STATISTICAL COMPUTING 1970

A MEETING on "Statistical Computing—The Present Situation and Future Prospects" was organized by Dr J. A. Nelder and Mr B. E. Cooper in December, 1966, at the Atlas Computer Laboratory, Chilton. The papers presented at that meeting, and some of the discussion, were published in *Applied Statistics* (**16**, No. 2, 1967).

The meeting on statistical computing held at Imperial College on July 16th and 17th, 1970, reported on the activities of the Working Party on Statistical Computing which was set up at the Chilton meeting, as well as presenting specially invited papers on topics of particular interest. It seemed appropriate, therefore, that *Applied Statistics* should publish the report of the Working Party and the invited papers, and they are included as a special section of this issue of the *Journal*. No excuse is needed for the inclusion of so important a subject as statistical computing in a journal devoted to the application of statistics, and there is particular interest in having these papers in the same issue because of the close interrelationships of the topics with which they deal.

J. N. R. JEFFERS

Statistical Computing and Computer Languages

By J. A. NELDER

Rothamsted Experimental Station

SUMMARY

The activities of the Working Party on Statistical Computing are discussed under two headings. (1) The Algorithm Section of *Applied Statistics;* problems of algorithm construction and testing are described, and illustrated by the formation of a sums-of-squares-and-products matrix from a data matrix. (2) Data structures; the possibility for the statistical programmer of defining and operating upon data structures relevant to statistics is not met by commonly available languages. The impending arrival of extensible languages should avoid the need for statisticians to implement special statistical languages, but increases the need for them to standardize their data structures if programs are to be easily used and transferred.

HISTORY

IN December 1966, Brian Cooper and I organized a meeting at the Atlas Computer Laboratory, Chilton, on "Statistical Programming—the Present Situation and Future Prospects". The papers, and some of the discussion, were published in *Applied Statistics* (**16**, No. 2), and at the end of the meeting a Working Party on Statistical Computing was set up. Since the Chilton meeting, the Working Party has met nineteen times, and now consists of eight members, with interests covering statistics as applied to agriculture, medicine, meteorology, industry and government. The Working Party has acted as a self-educating body; members have had their vision

enlarged by being brought face to face with the problems of others working in different environments with different kinds of data. Hidden assumptions are brought to light, and examined in a wider context.

Contact has been made with similar groups in Holland and the U.S.A. and a paper by Dr Van Reeken from the Dutch group appears in this issue (pp. 73–79).

The Working Party has followed two main lines of work, the establishment of the Algorithm Section in *Applied Statistics* and a more general study to try to uncover the basic structures of statistical analysis and to see how their definition and handling can be achieved in general-purpose languages.

Algorithms

The first of the Working Party's activities (showing immediate results) has been the establishment and maintenance of the algorithm section in *Applied Statistics*. This began with a paper by the Working Party on the construction and description of algorithms in **17**, No. 2, 1968, in which we tried to set down useful guide lines for contributors. The same issue also contained the first seven algorithms, drawing heavily, it need hardly be said, on members of the Working Party itself. In the first two years the *Journal* has published 31 algorithms, in Fortran and in Algol. The first algorithm in PL/1 has yet to appear, and so far only one has been submitted. The subjects of the algorithms have been varied, but general categories that we anticipated have appeared, e.g. probability functions, operations on data (e.g. formation of *SSP* matrices) and data handling in the sense of organizing the accessing and storage of data. In parallel with the algorithms, we have encouraged the publication of papers in the *Journal* relating to general aspects of computing in relation to statistics, and several such papers have appeared, including one by Chambers (1969) on model fitting, and one by Anderson and Lowe (1970) on regression.

We have not tried to establish a clearing house for algorithms or programs, or to issue any sort of seal of approval of published or unpublished programs, except for the refereeing of the algorithms we publish. Though these activities would be well worthwhile they would take more resources than we command. The process of refereeing is itself quite arduous, and Working Party members do most of it. Such refereeing is a specialized business requiring a considerable range of skills. The referee has to assess not only the statistical usefulness of the algorithm, but also the soundness of its numerical methods, the clarity of the structure, the efficiency of the coding (whether it will run fast), the conciseness of the coding (whether it will occupy minimal space), whether it has any dialect features and how machine-independent the specification is. In addition, the referee has the immediate practical task of testing whether the algorithm actually works, and the difficult theoretical task of assessing whether its general specification will allow it to be embedded easily in a range of programs.

This last is, perhaps, the most useful task the referee can perform; it immediately raises a general question concerning the relation between generality and efficiency. A hand-coded program to do a special operation will almost always be more efficient than a more general program used for the same purpose. This is basically because the more general program, having more options, must make more tests during execution to see which of its branches it must take for the specific problem in hand. Thus if a program is doing item-by-item operations on vectors, then it will certainly be slowed down if missing values are allowed in any of the operands with the result set missing if any of the operands is. The basic problem, therefore, is where the line between generality and efficiency should be drawn. Against the loss of speed in more general

programs must always be set the costs in people's time and machine-time of writing and testing special one-off programs; I suspect that these costs are often much greater than is commonly realized. It would be most interesting to have figures to show where the break-even point lies. Some types of generality can be cheaply bought, and these should be included in a good algorithm. For example, a line-printer plotting routine can fairly easily be written to include the page width as a parameter, instead of fixing it arbitrarily to that of a particular installation. Again, pathways through logic including loops can often be set up conveniently *outside* the loop (e.g. by the *ASSIGN* statement of Fortran), and so avoid the continual testing of conditions inside. Thus the problem of missing values in item-by-item operations on vectors can be eased if vectors carry a count with them showing the number of missing values they contain. A switch can be set outside the loop, jumping over the test for missing values when there are none. With no missing values such a routine will run almost as fast as one not catering for missing values at all, while retaining the generality when they do occur.

Another important point of algorithm design for general use concerns input/output statements. They are almost always better segregated completely from the rest of the algorithm, and their invocation needs to be able to be controlled by the user.

The Formation of SSP Matrices

The formation of a sums-of-squares-and-products matrix from a data matrix illustrates some of the problems to be met in designing a good algorithm. The method for the desk calculator, involving subtracting the correction for the mean from the uncorrected sum of squares (or products) is unsatisfactory, because of the loss of significant figures generated in floating-point arithmetic when one large number is subtracted from another nearly equal large number. An alternative of subtracting the mean from each observation before squaring is accurate, but unsatisfactory when the data matrix is so large that it cannot be held in core with the *SSP* matrix. The algorithm that updates the *SSP* for each row of the data matrix and simultaneously updates the means avoids this latter difficulty, though it may not be quite so accurate as the second method. This algorithm was given in *Applied Statistics* as Algorithm AS 12 by Herraman (1968). It used an accessing subroutine to get the next row of the data matrix to avoid external assumptions about the mode of storage of the data matrix, e.g. data stored by variates will not be compact by units; this algorithm went through a long process of modification and amendment, but even so is not the last word, because Beale (personal communication) has rightly pointed out that the coding in the inner loop can be tightened.

The development of this algorithm exemplifies three important points: (1) The importance of considering numerical accuracy (particularly on hexidecimal 32-bit-word machines); (2) the importance of establishing what the basic input data structure is; (3) the advantage of writing the algorithm to operate *sequentially*, i.e. to update an *SSP* matrix given a new set of rows. The sequential algorithm can be easily embedded in a program where the data matrix must be segmented because of its size, while still working efficiently in simpler situations.

I believe that we have still much to learn about the design of good statistical algorithms. The recent interest shown by numerical analysts in the processes of least-squares (see especially Golub, 1969) will have a considerable influence on statistical computations. In particular the work by Wilkinson (1965) and others on the extraction of latent roots and vectors has still to make its full impact, judging by the number of bad programs in present use.

I now turn to the second of the Working Party's activities, namely the investigation of data structures in relation to statistical computing and the extent to which existing and proposed languages meet the statistician's needs.

Data Structures

The kind of algorithms that are usually written in Fortran and Algol define operations on a list of operands, usually presented as formal parameters to the subroutine (procedure). Each formal parameter may be one of a limited number of kinds of object in the language, e.g. integer scalar, real array of one or more dimensions, a function and so on. The implication for the statistical programmer is that he must therefore break down the input for any algorithm into components able to be passed as parameters in the language in which he is writing. This represents a very considerable restriction on his ability to describe easily what he wants to do, especially when the operands he is dealing with are complex and require a long list of components to specify them. General programming immediately exposes the need for languages allowing the definition of new *operands* as well as the new operations defined by algorithms in conventional languages. Extending the class of operands is, of course, not a new idea; Cobol, for instance, introduced a class of data structures (records) with items grouped in a tree, and this class was incorporated into PL/1 which also defines operations on them. Such a class of structures is useful to the extent that it meets the programmer's needs; it clearly meets a need in commercial work, where record descriptions can often be formalized for specific jobs for considerable periods of time. The program, although specialized, has a reasonably long active life. In a research environment, where data collected often have a rapidly changing structure, corresponding to the development of ideas and theories, the Cobol structures would often prove too rigid to be useful.

Problem-oriented Languages

Cobol is an example of a problem-oriented language (POL), designed to allow a class of special problems (relating to commercial data-processing) to be specified naturally and executed reasonably efficiently. Other POLs have arisen for symbol manipulation (e.g. Snobol), simulation (e.g. CSMP), survey analysis (e.g. MVC and GSP) and many other applications. Members of the Working Party have been responsible for at least four such attempts to define and develop a POL for statistics. In parallel with all this specialist activity, language designers have been continually extending the possibilities of computer languages, regarded as a quite general exercise in communication between man and machine. We have now reached a point when a fundamental question must be posed: do we need a special statistical language, or will the next generation of general-purpose languages give us all we need? Before trying to answer this it is worth looking in some detail at the sort of facilities that general-purpose languages may be able to offer us in the future.

The new languages will be characterized by the property of *extensibility*. This means roughly that the programmer will be able to define new entities in the language, and also define how their occurrence in subsequent statements is to be understood and dealt with.

It is convenient to distinguish two kinds of extensibility, *semantic* and *syntactic*. Semantic extensibility is closely allied to the definition of new operands and structures:

it is a 'this-is-what-I-mean-by' kind of process. As an example we may take the group definition of Hendry's BCL which uses the system words *IS*, *EITHER* and *OR*

Name *IS* (title, *EITHER* surname *OR* initials, surname)
Initials *IS* (letter, ".", *EITHER* initials *OR* nil).

These groups define a class of objects consisting of a title and surname prefaced by an arbitrary number of initials (including none).

The identifier elements, such as *surname*, define internal storage, and, if prefaced by a format, external representation can be declared simultaneously. Once a structure has been defined it may be used in various contexts in the language (though in the BCL example only input and output are relevant). Its definition is an encapsulation of components, which may subsequently be invoked by the use of its name. This idea may be generalized to the definition of a *class* of structures, which are first defined in abstract, their components being set out, but without the implied allocation of storage; later *instances* of the structures may be named (storage being then set up) and manipulated.

At this point, we have the ability to define new types of object in the language, an extremely powerful facility and capable of extension to objects of indefinite complexity.

The other kind of extensibility (syntactic) represents a "this-is-how-I-shall-write-it" facility. It appears in a very restricted form in the Fortran "Equivalence" statement; thus

$$EQUIVALENCE\ (A(1), X), (A(2), Y), (A(3), Z)$$

represents a way of saying "I shall write X, Y, Z to mean the 3 elements 1, 2, 3 respectively of the array A". An important potential use of syntactic extensibility is to define the format for invoking functions; thus if *matmult* defines a multiplication of two matrices to produce a third, then we usually have to write something like

$$\text{matmult}\ (A, B, C)$$

to produce the operation that is mathematically described by

$$\mathbf{A} = \mathbf{BC}.$$

Syntactic extensibility would allow us to declare in our program that

$$\text{matmult}\ (A, B, C)$$

would be written as

$$A = B\ \text{op}\ C$$

where *op* stands for some suitable symbol(s) indicating matrix multiplication.

Consequences of Extensibility

If extensibility becomes available in computing languages it will have profound effects not only on users' programs but also on compilers and operating systems. An extensible language will contain a very few basic written forms (syntax) and associated concepts (semantics); Godfrey has suggested to me that a successful language of this kind will be characterized by having a ten-page manual. On this small but powerful base, different groups of users will be able to construct their own edifices, each with its appropriate structures (operands) and operations defined on those structures. Thus, for example, there might well be a 200-page manual on survey programs, describing the characteristic structures of surveys, such as the hierarchical data matrix shown in

Fig. 1, and operations such as those on multiway tables for which an example is shown in Fig. 2. An important consequence of extensibility is that compilers will no longer be the frozen entities they are at present. They will proliferate as new types of object in the language or new syntax are defined and used. The barriers between compilers and the programs they operate on will break down.

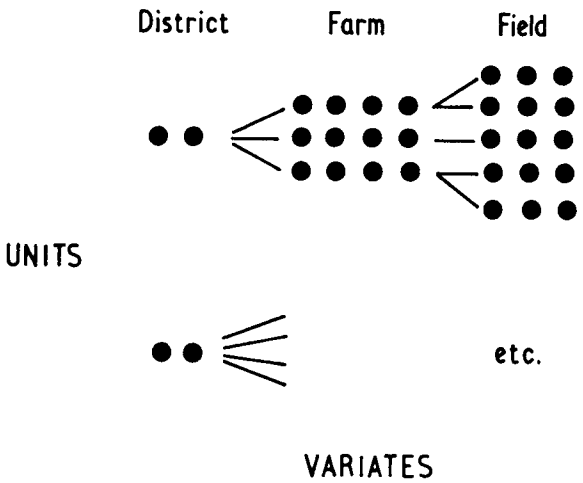

FIG. 1. Hierarchical data matrix from a survey.

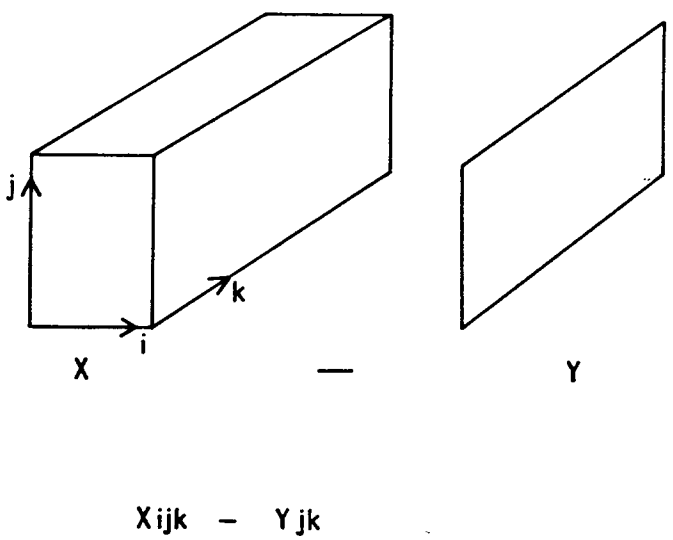

FIG. 2. A table operation showing the subtraction of a two-dimensional table from a three-dimensional one.

Thus the survey programs mentioned above would require a special form of the compiler to accompany them before they could be used. This in its turn will imply that the compilers themselves will have to be much more mobile in the future than they have been in the past. At present compilers, in their operating system environments, are

often a very special kind of program; they can be invoked only in special circumstances and in rather rigid ways. This rigidity will have to disappear if extensible languages are to become a reality. Irons (1970) gives an interesting account of an extensible language and its associated compilers.

Statistical Computing and Future Languages

We now return to the basic question—do we need a special statistical POL, or merely the right kind of general extensible language with which to work? The discussion above points to the answer: we do not need one *or* the other—we need both. First we need the basic extensible language to write our programs in and secondly we need the special operands and operators of statistics, defined in that language and using a convenient syntax for their declaration and execution. We may think of the basic language as a machine-independent assembler-type language, the sort of language in which compilers and operating systems can be written as well as users' programs with common facilities usable by all programmers. For further discussion in relation to operating systems, see Godfrey (1971) in this issue of the *Journal.*

Clearly this basic language is not one for statisticians either to design or to implement, except in so far that they should make sure that the facilities it provides are adequate for extension. For example, we should be sure that arrays with a variable number of dimensions can be defined and efficiently accessed. At the second stage the needs of statisticians are paramount, and here they should look intensively at their requirements, and in particular investigate when standards can usefully be laid down without imposing undue rigidity. Much of this work can be done outside the context of a particular language by examining the operands and operations of statistical analysis, and defining structures and algorithms accordingly. Much has been done already by those who have constructed statistical systems, but this work has yet to make the impact it should. (One reason for this is undoubtedly the lack of structure-defining facilities in the programming languages used.) Without agreement between statisticians the effective exchange of programs will remain as difficult as it is today. Dialect differences in compilers cause enough trouble in transferring programs, but they are far from being the only cause of this trouble. Incompatible formats, both internal and external, cause delays, first by difficulties in matching the new program with existing programs in the user's understanding, then secondly by the extra work required to write interfaces between them. The problems of writing interfaces are often so great that the user reluctantly discards the program he has obtained and sets about duplicating, once again, its facilities in his own fashion.

The papers given at this conference reflect the twin concerns of the Working Party with algorithms and with languages generally. Dr Van Reeken (1971) from Holland gives a paper on algorithms from the viewpoint of the Dutch Working Party, who have been very active in this field. Important as the subject of algorithms undoubtedly is, we make no apology for devoting the remaining three papers to the general question of languages and operating systems in relation to statistical computing. Our algorithms have to be written in a language and they have to run in an operating environment. How easily they can be expressed in a language, and how efficiently they can be organized in a program and run, are vital to us all. At first sight some of the subject matter of these papers may appear far removed from statistics, and in the sense of being removed from statistical theory perhaps it is. But where the analysis of data is concerned we believe that the subject matter is very relevant and deserves a full airing and a full discussion.

Many of us are immersed daily in the often time-consuming process of writing and maintaining statistical programs. It is easy to become so immersed that one never has time to look ahead and think strategically rather than tactically. In this conference we have tried to look at strategy.

Conclusion

I hope these papers will provide an assessment by a special group of computer users of their needs, how far these are met by present-day software (programming languages and operating systems), and what future improvements are required. We believe that there is an urgent need for better communication between language designers and their potential customers. (The recent IFIP Working Group 2.3 set up under the chairmanship of Dr Woodger is another response to the same need.) The designers must know what facilities will be useful to their customers, who in turn must know what they can reasonably ask for. In setting down our ideas in the papers for this conference, we hope simultaneously to persuade language designers to be more helpful to statisticians, and to let statisticians know what is happening and could happen in statistical programming.

References

Anderson, A. J. B. and Lowe, Bridget I. (1970). A multivariate analysis computer program. *Appl. Statist.*, **19**, 18–26.

Chambers, J. M. (1969). A computer system for fitting models to data. *Appl. Statist.*, **18**, 249–263.

Godfrey, M. D. (1971). Operating system considerations for statistical computing. *Appl. Statist.*, **20**, 45–56.

Golub, G. H. (1969). Matrix decompositions and statistical calculations. In *Statistical Computation*, pp. 365–397. New York: Academic Press.

Herraman, Carol (1968). Sums of squares and products matrix. *Appl. Statist.*, **17**, 289–292.

Irons, E. T. (1970. Experience with an extensible language. *Comm. Ass. Comp. Mach.*, **13**, 31–40.

Van Reeken, A. J. (1971). Report on the work of the Dutch Working Party on Statistical Computing. *Appl. Statist.*, **20**, 73–79.

Wilkinson, J. H. (1965). *The Algebraic Eigenvalue Problem.* Oxford: Clarendon Press.

Internal Data Structures

By J. C. Gower and I. D. Hill

Rothamsted Experimental Station *M.R.C. Computer Unit*

Summary

In most computer languages, the data are regarded as being held in machine storage either as individual scalar quantities or as members of rectangular arrays. It would make these languages more useful if the user were allowed to specify the shapes of other structures in which he wished to consider the data as being held, such as triangular arrays, trees, etc.

To go with the ability to define structures the user would require the ability to define operators to act upon structures.

Introduction

In these pages, Nelder and Cooper (1971) discuss how data are converted between external representations and internal forms that are usually based on mathematical