the function =,

```
      Y←6 7 7 3 4 8 5 7 3 2 6 6 6                    1.  Load first 13 readings.
      Y←Y,4 5 7 2 2 5 6 4 7 5 5 3                    2.  Concatenate remaining readings.
      ρY                                             3.  Check how many.
25                                                   4.  (Good.)
      ⎕←OF←+/(0,⍳10)∘.=Y                             5.  Calculate observed frequencies.
0  0  3  3  3  5  5  5  1  0  0                      6.
      YBAR←(+/Y)÷ρY                                  7.  Sample mean.
      VAR←(+/(Y-YBAR)*2)÷¯1+ρY                       8.  Sample variance.
      YBAR,VAR,VAR*0.5                               9.  Display sample mean, variance
                                                         and standard deviation.
5  3.166666667  1.779513042                         10.
      5,2.5,2.5*0.5                                 11.  Theoretical values for
                                                         comparison.
5  2.5  1.58113883                                  12.
      (⌊/Y),⌈/Y                                     13.  Least and greatest members of Y.
2  8                                                14.
      0.01×⌊0.5+100×EF←(25×2*¯10)×(0,⍳10)!10        15.  Calculate expected frequencies.
0.02  0.24  1.1  2.93  5.13  6.15  5.13  2.93  1.1  0.24  0.02     16.  Displayed
                                                                        after rounding.
      EF←A,EF[5 6 7],A←+/EF[⍳4]                     17.  Do some grouping of EF.
      OF←(+/OF[⍳4]),OF[5 6 7],+/OF[12-⍳4]           18.  Ditto for OF.
      (+/EF),+/OF                                   19.  Check the sums.
25  25                                              20.  (Good.)
      2 5ρOF,EF                                     21.  Display as a matrix.

                                                    22,23.
     6               3               5               5               6
     4.296875        5.126953125     6.15234375      5.126953125     4.296875
      ⎕←+/⎕←((OF-EF)*2)÷EF                          24.  Calculate chi-squared.
0.6750568182  0.8823816964  0.2158358135  0.00314360119  0.6750568182   25.  Individual
                                                                             terms.
2.451474747                                         26.  Sum of terms.
```

FIGURE 3. Numbers of heads in 25 flips with 10 coins.

is calculated, and then the eleven row totals are stored under the name OF (for observed frequencies) and displayed at line 6. At line 7 we calculate the sample mean and at line 8 the sample variance. Of course we know that ρY is equal to 25, so that we could just as well have specified 25 as the divisor in line 7 and 24 as divisor in line 8. At line 9 we ask for a display of these quantities and also the sample standard deviation. The mean, variance and standard deviation of the supposed binomial distribution governing the observations are noted at lines 11 and 12 for comparison -- agreement is satisfactory. At line 13 we ask for the least and the greatest of the twenty-five readings (the answer is already implied by the observed frequencies at line 6, but we see how this information can be asked for directly by compressing Y).

We now do an ordinary χ^2 goodness-of-fit test of the observed frequencies to the supposed binomial expectations. The expected frequencies (EF) are called for, and displayed after rounding to two decimal places. We notice that these expectations are low at both ends of the sequence. Following the old rule of thumb that expectations should not be much less than 5 for use in a χ^2 test, we group the first four and last four members of EF (line 17) and OF (line 18), and reassign the names EF and OF to these grouped frequencies. Now we wonder if we did that right, so we check that both the expected and the observed grouped frequencies add up to 25 (lines 19 and 20) and then we display them in a 2 × 5 matrix. Finally we calculate the χ^2 measure of goodness of fit, obtaining individual terms (line 25) and their sum (line 26). The latter value of 2.45 is entirely unremarkable in relation to the standard probability distribution tabulated as χ^2 with 4 degrees of freedom. Observations seem to agree perfectly with theory.

Figure 4 relates to another set of observations made by students in the introductory course. The students were asked to shuffle an ordinary pack of 52 playing cards, deal a hand of 13 cards and note the numbers of cards in each of the four suits, clubs, diamonds, hearts, spades. Each student did this several times, and in all 60 hands were observed. The observations were written down on a sheet of paper in four columns, sixty rows, like this:

C	D	H	S
2	5	5	1
3	6	3	1
2	4	6	1

```
      M←2  5  5  1  3  6  3  1  2  4  6  1  4  4  4  1  3  3  3  4  6  0  3  4           1.
      M←M,4  5  2  2  4  3  2  4  4  2  3  4  3  5  2  3  3  4  4  2  3  2  4  4         2.
      M←M,4  3  1  5  3  4  3  3  4  2  6  1  2  4  3  4  4  5  2  2  5  1  4  3         3.
      M←M,1  4  3  5  3  3  4  3  4  4  1  4  1  3  3  6  4  5  4  0  4  1  5  3         4.
      M←M,2  2  4  5  3  1  7  2  5  5  1  2  3  5  1  4  3  3  3  4  1  6  4  2         5.
      M←M,2  4  4  3  7  0  2  4  3  4  1  5  6  2  2  3  1  3  6  3  3  4  4  2         6.
      M←M,4  2  5  2  3  3  2  5  3  4  3  3  3  4  3  3  3  6  2  2  2  4  2  5         7.
      M←M,5  2  4  2  4  2  5  2  2  4  4  3  5  6  0  2  8  3  1  1  5  4  2  2         8.
      M←M,4  3  5  1  3  3  5  2  4  2  2  5  5  4  2  2  2  2  2  7  3  5  4  1         9.
      M←M,5  2  3  3  1  3  4  5  3  5  2  3  4  4  4  1  2  5  4  2  3  4  3  3        10.

      M←60 4⍴M                                                                         11.

      M[1 2 3 58 59 60;]                                                               12.


  2  5  5  1                                                                           13.

  3  6  3  1                                                                           14.

  2  4  6  1                                                                           15.

  4  4  4  1                                                                           16.

  2  5  4  2                                                                           17.

  3  4  3  3                                                                           18.

      +/13≠+/M                                                                         19.

0                                                                                      20.

      +/⎕←+/[1]M                                                                       21.

205  207  192  176                                                                     22.

780                                                                                    23.

      CS←(16÷39)×+/(M-3.25)*2                                                          24.

      CSBAR←(+/CS)÷60                                                                  25.

      CSBAR,(+/(CS-CSBAR)*2)÷59                                                        26.

3.48034188  7.24514128                                                                 27.

      ⎕←COF←+/0.58 1.21 2.37 4.11 6.25∘.≥CS                                            28.

7  15  20  43  54                                                                      29.

      ⎕←OF←(COF,60)-0,COF                                                              30.

7  8  5  23  11  6                                                                     31.

      EF←6 9 15 15 9 6                                                                 32.

      +/((OF-EF)*2)÷EF                                                                 33.

11.65555556                                                                            34.
```

FIGURE 4. 60 hands from a pack of playing cards. Explanation in the text.

and so on, finishing

4	4	4	1
2	5	4	2
3	4	3	3

The observations were intended to illustrate the theory of sampling a finite population.

In lines 1 to 10 of Figure 4 the observations are entered from the data sheet, row by row, as a vector M of length 240. In line 11, M is restructured as a 60 × 4 matrix. As a check on the restructuring, the first three and last three rows are asked for at line 12, for comparison with the data sheet. At line 19, a check is made that every row of M sums to 13 (the requested size of the hand) -- the number of rows in which the row sum is not equal to 13 is counted and (line 20) found to be 0. The four column totals of M and their sum are asked for at line 21. We see that the sum (780) is equal to 13 × 60, as it should be.

At this point in the original treatment of the data, further checks of accuracy in the entering of M were made. The data were entered afresh; each column of the data sheet was entered as a vector and compared with the corresponding column of M. We omit this check now for brevity.

From each row of M a 2 × 4 contingency table can be constructed showing the compositions of the hand and of the balance of the pack. Thus from the first row of M we have

	C	D	H	S
Hand	2	5	5	1
Balance of pack	11	8	8	12

Our purpose is to compare the empirical and the theoretical behaviors of some measure of independence in the sixty such tables. The measure chosen (arbitrarily) was the usual χ^2 criterion. The expected frequencies in each table are

3.25	3.25	3.25	3.25
9.75	9.75	9.75	9.75

and for the first table, quoted above, the χ^2 value is

$$\left(\frac{1}{3.25} + \frac{1}{9.75}\right) \{(-1.25)^2 + (1.75)^2 + (1.75)^2 + (-2.25)^2\} .$$

The multiplier in front of the sum of squares reduces to (16 ÷ 39).

At line 24 of Figure 4 the χ^2 corresponding to each row of M is called for, under the name CS, a vector of 60 components. The mean and variance of the sixty values is found (lines 25-27), for comparison with the values 3 and 6 respectively for the standard probability distribution tabulated as χ^2 with 3 degrees of freedom.

We now proceed to compare the frequency distribution of the members of CS with this tabulated probability distribution. We must form a list of grouped observed frequencies. Reference to a table of the χ^2 distribution with 3 degrees of freedom gave the following approximate quantiles: median (2.37), lower and upper quartiles (1.21, 4.11), lower and upper 10% points (0.58, 6.25). Cumulative observed frequencies (COF) of members of CS up to these five values are called for at line 28. (It is unnecessary to quote the five values with greater precision, because the actual value set of the χ^2 values is discrete and none is close to any of these theoretical quantiles -- as is readily demonstrated by displaying the whole of CS, though this is not done in Figure 4.) The cumulative observed frequencies are differenced at line 30 to yield the observed frequencies: 7 values of χ^2 are below 0.58, 8 are between 0.58 and 1.21, 5 are between 1.21 and 2.37, ..., 6 are above 6.25. The expected frequencies for the tabulated χ^2 distribution with 3 degrees of freedom are quoted at line 32 (they are 60 multiplied by 10%, 15%, 25%, ...). The ordinary χ^2 measure of agreement between lines 31 and 32 is called for at line 33. The value obtained, 11.66, comes at roughly the upper 4% point of the tabulated χ^2 distribution with 5 degrees of freedom. Whether one asserts that the discrepancy between observations and theory is "significant" depends on temperament and philosophy.

In fact, the observed behavior of the χ^2 values could differ from the so-called χ^2 distribution for two reasons other than mere chance: (i) the shuffling of the pack and counting were perhaps not always well done; (ii) even if shuffling and counting were perfect, the true probability distribution of the χ^2 criterion, under the specified procedure of observation, is discrete and only represented roughly by the continuous tabulated distribution. In further study of the data this true discrete distribution was determined and found to agree very satisfactorily with the observed frequencies.

(b) Programs

Let us turn now to analysis of variance. We begin with the same kind of "impromptu" dialog as above, and then introduce some stored programs.

As a simple example, we take a 4 × 6 table quoted by Bliss [1], relating to an experiment in 6 randomized blocks comparing the heights of loblolly pines grown from seed from 4 different sources. The reading obtained from each of the 24 plots is the average height of surviving trees after fifteen years in the plantation.

In the first two lines of Figure 5 the data are entered under the name Y , a 4 × 6 matrix, and then immediately displayed in lines 3-6. In line 7 the grand mean is calculated, displayed (at line 8), labeled GM, and subtracted from each member of Y to make a matrix RY of residuals. The members of RY are squared and summed to give the total sum of squares about the mean (TSS). In line 9 are displayed TSS, its number of degrees of freedom (23), and the corresponding mean square.

Row effects are calculated at line 10. The row means of RY are found, displayed (at line 11), and labeled RE for "row effects". The members of RE are squared and summed and multiplied by 6 to give the sum of squares for rows (SSR). In line 12 are displayed SSR , its number of degrees of freedom (3) and the corresponding mean square.

Column effects are similarly calculated at line 13. It will be noted that the initial summing of RY is over its first coordinate to yield column totals; otherwise line 13 looks similar to line 10. Answers are displayed in lines 14 and 15.

In line 16 the residuals after subtraction of row and column effects are calculated. First an "open product" of the vectors CE and RE , using the function + , is formed and subtracted from RY to yield the desired matrix of residuals, which is named RY again, so that the meaning of RY changes at this point. The members of the new RY are squared and summed to yield the residual sum of squares RSS . This is displayed in line 17, together with the number of degrees of freedom (15) and the corresponding mean square.

At this point we can write out the following analysis of variance table:

	Sum of squares	D.f.	Mean square
Columns (blocks)	17.16	5	3.43
Rows (seed source)	171.36	3	57.12
Residual	22.00	15	1.47
Total about mean	210.51	23	9.15

We may now do various things with the residuals RY . They may be displayed, and a scatter plot can be made of members of RY against the corres-

```
      Y←34.0 29.3 30.6 31.8 34.0 32.7 27.3 27.6 28.6 29.2 30.2 31.5 26.4 25.0       1.
      ⎕←Y←4 6 ρY, 26.6 25.2 27.4 26.2 24.8 24.3 26.0 26.5 25.8 24.2                  2.

                                                                                    3-6.
 34             29.3            30.6            31.8            34              32.7
 27.3           27.6            28.6            29.2            30.2            31.5
 26.4           25              26.6            25.2            27.4            26.2
 24.8           24.3            26              26.5            25.8            24.2
      TSS, 23, (÷23)×TSS←+/+/RY×RY←Y-GM←⎕←(+/+/Y)÷24                                 7.
28.13333333                                                                          8.
210.5133333  23  9.152753623                                                         9.
      SSR, 3, (÷3)×SSR←6×+/RE×RE←⎕←(+/RY)÷6                                         10.
3.933333333  0.9333333333  ¯2  ¯2.866666667                                         11.
171.36  3  57.12                                                                    12.
      SSC, 5, (÷5)×SSC←4×+/CE×CE←⎕←(+/[1]RY)÷4                                      13.
¯0.008333333333  ¯1.583333333  ¯0.1833333333  0.04166666667  1.216666667  0.5166666667  14.
17.15833333  5  3.431666667                                                         15.
      RSS, 15, (÷15)×RSS←+/+/RY×RY←RY-RE∘.+CE                                       16.
21.995  15  1.466333333                                                             17.
      RSS1←RSS-⎕←A×⎕←(A←+/+/RY×RE∘.×CE)×24÷SSR×SSC                                  18.
0.1838114556                                                                        19.
4.139219534                                                                         20.
      RSS1, 14, RSS1÷14                                                             21.
17.85578047  14  1.27541289                                                         22.


    ∇ LOBLOLLYDATA
[1]   Y← 34 29.3 30.6 31.8 34 32.7 27.3 27.6 28.6 29.2 30.2 31.5 26.4 25
[2]   Y← 4 6 ρY, 26.6 25.2 27.4 26.2 24.8 24.3 26 26.5 25.8 24.2
[3]   '4×6 MATRIX (Y) OF AVERAGE HEIGHTS OF LOBLOLLY PINES IN FEET'
[4]   'SEED FROM 4 SOURCES (ROWS), 6 RANDOMIZED BLOCKS (COLUMNS)'
[5]   'WAKELY, 1944, QUOTED BY BLISS, VOL. I, TABLE 11.7.'
    ∇
```

FIGURE 5. Analysis of a two-way table.

ponding fitted values (Y - RY). At line 18 we ask for calculation of Tukey's nonadditivity test. Inside the parenthesis, the open product of RE with CE , using multiplication, is formed, multiplied term by term with RY and summed; the sum is named A . The sum of squares of the members of the open product of RE and CE is easily seen to be equal to SSR times SSC , divided by 24. Thus what is displayed at line 19 is the regression coefficient of the members of RY on the corresponding products of RE and CE . This coefficient is multiplied by A to give Tukey's 1 degree of freedom term, displayed at line 20. This is subtracted from RSS to give a new residual sum of squares, labeled RSS1 . At line 22 the latter is displayed, together with the number of degrees of freedom (14) and the corresponding mean square. Thus the residual line in the above analysis of variance has been decomposed into

Tukey's term	4.14	1	4.14
Residual	17.86	14	1.28

Evidence of nonadditivity is only weak -- as is not surprising with so few residual degrees of freedom.

If we were to consider either repeating the above analysis after (say) taking logarithms or reciprocals of Y , or performing similar analyses on other sets of data arranged in a row-column crossclassification, we might like to store the basic procedure. We might also like to store the data, in case we should want to do something with this material on another occasion. Programs are stored by the device of defining a function, and then having the definition copied into a storage space for future reference.

There are several possible syntaxes for defined functions. The function may have either no or one or two explicit arguments, that have to be specified when the function is called. (A defined function may also have any number of concealed arguments, since the program may refer to variable names that need to have been defined before the function was called. For example, the program might refer to a number named PI , not defined in the program; and then provided we have previously given PI a value, such as perhaps 3.14159, the program will be able to calculate with this value.) A defined function may have either no or one explicit range variable or result, together with various other sorts of output, namely displays and stored material. The one or two explicit arguments and the one explicit result may be scalars or arrays. The examples will show some of these possibilities.

First let us store the loblolly pine data. We may define a function LOBLOLLYDATA with no arguments of any kind and no explicit result as output. This function will have two items of output: (i) a typed statement giving a little information, in case we should call for this function at a later date when we have forgotten just what it is, and (ii) the stored matrix Y . To open the definition of the function we type an upside-down delta and the name of the function. The following rows are automatically numbered on the left in square brackets. After each row has been typed and the release key pressed, the instruction is stored but not executed. The definition is terminated by typing another upside-down delta. Our first two lines of LOBLOLLY-DATA repeat what we did at lines 1 and 2 at the top of Figure 5. Lines 3-5 are in quotes. Each of these lines will be read as a character vector, and since it is not assigned anywhere it will be printed out (without quotes) when the function is executed. Note that the information given includes the name under which the data matrix is stored, namely Y .

Now let us store the procedure for the analysis of variance. The big question here is the degree of generality to be attempted. We might as well make the program refer to any row-column crossclassification, not necessarily of size 4 × 6. It will be convenient to have the output labeled. If we are more ambitious we may consider a program to handle data in a rectangular array of any number of dimensions, with various possibilities for estimating cross main effects and interactions. But first let us try a simple two-dimensional crossclassification.

We shall name the function ROWCOL and let it have one explicit argument, the matrix of readings Y . Like LOBLOLLYDATA , this function will have no explicit result as output, but will have both displayed and stored output. A possible program for ROWCOL is shown in Figure 6.

Whereas most other computer languages have a considerable vocabulary relating to the flow of an algorithm -- specifying conditional branches, loops, etc. -- APL has precisely one symbol for branching, the right-arrow (→). Nothing appears to the left of this arrow (except for the statement number in square brackets), and to the right appears an expression having (usually) a nonnegative integer value. The arrow means "go to the statement numbered" -- for example, "→ 3" means "go to the statement numbered 3". If

```
    ∇ ROWCOL Y
[1]    →3×ι(×/2≤ρY)×2=ρρY
[2]    →0,ρ⎕←'NO GO.'
[3]    ('GRAND MEAN (GM) IS ';GM←(+/+/Y)÷×/ρY)
[4]    'TOTAL SUM OF SQUARES ABOUT MEAN (TSS), DEGREES OF FREEDOM AND MEAN SQUARE ARE'
[5]    TSS,(¯1+×/ρY),(÷¯1+×/ρY)×TSS←+/+/RY×RY←Y-GM
[6]    '    *ROWS*'
[7]    ('EFFECTS (RE) ARE ';RE←(+/RY)÷(ρY)[2])
[8]    ('SUM OF SQUARES (SSR), D.F. AND MEAN SQUARE ARE ';SSR,(¯1+(ρY)[1]),(÷
       ¯1+(ρY)[1])×SSR←(ρY)[2]×+/RE×RE)
[9]    '    *COLUMNS*'
[10]   ('EFFECTS (CE) ARE ';CE←(+/[1]RY)÷(ρY)[1])
[11]   ('SUM OF SQUARES (SSC), D.F. AND MEAN SQUARE ARE ';SSC,(¯1+(ρY)[2]),(÷
       ¯1+(ρY)[2])×SSC←(ρY)[1]×+/CE×CE)
[12]   '    *RESIDUALS*'
[13]   ('SUM OF SQUARES (RSS), D.F. AND MEAN SQUARE ARE ';RSS,(×/¯1+ρY),(÷×/¯1+ρY)×RSS
       ←+/+/RY×RY←RY-RE∘.+CE)
[14]   →0,ρ⎕←'THE MATRIX OF RESIDUALS IS NAMED RY.'
[15]   COMMENT--THIS PROGRAM PERFORMS A STANDARD ANALYSIS OF VARIANCE ON A ROW-COLUMN
       CROSSCLASSIFICATION.
    ∇

       LOBLOLLYDATA

4×6 MATRIX (Y) OF AVERAGE HEIGHTS OF LOBLOLLY PINES IN FEET
SEED FROM 4 SOURCES (ROWS), 6 RANDOMIZED BLOCKS (COLUMNS)
WAKELY, 1944, QUOTED BY BLISS, VOL. I, TABLE 11.7.

       ROWCOL 1000÷Y

GRAND MEAN (GM) IS 35.92360518
TOTAL SUM OF SQUARES ABOUT MEAN (TSS), DEGREES OF FREEDOM AND MEAN SQUARE ARE
313.1445502  23  13.61498044
    *ROWS*
EFFECTS (RE) ARE ¯4.646848272  ¯1.434914258  2.379661791  3.702100739
SUM OF SQUARES (SSR), D.F. AND MEAN SQUARE ARE 258.1231075  3  86.04103583
    *COLUMNS*
EFFECTS (CE) ARE 0.137187286  1.954854887  0.001469058796  ¯0.1457289384
      1.478508235  0.4692740584
SUM OF SQUARES (SSC), D.F. AND MEAN SQUARE ARE 25.07088723  5  5.014177446
    *RESIDUALS*
SUM OF SQUARES (RSS), D.F. AND MEAN SQUARE ARE 29.95055547  15  1.996703698
THE MATRIX OF RESIDUALS IS NAMED RY.

       RSS1←RSS-⎕←A×⎕←(A←+/+/RY×RE∘.×CE)×24÷SSR×SSC
¯0.07821979588
1.649752151

       RSS1, 14, RSS1÷14
28.30080332  14  2.021485952
```

FIGURE 6. Continuation of Figure 5.

there is no statement in the program having the indicated number, the command is interpreted as "stop". In particular, since the statements are numbered from 1 upwards, the statement "→ 0" always means "stop". What appears to the right of the arrow is not always a number (scalar); it may be a vector. If V is a non-empty vector, "→ V" means "go to the statement whose number is the first member of V". If V is an empty vector, the command "→ V" means "go to the next statement", as though the next statement number had been automatically concatenated onto V . A branch is made conditional by the device of computing the statement number to be gone to.

Let us now consider the program for ROWCOL . The name of the argument (Y) is a dummy variable in the program. When we call the program the argument may have any name.

The first statement of ROWCOL is a conditional branch. It means: "If the argument (Y) is a matrix having at least 2 rows and 2 columns go to statement 3; otherwise go to the next statement (2)." Statement 2 calls for the phrase "NO GO." to be typed out and then execution is stopped. In fact statement 2 could have been more simply written as the two statements:

```
[2]  'NO GO.'
[3]  → 0
```

and then the following statements would have been numbered 1 higher. In the compressed version shown in Figure 6, the character vector "NO GO." is first displayed, then counted (there are 6 characters including the space and the period), and then the statement reads "→ 0, 6", which means the same as "→ 0" or "stop". Statements 1 and 2 have been put in merely to prevent accidental application of the function to an argument that is not a suitable two-way table.

Statement 3 corresponds to the last bit of line 7 in Figure 5. Mixed output is called for, namely a character vector followed by a computed quantity. The semicolon in the middle and the outer parentheses are mandatory when such mixed output is specified. Statement 4 is a simple character vector to be typed out. Statement 5 corresponds to the rest of line 7 in Figure 5. Statement 6 calls for an indented heading. Statements 7 and 8 correspond to line 10 of Figure 5. (Note that statements 8, 11, 13 and 15 have overflowed in this print-out onto a second line. With wider paper they could be printed on one line.) Similarly for statements 9-11, corresponding to line 13 of Figure 5. Statements 12 and 13 correspond to line 16 of

Figure 5. Finally, statement 14 calls for a note about the matrix of residuals to be typed out (so that the user will know what name this has been stored under) and then execution is stopped. Statement 15 is never executed, because execution of the program must necessarily stop either at statement 2 or at statement 14. Thus statement 15 is entirely useless baggage when the program is executed; its only virtue is as a note to the reader if a print-out of the program is called for.

In the lower part of Figure 6, we see LOBLOLLYDATA called. (Single rather than double spacing has been used in the printing here.) Then ROWCOL is called. If we had called ROWCOL Y , the material of Figure 5 would have been repeated. Instead, we have used as argument the reciprocals of the original Y-values (multiplied by 1000). Execution terminates with the remark about the matrix of residuals. (An unfortunate consequence of the single-spaced printing is that the high-minus signs of the last two members of CE look like underscoring of the line above.) We have then gone back to the "impromptu" mode and called for Tukey's test again. The one-degree-of-freedom term is now a little smaller than the residual mean square.

This function ROWCOL involves only very simple computation. Procedures for regression tend to be more interesting computationally, but they raise a variety of issues concerning objectives that are not too appropriate for discussion in the present context. We shall content ourselves by illustrating some possibilities of handling multidimensional arrays in APL.

Figure 7 presents a function called POWERS that calculates all powers of a given square matrix from the zeroth to the Nth, for some given N . The matrices are stacked in a three-dimensional array, which however is printed out layer by layer as a sequence of matrices. Such powering of matrices is of interest in the study of Markov chains.

The function has two explicit arguments, the matrix (M) to be powered and the highest power (N) to be taken. The function also has an explicit result, the stack (Z) of powers of M . Z , M and N are dummy variables in the program. They may be replaced by any other variable names when we call the function. The program contains a loop, controlled by an index variable J . We have no use for J outside the program, so J has been declared to be a dummy (local) variable in the program by the device of adding "; J" at the end of the function header.

```
    ∇ Z←M POWERS N;J
[1]    →3-(=/ρM)×(2=ρρM)×0=ρρN
[2]    →4×ι(N=⌊N)×1≤N
[3]    →0,ρ⎕←'NO GO.'
[4]    Z←((N+1),ρM)ρ1,(ρM)[1]ρ0
[5]    J←1
[6]    Z[J+1;;]←Z[J;;]+.×M
[7]    →6×N≥J←J+1
[8]    COMMENT--ALL POWERS FROM ZEROTH TO N-TH OF A SQUARE MATRIX 'M' ARE STACKED IN A
       3-D ARRAY.
    ∇

       P←3 3 ρ0 0.95 0.05 1 0 0 0 1 0
       PP←P POWERS 20
       PP[1 2 3 19 20 21;;]
   1                 0                 0
   0                 1                 0
   0                 0                 1

   0                 0.95              0.05
   1                 0                 0
   0                 1                 0

   0.95              0.05              0
   0                 0.95              0.05
   1                 0                 0

   0.6818934085      0.3039615833      0.01414500823
   0.2829001645      0.6818934085      0.03520642699
   0.7041285399      0.2829001645      0.01297129559

   0.3039615833      0.6619437463      0.03409467042
   0.6818934085      0.3039615833      0.01414500823
   0.2829001645      0.6818934085      0.03520642699

   0.6619437463      0.3228581746      0.01519807916
   0.3039615833      0.6619437463      0.03409467042
   0.6818934085      0.3039615833      0.01414500823
       (+/[1]PP)÷21
   0.5043028807      0.4720580318      0.02363908749
   0.4727817499      0.5043028807      0.02291536944
   0.4583073888      0.4727817499      0.06891086132
       (÷231)×(22-ι21)+.×PP
   0.5123351269      0.4643404025      0.02332447053
   0.4664894105      0.5123351269      0.02117546257
   0.4235092514      0.4664894105      0.1100013381
```

FIGURE 7.   Powering a transition matrix.

The first three statements of the program merely check whether N is scalar and M is a square matrix (statement 1) and also whether N is a positive integer (statement 2); if any of these conditions fails execution stops (statement 3). Real business begins at statement 4, where a three-dimensional array Z of the desired size is formed out of 1's and 0's. The size of Z is (N+1)×p×p , where M is p×p . The first layer of Z is a unit p×p matrix; all the other layers are nonsense, but we shall change them. Statement 5 initializes the loop variable J at the value 1. Statements 6 and 7 constitute the loop. Statement 6 says that the (J + 1)st layer of Z is to be replaced by the ordinary inner matrix product of the Jth layer and M . Statement 7 adds 1 onto J and returns execution to statement 6 until the (N + 1)st layer has been replaced, and then it becomes "→ 0". Statement 8 is a nonexecutable comment intended only for the reader of a print-out of the program.

Below the definition of POWERS we see an example of its use. A 3 × 3 transition matrix P is defined. The 21 × 3 × 3 stack of its powers from 0th to 20th is labeled PP . The first three and last three layers of PP are displayed; these matrices would usually be referred to as I, P, P^2, P^{18}, P^{19}, P^{20}. The average of all 21 layers of PP is then found, and finally the Cesaro mean of the 21 layers. The latter calculation involves the inner product of a vector of weights with the stack PP, an example of an inner product of non-matrices.

After this brief excursion into three-dimensionality, let us return to the generalization of our program ROWCOL . We shall consider the application of analysis of variance to a rectangular array in an arbitrarily large number of dimensions.

(c) Multidimensional arrays

The foregoing examples will have suggested that APL is beautifully adapted to expressing computations with vectors, matrices and three-dimensional arrays. Most computational problems arising in statistics are naturally expressed in terms of such arrays of small fixed dimensionality.

Rectangular arrays of arbitrarily high dimensionality are sometimes presented by multiply classified data. To preserve the multidimensional structure during the analysis is attractive, rather than to bury it in some kind of design matrix, although the latter procedure may be advisable if the data array is incomplete.

Of course, with enough trouble in the programming, any computation with multidimensional arrays can be expressed in any general-purpose computer language (even FORTRAN!). The interesting question is not what can be done but how easily it can be done. Arrays of arbitrary dimensionality are less easy to operate with in APL than arrays of given small dimensionality, especially vectors and matrices. Quite possibly some changes will be made in APL during the next year or two that will facilitate the handling of multidimensional arrays. As APL now is, we find ourselves making transpositions that permute the coordinates of such arrays, and performing various other kinds of restructuring, all of which are a bit troublesome to think through.

In Figure 8 we see two functions, ANALYZE and EFFECT, which together constitute a possible generalization of ROWCOL to arrays of arbitrary dimensionality. A slightly wider vocabulary is used here than in our previous programs.

We have already encountered the dyadic use of the function comma (,) for concatenation in building up a vector. In ANALYZE, statements 4 and 6, we see a monadic use of comma to destructure an array and reduce it to a vector. Otherwise, all symbols in ANALYZE are already familiar. The first two statements check that the argument Y has at least two dimensions and each member of ⍴Y is not less than 2. At statement 3 a scalar J is set equal to 1 and an array DF is set equal to a vector of 1's. J and DF are needed later in EFFECT; no use is made of them now. Statement 4 calculates and prints out the grand mean, and corresponds to statement 3 of ROWCOL. Note that because we have placed the comma before Y , so reducing Y to a vector, we are able to sum the whole of Y with a single "+/" instead of by as many of these as there are dimensions of Y . Similarly statements 5 and 6 correspond to statements 4 and 5 of ROWCOL. That is as far as ANALYZE takes us. Statements 7-9 give a little information to the user and then execution is stopped.

The purpose of the function EFFECT is to estimate a designated main effect or a designated interaction. The argument V is a list of one or more dimension (factor) numbers, just one for a main effect, two or more for an interaction. Only our understanding of the meaning of the dimensions of Y -- which dimensions correspond to crossed factors, which to nested classifications, and which of the latter is nested in which -- will inform us what

```
    ∇ ANALYZE Y
[1]    →3×⍳(×/2≤⍴Y)×2≤⍴⍴Y
[2]    →0,⍴⎕←'NO GO.'
[3]    DF←(1+⍴⍴Y)⍴J←1
[4]    ('GRAND MEAN (GM) IS ';GM←(+/,Y)÷×/⍴Y)
[5]    'TOTAL SUM OF SQUARES ABOUT MEAN (TSS), DEGREES OF FREEDOM AND MEAN SQUARE ARE'
[6]    TSS,NU,(÷NU←¯1+×/⍴Y)×TSS←+/,RY×RY←Y-GM
[7]    'PROCEED BY REPEATEDLY CALLING THE FUNCTION ''EFFECT''.'
[8]    'THE ARRAY OF RESIDUALS IS ALWAYS NAMED RY, WITH DEGREES OF FREEDOM NU.'
[9]    →0,⍴⎕←'RENAME SS EACH TIME TO SAVE IT.'
[10]   COMMENT--THIS PROGRAM BEGINS AN ANALYSIS OF VARIANCE OF A PERFECT CROSSED AND/OR
       NESTED ARRAY.
    ∇

    ∇ Z←EFFECT V;M;K;P;IND
[1]    →3-×/+/M←(,V)∘.=⍳⍴⍴RY
[2]    →4×⍳×/0≤IND←1-+/[1]M
[3]    →0,⍴⎕←'NO GO.'
[4]    DF←((J←J+1),1+⍴⍴RY)⍴(,DF),IND,0
[5]    NU←NU-DF[J;1+⍴⍴RY]←(×/(⍴RY)[V])-+/DF[;+/1+⍴⍴RY]×(+/DF[J;])=DF+.×DF[J;]
[6]    SS←K×+/,Z×Z←(÷K)×+/[1]((K←×/IND/⍴RY),(⍴RY)[V])⍴,((P←(IND/⍳⍴⍴RY),V)⍳⍳⍴⍴RY)⍉RY
[7]    RY←RY-P⍉((IND/⍴RY),(⍴RY)[V])⍴,Z
[8]    ('SUM OF SQUARES (SS), D.F. AND MEAN SQUARE ARE ';SS,DF[J;1+⍴⍴RY],SS÷DF[J;1+⍴⍴RY])
[9]    →0
[10]   COMMENT--'ANALYSE' SHOULD BE CALLED FIRST.
    ∇


       EGGDATA
6×2×2×2 ARRAY (Y) OF PERCENT FAT CONTENT OF DRIED WHOLE EGGS
NESTED SAMPLING: 6 LABORATORIES, 2 ANALYSTS, 2 SAMPLES, DUPLICATE DETERMINATIONS
MITCHELL, 1950, QUOTED BY BLISS, VOL. I, TABLE 12.1.
       ANALYZE Y
GRAND MEAN (GM) IS 42.0875
TOTAL SUM OF SQUARES ABOUT MEAN (TSS), DEGREES OF FREEDOM AND MEAN SQUARE ARE
1.0231  47  0.02176808511
PROCEED BY REPEATEDLY CALLING THE FUNCTION 'EFFECT'.
THE ARRAY OF RESIDUALS IS ALWAYS NAMED RY, WITH DEGREES OF FREEDOM NU.
RENAME SS EACH TIME TO SAVE IT.
       E1←EFFECT 1
SUM OF SQUARES (SS), D.F. AND MEAN SQUARE ARE 0.443025  5  0.088605
       SS1←SS
       E12←EFFECT 1 2
SUM OF SQUARES (SS), D.F. AND MEAN SQUARE ARE 0.247475  6  0.04124583333
       SS12←SS
       E123←EFFECT 1 2 3
SUM OF SQUARES (SS), D.F. AND MEAN SQUARE ARE 0.1599  12  0.013325
       RSS, NU, (÷NU)×RSS←+/,RY*2
0.1727  24  0.007195833333
```

FIGURE 8. Analysis of variance of a multidimensional array.

effects can meaningfully be asked for, and which interactions will serve as error estimates for which comparisons. Provided all this is understood, successive applications of EFFECT, with arguments arranged in order of nondecreasing length (main effects first, then two-factor interactions, then three-factor interactions, etc.), will yield a correct analysis of variance. EFFECT goes to work, not on the original data array, but on the residual array RY left by ANALYZE and by any previous applications of EFFECT. If (for example) EFFECT 1 3 is called, this will yield an interaction of the factors (dimensions) numbered 1 and 3, but just what this interaction means and how many degrees of freedom it has will depend on whether previously either or both of EFFECT 1 and EFFECT 3 have been called -- but not (say) on whether EFFECT 2 has been called. To arrange the computation so that the complete residual array is calculated at each stage would be suicidal for the operator of a desk calculator, but is good practice with a big computer in order to suppress round-off error.

EFFECT has an explicit result, corresponding to the vectors RE and CE in ROWCOL. The user supplies a name (in place of the dummy Z shown) for this effect if he wants to keep it, a different name for each effect. If for example V is the vector (1, 3), he might choose to label the effect matrix E13 . The names M , K , P , IND appearing in the program are designated as dummy variables by being listed after semicolons in the header.

The first three statements of EFFECT check that V is made up of one or more members of the set (1, 2, 3, ..., ⍴⍴RY), with no repetitions. The comma placed in front of V in statement 1 turns V into a vector even if it has only one member (i.e. even if V is scalar); hence M is a matrix even if it has only one row. In statement 2 a "logical" indicator vector IND is obtained, consisting of 1's and 0's, of length ⍴⍴RY (the dimensionality of the data array); 0's are in the places numbered by the members of V , 1's are elsewhere. Thus if the data array has five dimensions and V is (1, 3), IND is (0, 1, 0, 1, 1). Statements 4 and 5 are concerned entirely with finding the number of degrees of freedom associated with effect V , and the number of residual degrees of freedom. DF is a matrix, of which the first row was found in ANALYZE. Each time EFFECT is called, a new row is added onto DF , consisting of the vector IND followed by (at first) a 0, which however is subsequently changed (in statement 5) into the number of degrees of freedom in the effect. In the illustration at the foot of

Figure 8 the data array is four-dimensional. After the third calling of EFFECT, DF looks like this:

```
1   1   1   1   1
0   1   1   1   5
0   0   1   1   6
0   0   0   1  12
```

One detail in statement 5 calls for comment. In the middle we see the expression DF[; +/ι+ρρRY] . This represents the vector whose elements are the last column of DF (before the final 0 has been changed). Since monadic ρ always yields a vector, ι+ρρRY is a vector of unit length, not a scalar. Putting "+/" in front reduces this vector to a scalar without otherwise changing it. Without the "+/" we should obtain the last column of DF as a one-column matrix instead of a vector. Distinctions of this kind are important in APL ! The reader may now enjoy trying to decipher the somewhat tortuous calculation expressed by statement 5.

Statements 6 and 7 of EFFECT are the ones that directly tackle the multidimensional array RY . What we need to do is sum RY over all coordinates except those listed in V , divide by the appropriate divisor (labeled K) to form the effect array Z , and then from K repetitions of Z build up an array of fitted values, which is subtracted from RY to form the new array of residuals, named RY again. We also need to calculate the sum of squares for the effect, named SS .

The way this is done is first to permute the coordinates of RY so that the coordinates numbered V come last. Then the array is restructured so that all the other coordinates are elided into one coordinate, and a single summation is performed over that coordinate to yield Z . This, together with the finding of SS , is specified in statement 6. In statement 7, the array Z is repeatedly stacked on itself, and then the inverse of the previous permutation is performed on the coordinates to yield the array of fitted values, to be subtracted from RY . The transposition symbol for permuting the coordinates of an array is a sort of backward-sloping phi. On its right is the array to be transposed (say A), on its left a logical vector (say T). The Ith coordinate of A appears as the T[I]th coordinate of the result. Two other symbols in statements 6 and 7 are used in ways we have not met so far. The compression symbol / may be preceded, not only by a standard dyadic scalar function like + or × , but instead by a logical vector, here IND . The following vector (which is of the same length as IND) is compressed by omission of members corresponding in position to the 0's in IND . Near the end of statement 6 two consecutive iotas will be seen. The first of these is a dyadic iota, set between two permutation vectors of equal length. The result is a vector listing the positions in the left-hand vector where the members of the right-hand vector occur.

As a brief illustration in the small amount of space available in the lower part of Figure 8, these functions are applied to some data stored under the name EGGDATA, taken from Bliss's book and not reproduced in Figure 8. The four-dimensional array is a simple nested arrangement with no crossed factors other than the highest-level classification (laboratories). EGGDATA is called, then ANALYZE, then EFFECT three times, and finally the residual sum of squares is found. The standard analysis of variance table looks like this:

	Sum of squares	D.f.	Mean square
Between laboratories	0.4430	5	0.0886
Between analysts within laboratories	0.2475	6	0.0412
Between samples within analysts	0.1599	12	0.0133
Residual	0.1727	24	0.0072
Total about mean	1.0231	47	0.0218

The effects E1, E12, E123 are stored and can be displayed. The sums of squares SS1, SS12, SS (we did not rename this last) and RSS are all stored, and we can check that their sum is equal to TSS , as it should be. The final residual array RY is stored and available for further study.

4. CONCLUDING REMARKS

Some comments on the foregoing description of APL are in order. Many things have not been said. The following matters have not been explained:

(i) how to sign on and off,

(ii) how to move material in and out of storage,

(iii) how to correct an error in typing,

(iv) how to add to, emend or "edit" a function definition (program),

(v) how to interpret error reports,

(vi) how to obtain various items of information, such as: (a) the amount of unused space remaining in the active workspace, (b) the number of users of the system currently signed on, (c) the amount of central processor time used since sign-on.

Such matters as these, for which excellent provision has been made, are of immediate concern to anyone working at an APL terminal, but they are irrelevant to the purpose of this paper, which is to convey the flavor, and suggest the suitability, of APL in expressing statistical calculations. Details of implementation of the language in conversational mode are in principle open to modification, more so than the language itself--which may indeed be extended by addition of new standard functions but could hardly be reorganized or radically changed while still referred to by the same name. Implementation details are explained in the manual.

The manual should be referred to also for precise definitions of APL symbols. Explanations of symbols given above have been informal, intended to help the reader make sense of the illustrations quickly, rather than to answer the questions that will arise as soon as he tries to write in APL himself.

Many things have been omitted that were fully relevant to the purpose of this paper. An attempt to show everything would have overwhelmed author and reader both. The reader should particularly note this: just because no mention has been made of some topic that interests him, he ought <u>not</u> to conclude without further inquiry that APL is unfit to handle that topic. Deliberately, less than the full vocabulary of APL has been described, and character manipulation and format control have been barely mentioned. Interactive programs in which the machine questions the user and acts on his answers have not been illustrated. Many very common operations in statistical work, in particular regression, have not been shown; nor has the organization of functions (programs) together to form blocks in a complex calculation, functions being called by functions.

Elegant programs for many common statistical calculations have been composed by Smillie [13]. The scope of APL in that kind of use can be seen very well from this collection. Many good programming devices are illustrated.

In one respect, the modest illustrations in this report show correctly a feature of APL terminal use that can be lost sight of. By all means let the user be uninhibited in defining functions (storing programs), and then changing and dropping them. But he should not forget that the terminal can also be used like the old desk calculator, and he can always revert to the impromptu dialog style. Often a very few APL symbols will express a useful block of computation. In a less concise language such a block would be worth storing as a subroutine. To store it as an APL function will only be profitable if the user can remember the function's name and syntax the next time he wants it--remember these sooner than he can reinvent the program. He should think of stored programs always as aids to free dialog, not as controls to his thinking (as program packages in other languages have so often been). The whole of a complex calculation need not necessarily be enshrined in a stored program.

That brings us to the last question we shall consider: what is the slant of APL?

We are conscious of a great change when we lay aside paper-and-pencil calculations, assisted possibly by slide rule or desk calculator, and go courting a computer. We have to think more explicitly about our procedures, in matters that formerly were left to judgment or common sense. We must learn new terminology, and--much more important--we find ourselves thinking about the calculations in a somewhat different way. Any method of communicating with the computer (language, program package, interactive system) has a slant; it will not just transmit our thinking, it will modify our thinking. Some things are easier to do than others, our attention is directed in particular ways, unexpected questions must be answered. What slant does APL have for the statistician?

None whatever of a direct statistical kind! There is nothing in APL that can plausibly affect a statistician's philosophy, ethics, basic concepts or theoretical ideas--except indirectly (the more effortless computation becomes, the more the statistician will compute, and some of his basic ideas will be changed by the experience). By contrast, any computing system or program package designed specially to aid the statistician will also to some extent bias his statistical thinking. Schatzoff's COSMOS begins with the six Beaton operators. There is vastly more to COSMOS than that, the Beaton operators can be avoided, but there is a definite "atmosphere" that will slant the user toward the Beaton operators. It will come natural to him to think of some problems in one way rather than another, to arrange his computations so. Of course, this is far more so if the statistician relies on a program package such as BMD . Then if he wishes to do something different from what the authors intended he needs to be ingenious in "bucking the system" or "cheating". Of course, again, the statistician will be slanted or constrained in his thinking if he relies on a library of statistical programs written in APL when he himself is too little familiar with APL to indulge effectively in free dialog. Such slanting is not necessarily bad. For some users it is a positive help, as has already been remarked in section 1.

Though APL has no substantive statistical slant, it certainly has a distinctive character among programming languages. The conciseness, the generality and the affinity with mathematical thinking lead to transparency--or (to change the metaphor) the machinery of APL creaks less than that of other languages.(5) Apart from the sheer ease of expression, APL has at least one noticeable algorithmic slant: it encourages us to replace loops whenever possible by array operations, and therefore to think in terms of arrays in various numbers of dimensions.

Compare the mathematical concept of a finite sum, say $\sum_{i=1}^{n} a_i$, with an APL sum reduction, say +/A . In the mathematical expression, no particular order of summation is implied, because addition is associative. In computation, addition is associative in integer arithmetic (provided there is no overflow) but not in floating-point arithmetic, and there are occasions when lack of associativity matters. The sum reduction in APL is carried out in a known order (right to left). If we desire a different order of summation, we may permute the array before reducing it. Thus we can control the order of summation as we wish, and yet use a notation (+/A) that views the array as a whole and does not bring the order of summation into the limelight by making us write a loop. Very often in practice that order of summation is unimportant, and we ought not to have it obtruded on us.

Thus we become sharply aware of the difference between repetitive operations that can be expressed as operations on arrays defined at the outset, on the one hand, and successive approximations and recursive definitions wherein some kind of looping is essential, on the other hand. Our understanding of algorithms is increased. For this sort of reason it may be claimed that the development of APL has been a contribution not only to computation but to computer science.

Further reading. For further information about APL, whether or not a terminal is available to experiment with, see the Falkoff-Iverson manual [4]. After some experience with an APL terminal, one may read Pakin's reference manual [8] with interest. Many general topics concerning computing and computers are discussed lucidly by Brooks and Iverson [2]. Iverson's orginal book on his language [6] was a companion piece to the Brooks-Iverson book, but is far more difficult to read. Moreover, there are differences in principle as well as in detail between the language there presented and what is now implemented as a computer language. An examination of both books together shows the range of ideas underlying APL. In a later book [7] Iverson illustrates his language by a high-school-level account of transcendental functions. Smillie has several forthcoming articles on the use of APL in statistics.

Acknowledgements. I am indebted to my former colleague, Dr. Trenchard More, Jr., for introducing me to APL and for many stimulating conversations; and to International Business Machines Corporation for permission to use their experimental APL system. Several colleagues and friends have made helpful comments on a draft of this paper, notably Professor I. Richard Savage.

APPENDIX ON TERMS

Computers perform a great variety of logical tasks - numerical calculation and many kinds of manipulation of symbols. A procedure for carrying out such a task is sometimes called an algorithm (wrongly, according to Webster and Oxford). A precise specification of such a procedure, especially one that is coded so that it may be followed by a computer, is called a program. A program consists of a sequence of statements or commands; some statements specify the value of one or more quantities in terms of operations on other quantities; other statements, known as branches, specify which statement shall next be executed. A set of statements that are executed repeatedly, because a branch at the end returns execution to the beginning of the set, is called a loop.

For example, a well-known procedure for finding the square root y of a given positive number x may be expressed informally thus: guess a value for y; find the average of y and the quotient $(x \div y)$; this is a better guess; repeat until the better guess is equal to the previous guess to however many decimal places you want. This might be set out in numbered statements more formally thus: -

```
1. Set $y = 1$ .
2. Set $y' = y$ .
3. Set $y = \frac{1}{2}(y' + \frac{x}{y'})$ .
4. If $|y - y'| \geq 10^{-4}$ , go to statement 2; otherwise, stop.
```

Here statements 1-3 are specification statements, statement 4 is a branch, statements 2-4 constitute a loop.

Numbers are represented in the computer in the binary scale. A binary digit is called a bit, and 8 bits are called a byte. Three sorts of numbers are recognized in the APL system, namely (i) logical numbers equal to either 0 or 1, which can be stored as single bits, (ii) signed integers less than 2^{31} (roughly 2.1×10^9) in magnitude, which can be stored in 32 bits or 4 bytes, (iii) signed floating-point numbers whose magnitude lies between (roughly) 7.2×10^{75} and the reciprocal of that and whose precision is equivalent to about 16 decimal digits -- such numbers can be stored in 64 bits or 8 bytes. The number 7.2×10^{75} is an example of a number expressed in floating-point decimal notation. An alternative notation for it, used sometimes by the computer in output, is 7.2E75 or 0.72E76 , etc., where the number following the E is a positive or negative integer denoting the power of 10 multiplying what precedes the E . (See line 21 in Figure 2.)

The precise way the output of the computer (or input to it) is displayed, the number of spaces, the number of digits after the decimal point, whether in floating-point or fixed-point notation, how many numbers to a line, etc. -- all such details of display are called format.

We speak of the operator or user of an APL terminal. In some contexts the word "operator" would be taken to mean the "system operator", but we do not refer to that august personage.

The physical material of a computer is called hardware. Programming, especially the basic system programming that permits the hardware to be used, is called software.

Until recently, most computer operation has been in batch mode. Programs are executed one after another, more or less in order of submission. Execution has two steps, (i) a translation of the whole program into machine language, known as compilation, (ii) running the compiled program. Results of each job are put out through a printer or punch or transferred to tape, usually so that they cannot be considered to be retained within the computer, though if they are punched or taped they may possibly be resubmitted for further processing by the computer on another occasion. Results are returned minutes or hours after submission of the program.

Recently there has been much interest in interactive use of computers, where the computer responds rapidly and automatically retains material, so that successive commands can be given. In the conversational mode almost instantaneous response to relatively simple commands may be obtained through time-sharing.

FOOTNOTES

(1) No attempt has been made here either to list all special systems relevant to statistical work or to evaluate any of them. The writer himself has used none of them. As for the tediousness of programming in FORTRAN etc., there are other troubles besides the negotiation of arrays. Not least of these is the abundance of arbitrary conventions and restrictions that are hard to remember accurately.

(2) In the best of all possible worlds special systems would themselves be programmed in the best general-purpose algorithmic language and would permit the user to leave the special system and carry on alone whenever he wanted to.

(3) "Most of the concepts and operations needed in a programming language have already been defined and developed in one or another branch of mathematics. Therefore, much use can and will be made of existing notations. However, since most notations are specialized to a narrow field of discourse, a consistent unification must be provided. For example, separate and conflicting notations have been developed for the treatment of sets, logical variables, vectors, matrices, and trees, all of which may, in the broad universe of discourse of data processing, occur in a single algorithm." ([6], pp. 1-2)

(4) This description of the experimental APL system at Yorktown Heights is based on my experience during a year of permitted use of the system. I believe that the facts asserted are substantially correct as far as they go. My concern is to suggest that something like this APL system can make communication between statistician and computer so easy that the sheer effort of coding is negligible. No need to employ expert programmers and unravel the misunderstandings! Even I now write my own programs!

(5) I should say that the machinery creaks a bit in statements 6 and 7 of EFFECT in Figure 8.

REFERENCES

[1] BLISS, C. I. Statistics in Biology, vol. 1. McGraw-Hill, New York, 1967.

[2] BROOKS, F. P. and IVERSON, K. E. Automatic Data Processing. Wiley, New York, 1963.

[3] DIXON, W. J. (ed.). BMD: Biomedical Computer Programs. UCLA Student Store, Los Angeles, 1964.

[4] FALKOFF, A. D. and IVERSON, K. E. The APL Terminal System: Instructions for Operation. IBM, Yorktown Heights, N. Y., 1966, revised 1968.

[5] HILSENRATH, J., ZIEGLER, G. G., MESSINA, C. G., WALSH, P. J. and HERBOLD, R. J. OMNITAB: A Computer Program for Statistical and Numerical Analysis. National Bureau of Standards Handbook 101. U.S.G.P.O. 1966, revised 1968.

[6] IVERSON, K. E. A Programming Language. Wiley, New York, 1962.

[7] IVERSON, K. E. Elementary Functions: An Algorithmic Treatment. Science Research Associates, Chicago, 1966.

[8] PAKIN, S. APL\360 Reference Manual. Science Research Associates, Chicago, 1970 (issued for private distribution 1968).

[9] POMPER, I. H. et al. STORM: Statistical Oriented Matrix Program. IBM, Yorktown Heights, N.Y., 1963. (Reference copied from J. M. Chambers [14], p. 132.)

[10] Proceedings of the IBM Scientific Computing Symposium on Statistics, October 21-23, 1963. IBM, White Plains, N. Y. 1965.

[11] SCHATZOFF, M. Console oriented model building. Proceedings of the 20th National Conference, Association for Computing Machinery, 1965.

[12] SCHATZOFF, M. Applications of time-shared computers in a statistics curriculum. Journal of the American Statistical Association, 63(1968), 192-208.

[13] SMILLIE, K. W. STATPACK1: An APL Statistical Package. Department of Computing Science, University of Alberta, Publication No. 9, 1968.

[14] Statistical programming; papers and discussion at a meeting organized by J. A. Nelder and B. E. Cooper, December 15, 1966. Applied Statistics, 16 (1967), 87-151.

ERRATA

Page 8, line 1. For "the integer part of" read "the greatest integer not greater than".

Page 8, line 5. For 14 read 16 .

Page 8, line 7 from foot. For "functions defined by programs" read "functions defined by the user".

Page 9, line 9 from foot, and later. For "concatenation" the approved term is "catenation".

Page 11, line 10 from foot. For "sum compression" the approved term is "sum reduction". Similarly, lines 4-3 from foot, for "coordinate to be compressed" read "coordinate to be reduced". And page 14, line 11, for "compressing Y " read "reducing Y ". (But the operation described later, page 25, last line, to page 26, line 4, is known as compression and the text is correct.)

Page 17, line 14 from foot and later. For "open product" read "outer product".

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Yale University	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

USE OF IVERSON'S LANGUAGE APL FOR STATISTICAL COMPUTING

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(Last name, first name, initial)*

Anscombe, Francis J.

6. REPORT DATE July, 1968	7a. TOTAL NO. OF PAGES 43	7b. NO. OF REFS 14

8a. CONTRACT OR GRANT NO. Nonr 609(52)	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report No. 4
b. PROJECT NO. RR 003 05 01	
c. Task No. NR 042-242	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Logistics and Mathematical Statistics Branch, Office of Naval Research, Washington, D.C. 20360

13. ABSTRACT Problems of communication between statistician and computer are reviewed. Needs are diverse. For the professional mathematically-oriented statistician Iverson's language APL is very suitable, more so than special statistical computing systems. A partial description of APL is given, and illustrated by some typical statistical calculations. An experimental implementation of the language in conversational mode through typewriter terminals is described. The system may be used as a desk calculator, and programs and data may be stored. The purpose of the paper is not to duplicate the manual, but to convey the flavor and suggest the suitability of APL for expressing statistical calculations, without presenting too much unfamiliar notation. Two attractive features of the language are its capacity to represent array operations concisely and its high degree of consistency and generality

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
APL						
Iverson, Kenneth E.						
Statistical computing						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through ______________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through ______________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through ______________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.