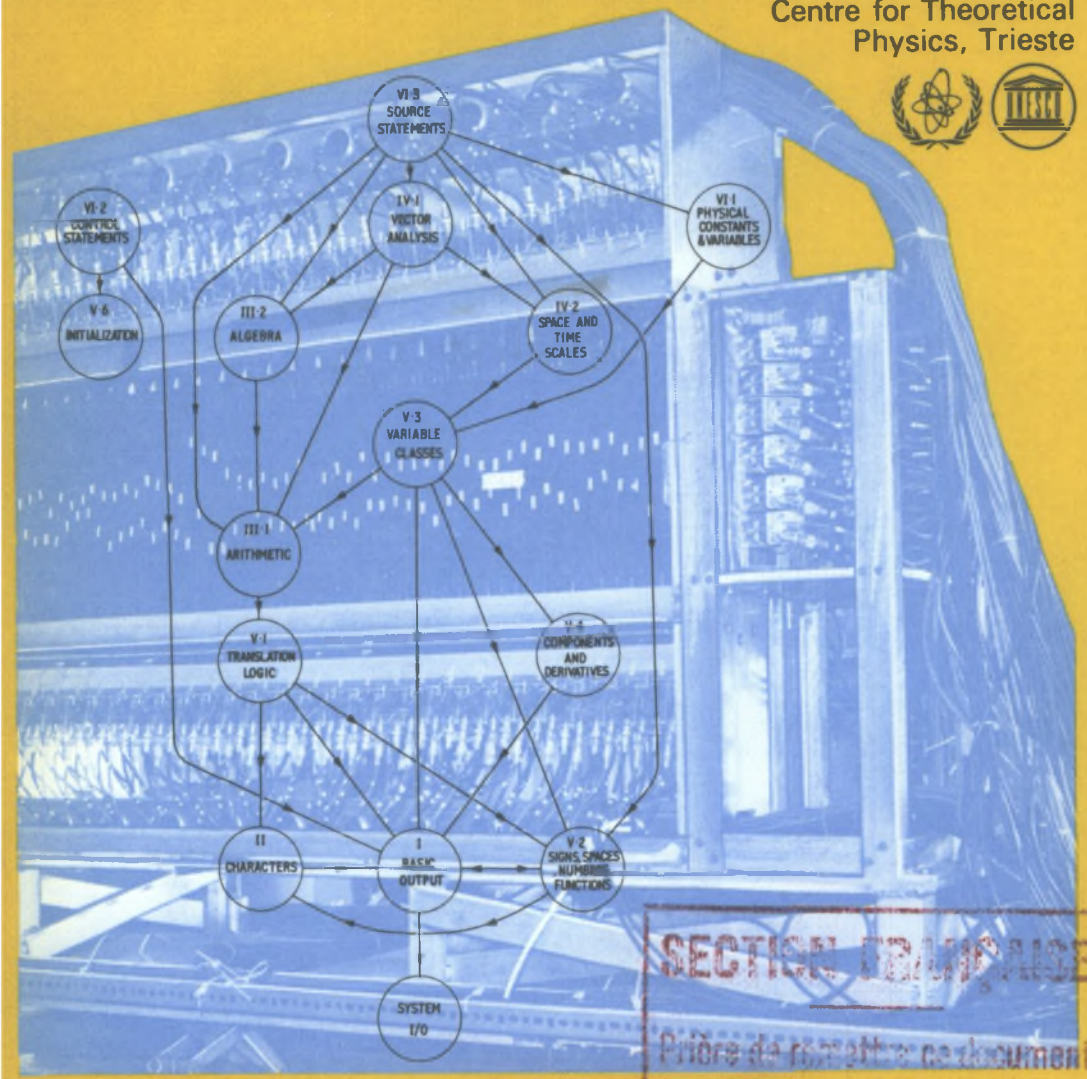
COMPUTING AS A LANGUAGE OF PHYSICS

Lectures presented at an International Seminar Course

Trieste, 2-20 August 1971 organized by the International Centre for Theoretical Physics, Trieste



INTERNATIONAL ATOMIC ENERGY AGENCY, VIENNA, 1972

COMPUTING
AS A LANGUAGE OF PHYSICS

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS, TRIESTE

COMPUTING AS A LANGUAGE OF PHYSICS

LECTURES PRESENTED AT
AN INTERNATIONAL SEMINAR COURSE
AT TRIESTE FROM 2 TO 20 AUGUST 1971
ORGANIZED BY THE
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS, TRIESTE

INTERNATIONAL ATOMIC ENERGY AGENCY
VIENNA, 1972

THE INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS (ICTP) in Trieste was established by the International Atomic Energy Agency (IAEA) in 1964 under an agreement with the Italian Government, and with the assistance of the City and University of Trieste.

The IAEA and the United Nations Education, Scientific and Cultural Organization (UNESCO) subsequently agreed to operate the Centre jointly from 1 January 1970.

Member States of both organizations participate in the work of the Centre, the main purpose of which is to foster, through training and research, the advancement of theoretical physics, with special regard to the needs of developing countries.

COMPUTING AS A LANGUAGE OF PHYSICS
IAEA, VIENNA, 1972
STI/PUB/306

Printed by the IAEA in Austria
December 1972

FOREWORD

The International Centre for Theoretical Physics has maintained an interdisciplinary character in its research and training program as far as different branches of theoretical physics are concerned. In pursuance of this aim the Centre has organized extended research courses with a comprehensive and synoptic coverage in varying disciplines. The first of these – on plasma physics – was held in 1964; the second, in 1965, was concerned with the physics of particles; the third, in 1966, covered nuclear theory; the fourth, in 1967, and the sixth, in 1970, dealt with condensed matter and imperfect crystalline solids; the fifth, in 1969, and the seventh, in 1971, were courses on nuclear structure. The proceedings of all these courses were published by the International Atomic Energy Agency. The present volume records the proceedings of the eighth course, held from 2 to 20 August 1971, which dealt with computing as a language of physics. Grants from the United Nations Development Programme, the Organization of American States, the International Bureau for Informatics-International Computation Centre and the Digital Equipment Corporation made it possible for the Centre to increase the participation of scientists from developing countries.

The program of lectures was organized by Professors L. Bertocchi (Trieste, Italy), L. Kowarski (CERN), S. J. Lindenbaum (Brookhaven and New York, USA) and K. V. Roberts (Culham, UK).

Abdus Salam

EDITORIAL NOTE

The papers and discussions incorporated in the proceedings published by the International Atomic Energy Agency are edited by the Agency's editorial staff to the extent considered necessary for the reader's assistance. The views expressed and the general style adopted remain, however, the responsibility of the named authors or participants.

For the sake of speed of publication the present Proceedings have been printed by composition typing and photo-offset lithography. Within the limitations imposed by this method, every effort has been made to maintain a high editorial standard; in particular, the units and symbols employed are to the fullest practicable extent those standardized or recommended by the competent international scientific bodies.

The affiliations of authors are those given at the time of nomination.

The use in these Proceedings of particular designations of countries or territories does not imply any judgement by the Agency as to the legal status of such countries or territories, of their authorities and institutions or of the delimitation of their boundaries.

The mention of specific companies or of their products or brand-names does not imply any endorsement or recommendation on the part of the International Atomic Energy Agency.

CONTENTS

PART I: GENERAL INTRODUCTION

COMPUTERS AND PHYSICS

K. V. ROBERTS
UKAEA Research Group,
Culham Laboratory,
Abingdon, Berks,
United Kingdom

Abstract

COMPUTERS AND PHYSICS.

This introductory paper begins by discussing from a fairly fundamental point of view what computers really do and why they should be important to physics. For example, how significant has their impact been in the quarter of a century which has elapsed since the electronic digital computer was invented, and what may be expected of them in the future? How can we ensure that they realize their true scientific potential and that massive programming effort is used to maximum effect? Does computational physics have something to contribute to computer science and software engineering? A brief look is then taken at one particular field of computational physics, namely the numerical solution of sets of coupled partial differential equations which describe the time evolution of classical systems.

1. INTRODUCTION

I shall begin this introductory paper by discussing from a fairly fundamental point of view what computers really do and why they should be important to physics. How significant has their impact been in the quarter of a century which has elapsed since the electronic digital computer was invented, and what may be expected of them in the future? How can we ensure that they realize their true scientific potential and that massive programming effort is used to maximum effect? Does computational physics have something to contribute to computer science and software engineering? I shall then take a brief look at one particular field of computational physics, namely the numerical solution of sets of coupled partial differential equations which describe the time evolution of classical systems.

An excellent review of the subject is given in the book Computers and their Role in the Physical Sciences, edited by Fernbach and Taub (1970). This describes the origin of the electronic digital computer (in which computational physics played a considerable part) and gives many references. More specialized papers can be found in the Journal of Computational Physics (Academic Press), Computer Physics Communications (North-Holland) and the annual review series Methods in Computational Physics (Academic Press). The International Physics Program Library, operated by Queen's University, Belfast, in association with Computer Physics Communications, has recently been established to publish the programs themselves in digital form.

Figure 1 indicates the main branches of computational physics, together with certain fields which might more properly be regarded as part of computer science (languages and translators) or software engineering (operating systems). The relation between these fields and computational physics may be compared to the relation between mathematics and theoretical physics, or between engineering and experimental physics (Fig. 2). Good languages and good operating systems are vital to the proper growth of computational

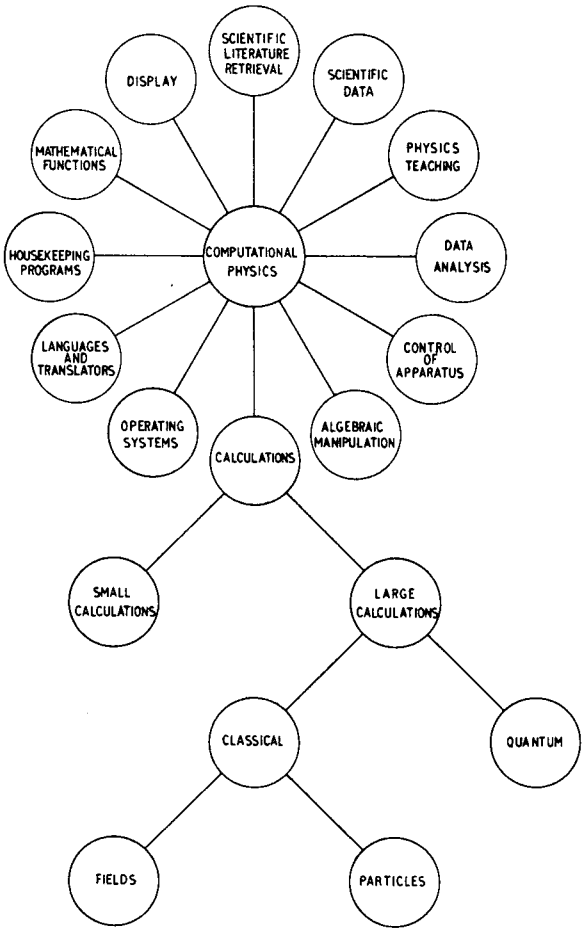


FIG. 1. Some of the areas in which computing has an impact on physics.

physics and therefore physicists can be expected to play a part in their development just as they have always done in many branches of mathematics and engineering.

Figure 3 represents the interplay between the three main ways of approaching a physical problem: experimental, theoretical and computational. Each has its characteristic methods of approach, its advantages and limitations, some of which will be mentioned below.

1.1. Theoretical physics

Theoretical physics makes considerable use of analogies, many of which are geometrical in character; for example, the calculus was originally based on the idea of gradients and areas. Familiar concepts in

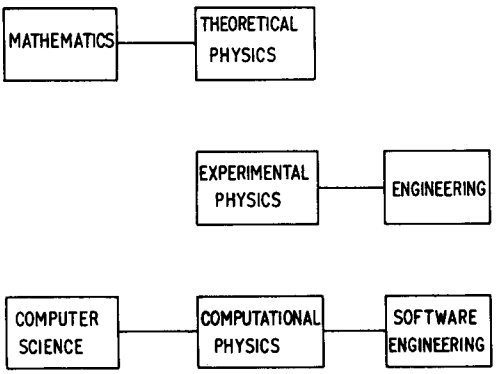


FIG. 2. In the past, mathematics and theoretical physics have been closely associated with one another and similarly for experimental physics and engineering. Computational physics requires advanced techniques in computer science and software engineering and in turn may be expected to contribute to these disciplines.

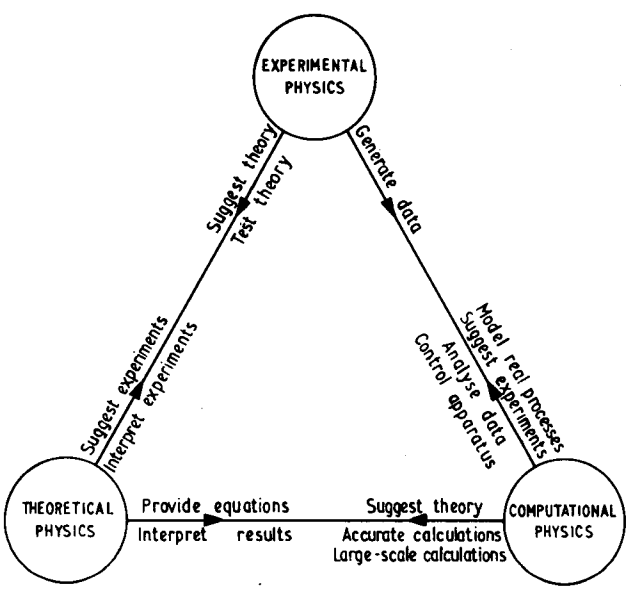


FIG. 3. Interplay between the three main ways of approaching a physical problem.

three dimensions are generalized to n or to an infinite number of dimensions. Theoretical physics relies heavily on the use of symbolism, enabling many actual cases to be described by a single algebraic formula; it is position-free, since it can survey any portion of space-time with equal ease, for example, the inside of a neutron star at some distant epoch; and it is scale-free, ranging at will from the scale of a quark to that of the whole universe, and from 10^{-23} seconds to 10^{10} years. It is universal, in the sense that one piece of theory, such as Coulomb's law or Laplace's equation, can be applied to innumerable actual situations.

Theory makes extensive use of linearization. There is almost a motto, "When in doubt, linearize". Any linear process is relatively easy to solve by analytic techniques, and weakly non-linear processes by perturbation theory. Strongly non-linear processes are much more difficult. Symmetry and conservation laws are related to one another and play an essential role, not only in basic theory but also in the solution of practical problems, as by the method of separation of variables. Complex function theory has a similar dual role; it appears to be fundamental to high-energy physics and to the theory of ordinary differential equations and at the same time is of great practical use in the solution of two-dimensional problems because of its relation to Laplace's equation. Many of the mathematical methods used in theoretical physics have been summarized by Morse and Feshbach (1953).

Approximation techniques are essential. Sometimes this means separating out a few of the many degrees of freedom of a large system, or distinguishing between widely different time-scales as in the method of adiabatic invariants. In other cases such as statistical mechanics the number of degrees of freedom is treated as infinite since this makes the formulae much simpler.

These are some of the mathematical tools; practical tools include pencil and paper, chalk and blackboard, books, journals and meetings. Theoretical physics is cheap but it requires high IQ. Another important feature is that theory is self-enhancing; by practicing it, one becomes a better theoretician. This is not necessarily true of experimental physics (which requires the organization of staff and finance, the building of apparatus and the management of contracts), nor of computational physics (which involves struggling with awkward and unreliable computing systems, much handling of cards and paper, and a continual search for errors in the programs). A major task will be to build this feature of 'self-enhancement' into computational physics by improving the techniques.

1.2. Experimental physics

Experimental physics provides the ultimate test and source of information for theory, just as theory provides the equations to be solved by computation. With great ingenuity the scope of experiments and observations has gradually extended both ways from human scale of the range 10^{-13} cm to 10^{10} light years in length, and 10^{-23} seconds to 10^{10} years in time. But experimental physics is neither position-free nor scale-free, and the cost of an experiment depends very much on the scale of the phenomenon which is under investigation. Where the expense is high, it may be preferable to use theory or computation, although experimental modelling is often also of great use.

No experiment is exact and the potential sources of error must always be carefully examined.

1.3. Computational physics

Computational physics combines some of the features of both theory and experiment. Like theoretical physics it is position-free and scale-free, and it can survey phenomena in phase-space just as easily as real space. It is symbolic in the sense that a program, like an algebraic formula, can handle any number of actual calculations, but each individual calculation is more nearly analogous to a single experiment or observation and provides only numerical or graphical results.

To some extent it is possible to solve the equations on a computer without understanding them, just as one can carry out an exploratory experiment. With more complicated phenomena involving a considerable range of length and time scales it is, however, essential to make analytic approximations before putting the problem on to the computer, otherwise impossibly large amounts of machine time or storage space may be needed. Not more than about 10^6 degrees of freedom can be handled on present-day computers, or 10^3 if they all interact with one another. Thus computational physics can fill in the range between few-particle dynamics and statistical mechanics.

Diagnostic measurements are relatively easy compared to their counterparts in experiments. This enables one to obtain many-particle correlations, for example, which can be checked against theory. On the other hand, there must be a constant search for 'computational errors' introduced by finite mesh sizes, finite time steps, etc., and it is preferable to think of a large-scale calculation as a numerical experiment, with the program as the apparatus, and to employ all the methodology which has previously been established for real experiments (notebooks, control experiments, error estimates and so on).

Computational physics is particularly suitable for non-linear, non-symmetrical phenomena where the usual theoretical methods do not apply (such as in weather calculations), but often the programs are easier to write and the calculations go much faster in simple situations such as rectangular Cartesian geometry with rigid, perfectly conducting walls.

It is often possible to take situations that normally are only handled algebraically and to display them in pictorial form. Thus computing can put life into somewhat abstract subjects and might be of great help, for example, in the teaching of complex variable theory.

Finally, there is great danger if computational physicists become too preoccupied with mundane details of computing at the expense of the physics itself, but the only solution here seems to be to get the details right once for all, just as at one stage it was necessary to introduce rigorous limiting processes into mathematics.

1.4. Examples

Sometimes one method of approach will be more appropriate and sometimes another; frequently they will work in pairs and at times all three methods must be used together. An example where computational techniques

are particularly appropriate is in the solution of equations which describe the internal structure and evolution of stars (Iben, 1970). The equations are complicated and non-linear but they are well-defined, and provided that spherical symmetry can be assumed, they are well within the range that computers are able to handle. On the other hand, analytic methods have difficulty because of the non-linearity, while it is clearly awkward to do experiments or even to make observations (except with neutrinos) in the interior of a star.

The book by Betcher and Criminale (1967) on Stability of Parallel Flows gives a good account of the way in which analytic and computational techniques can support one another in one branch of fluid dynamics. Harlow (1970) has provided a general bibliography of papers dealing with numerical techniques for solving two- and three-dimensional time-dependent problems in fluid flow.

1.5. Physics and information

The purpose of a computer is to process information. Physicists do spend a great deal of time handling information of one kind or another and any impact that computers have on physics must eventually result from this fact. Many of the techniques used for handling scientific information have reached a high degree of sophistication, particularly in theoretical physics, and here it is likely to take a long time before computers can compete on equal terms; for example, the developments in the physical sciences which occurred within 5 years due to the discovery of Schrödinger's equation can hardly be paralleled by those which have occurred within 25 years due to the invention of the electronic digital computer. But in cases where conditions have been more suitable for the introduction of computers, such as the processing of large amounts of digital data from measuring devices and the automatic control of experimental equipment, their impact has been more obvious.

2. HARDWARE

Let us therefore go right back to the beginning and try to see what computers can in principle do. Basically, an electronic computer is a device for handling binary information or data contained in a fast memory or store. The data is conventionally represented as an ordered set of 0's and 1's (bits), grouped into bytes and words. In processing this data the computer obeys a sequence of instructions which are themselves represented by binary information and are drawn from the same store (Fig. 4). The sequence of instructions is called a program.

It is preferable to think of the program as fixed information, while the data will in general vary during the course of the computation. There is in fact an interesting analogy between a data processor and a dynamical system, in which the program corresponds to the Hamiltonian $H(q,p)$ while the data values correspond to the complete set of canonical co-ordinates and momenta $\{q,p\}$ which between them define the current state of the system. As the computation proceeds, the progressive modification of the data by the program corresponds to the time evolution of the dynamical system.

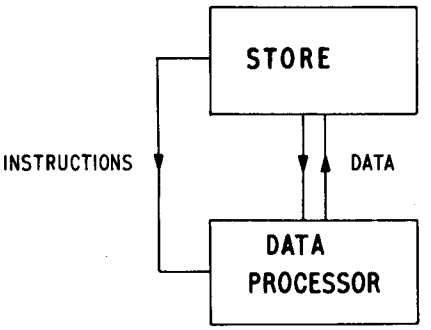


FIG.4. The computer processes data by means of a sequence of instructions, both being drawn from the same store.

The data values can be made to control:

(a) The action of the current instruction.
(b) The location of the next instruction to be obeyed.

This facility enables the program to make decisions which depend on the current data values and is nowadays usually implemented by first transferring the necessary control information from the main store into subsidiary fast storage devices called registers, one or more of which can be consulted while an instruction is being interpreted.

2.1. Dynamic program modification

Because the program instructions can also be regarded as data it is possible, in principle, for a computer to process its own instructions during the course of a run. This is quite a fundamental idea because it means that the program itself can evolve dynamically, as well as the data values. At one time this property was regarded as essential (Goldstine and Von Neumann, 1963; Elgot and Robinson, 1964; Goldstine, 1970), but it seems that the essential tasks have now been taken over by the use of registers, and self-modifying programs are currently regarded as bad practice because they are so difficult to understand. For example, no legal FORTRAN or ALGOL program can modify itself.

In mathematics, one sometimes finds that a generalization is remarkably productive and leads to a host of new results (real → complex numbers); at other times, it almost seems to kill progress altogether (time-dependent Hamiltonians $H(q,p,t)$ or non-Hamiltonian systems; groups → semi-groups). We do not know on which side of the fence these dynamically, self-evolving programs are likely to lie. I shall not discuss them further here but it may be that this is an area where substantial advances in computational physics will be made one day.

2.2. Universality of hardware

There is a sense in which all computers are the same (Pasta, 1970):

> "As an example of the kind of thing we are talking about, consider the Turing machine, a model invented in the 1930's by the mathematician A. M. Turing. This abstract model is very simple. In one form it is a device with a finite number of internal states and a tape of arbitrary length marked into squares. At any moment it can read a symbol on the tape. Based on that symbol and on the internal state, the machine can initiate actions to change the symbol and to move the tape one square left or right.
>
> One would expect such a machine to be limited in the kinds of things it could do and yet Turing showed that any effective computation performed on any computer can be performed on a Turing machine. The universality of this machine allows us to establish truths about it which will apply to all other machines and consideration of this and other equivalent models has increased our understanding of computers, programs and computations, all of which can be fitted into this simple model."

Turing's theorem suggests that any fundamental advances in computational physics are much more likely to come from better theoretical techniques, from improved algorithms and languages or from software engineering than from improved hardware. During the past 25 years there have been steady quantitative improvements in the architecture of computers and in their speed, storage size, reliability, versatility and convenience, together with a parallel decrease in the cost per unit of computation, but there have been no radical changes of principle.

2.3. Some practical improvements

There are, however, a number of potential improvements of a practical kind whose combined effect might be so dramatic as to appear fundamental. These include:

(a) Networks of computers linked together via the communications system
(b) Massive direct-access storage devices
(c) Ultra-high-speed character and vector displays
(d) An extended character set, including the Greek alphabet and mathematical symbols
(e) Improved ergonomics of man-machine interaction
(f) Further decreases in cost, and improvements in reliability of on-line systems

These developments might make it practicable for a 'power-assisted' algebra facility to be introduced, by which a theoretical physicist working at a console could automatically and almost instantaneously manipulate analytic expressions appearing on the screen by issuing commands to the system to perform standard transformations and integrals. This has already been partly implemented at Stanford University in an experiment

on the teaching of elementary algebra in schools, but in order to compete with pencil and paper it is important to get the practical details right.

Very fast, powerful and selective information-retrieval facilities might also become possible, enabling a scientist working in one field to familiarize himself rapidly with the state of the art in another. In this connection, a fundamental technique that has been developed in computer science might well be applied to reduce the bulk of the regular scientific literature, namely that of the subroutine or macro. Theorems, diagrams, formulae, definitions, conventions, etc., which are constantly being reproduced in full, could be stored in one place and automatically called into use when required, simply by naming them. At the same time, the notation could be automatically changed to fit that of the paper in which they were called.

Another possibility is to have a dynamic style of publication, containing not only algebraic formulae but programs for evaluating them numerically or displaying them graphically on a screen as a function of parameters selected by the 'reader'.

3. DATA TRANSFORMATIONS

So far, we have only considered binary strings of 0's and 1's. These are not in themselves very interesting and their importance lies in the ease with which they can be transferred to and from other types of data format (Fig. 5). Binary or 'digital' data is freely interchangeable between electrical signals, magnetic recording media and holes in punched cards or paper tape, although at different speeds. Electrical information can readily be converted from analogue to digital form and vice versa, although with some loss of content. Apart from this, it should be emphasized that output by the computer is usually much faster, cheaper and more convenient than input as illustrated by the dashed lines in Fig. 5. It is relatively easy for a computer to display a table, draw a graph or make a movie film or even to talk, but much harder to get this information back into digital form. Therefore, so far as computers are concerned, digital information ought to be regarded as the primary form, while printed output, graphs, speech, etc. are temporary forms intended only for communication with people.

3.1. Digital information

Digital information has a number of important advantages. It can be transmitted almost instantaneously from point to point, updated, duplicated, stored and retrieved, automatically manipulated in different ways and displayed to people in a variety of forms. We can in fact regard a set of data as an operand D and a display program as an operator P_i, various forms of display Δ_i being generated as products

$$\Delta_i = P_i D \qquad (1)$$

If, for example, a calculation leaves its output in a random access file, then not only can a physicist working at a console cause the results of the

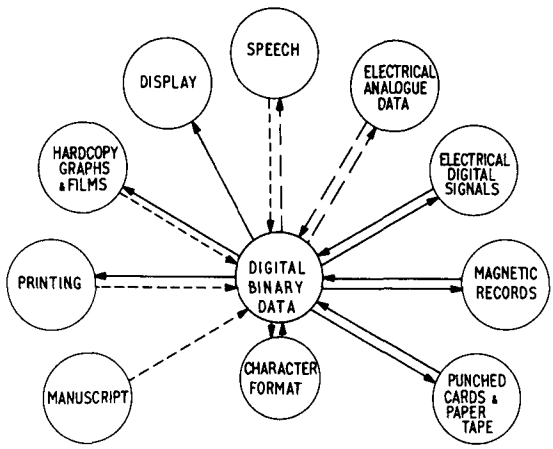


FIG. 5. All forms of digital data are freely convertible into one another (full lines). Printing and display are also straightforward. Analogue-digital conversion can be carried out without difficulty and a computer can be made to talk (long dashes). Those transformations which are represented by short dashes are much more difficult to carry out and should be avoided where possible by storing all data in digital form.

calculation to be displayed in various ways so that he can understand their meaning, but he can also use the same file as input for a further series of calculations. These advantages are lost if the output file is simply printed and then destroyed.

Digital information does, however, have a number of grave disadvantages which must be carefully taken into account if it is to serve as a medium for scientific communication. It is extremely fragile, and on many computer systems even minor damage to an index can cause all the data on a storage device to be lost. Few of the scientific discoveries of antiquity would have survived if their recording media had suffered from this disability. And digital information does rely heavily on good indexing; compare browsing through a magnetic disc file with browsing through a library of scientific books.

4. ALGORITHMS, PROGRAMS AND SOFTWARE

Computers can carry out any process which we know how to reduce to algorithmic form; that is any process for which we can prescribe a definite set of rules no matter how complicated. Ultimately this process must be reduced to the manipulation of a binary bit pattern and the algorithm itself must be expressed in a similar form (Fig. 4), but in practice we can develop our algorithms in a more convenient language and then use a second algorithm to carry out the conversion automatically (Fig. 6). In fact, a primary input device such as a teletype usually performs a preliminary conversion to binary form, and this is then subsequently transferred by one or more system programs such as compilers, link editors, etc. until the binary instruction code of the machine is finally reached.

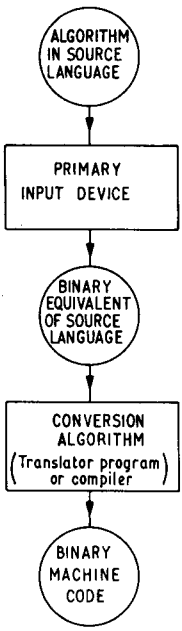


FIG. 6. An algorithm can be expressed in any 'source language', a second algorithm or sequence of algorithms being used to convert this automatically into binary machine code.

Three requirements are:

A. It must be possible to find an algorithm to carry out the required process.
B. The algorithm must be coded for a specific machine.
C. The number of computer operations required for the process must not be too large.

Much of the effort in computational physics at the present time is occupied by requirement B, and since this is rather a mechanical task it tends to divert attention from the physics proper. However, just because it is a mechanical task it should itself be automated. The ultimate solution is one in which the languages which are most suitable for people who are investigating and expressing the algorithms are also intelligible to computers and can be automatically converted by them into efficient binary code. Some comments on how this may be achieved will be made in Section 8, in connection with Symbolic ALGOL.

Algorithms for some of the processes used in physics have existed for many years, for example, arithmetic, and the solution of sets of coupled ordinary differential equations by finite difference methods. Here the computer was able to make an immediate impact. In high-energy physics a great deal of effort has been put into algorithms for pattern recognition in connection with the processing of bubble chamber data, and with considerable success (Snyder, 1970; Kowarski, 1970). Some success has been achieved with

automatic theorem proving and with the automatic solution of elementary integrals by analytic methods, but neither have influenced physics as yet. In other cases where theoretical physicists have no algorithms and must proceed intuitively, as in the formulation of new concepts, computers have also naturally had little influence.

4.1. Algorithmic and programming languages

In order to satisfy requirements A and B it will be necessary to develop:

D. Powerful, intelligible algorithmic languages.
E. A substantial body of algorithms expressed in these languages, for the solution of physical problems.
F. Means for converting these algorithms into efficient binary machine code for the various types of computer system.

A high-level programming language such as FORTRAN or ALGOL enables algorithms to be expressed in a form which is relatively convenient for people to use, while at the same time allowing them to be translated without too much difficulty into reasonably efficient machine code. Algorithms are in many ways similar to mathematical theorems and need to be made intelligible and universal for the same reasons. Unfortunately the existing languages cannot be compared in scope to mathematical notations such as non-commutative algebra and the tensor calculus. Furthermore, it is nowadays very difficult to introduce a new programming language because of the cost of developing and maintaining the necessary translators or compilers for a variety of different computer systems. The result has been that for physicists the state of the art has remained frozen for many years; although many research languages have been developed by individual computer scientists during the last two decades, only FORTRAN (introduced in 1957) and ALGOL (introduced in 1960) are of major importance in physics. These have awkward deficiencies which in principle could be easily put right, but which remain uncorrected because of the difficulty of reaching international agreement and then modifying all the existing compilers. The restriction to six-character identifiers in FORTRAN and the omission of complex numbers, COMMON and EQUIVALENCE declarations and standard input-output facilities from ALGOL are typical examples.

It was hard for mathematics to progress until a good notation had been introduced in order to express the operations of arithmetic and algebra (Ball, 1908); try calculating in Roman numerals! It has also been said that the development of English mathematics was held up for more than a century by reliance on the methods and notation of Newton rather than those of Leibnitz. Computational physics is likely to remain equally constrained until it becomes a straightforward matter to introduce powerful new notations in which algorithms can be expressed. Even the hardware restriction to upper-case letters, numerals and a few special characters constitutes a severe limitation, compared to the great variety of symbol types, sizes and positions which are exploited in mathematics.

The solution appears to be for scientists themselves to develop and publish machine-independent or portable compilers, program generators and macro-processors in addition to the growing literature of application

programs and packages. If these are written in modular form and well documented it should be relatively straightforward to extend them to meet new situations in the same way that mathematical theorems are continually generalized and extended.

5. A SCIENTIFIC SOFTWARE LITERATURE

Clearly, theoretical physics would hardly progress at all if every worker had to build up all the mathematics that he needed right from the beginning, and it will not be practicable to develop the enormously complex algorithms and programs that will be required in computational physics unless each individual is able to stand on the shoulders of his predecessors.

5.1. Coding problem

It might be argued that although the algorithms themselves should be published in the regular scientific journals, coding them for a specific application should be left to the individual worker. This, however, is unrealistic because of the very high cost of coding and because of the long delays involved.

Some figures have recently been published on the costs of computer software and the effort needed to write it. For IBM-360 software the cost of each instruction has been estimated at $50-60, with 0.2 instructions produced per man-hour (Bemer, 1970). Figure 7 shows the growth in software requirements in terms of lines of code for successive machines (McClure, 1969), while Fig. 8 expresses it in terms of millions of man-hours spent (Bemer, 1970). Both increase exponentially with time, by a factor of about 200 in 10 years, and it seems that both Parkinson's law and the Peter Principle must surely be in operation (David, 1969). Bemer remarks: "My nightmares come from imagining a new system scheduled for 1972. If the McClure chart holds true to give 25 million instructions, then the best figures we have say it will cost a billion and a quarter dollars, produced by 15 000 programmers." Yet, according to Barbe (1970), only 2% of the $36 billions' worth of software in operation in the United States is transferable from one computer to another; the rest is doomed to die with the hardware.

Physicists may be doing a little better, since Snyder (1970) estimates that a 60 000-word bubble-chamber analysis program written in FORTRAN might require 10 man-years of programming effort which would represent a coding rate some 15 times faster.

5.2. Publication, portability and modularity

Since computational physicists do not generally have this amount of money to spend, the operating systems, compilers and applications programs which they need will not get written unless some better method is found. It does appear, however, that three techniques which have worked well in science and mathematics in the past could go a long way towards solving the problem.

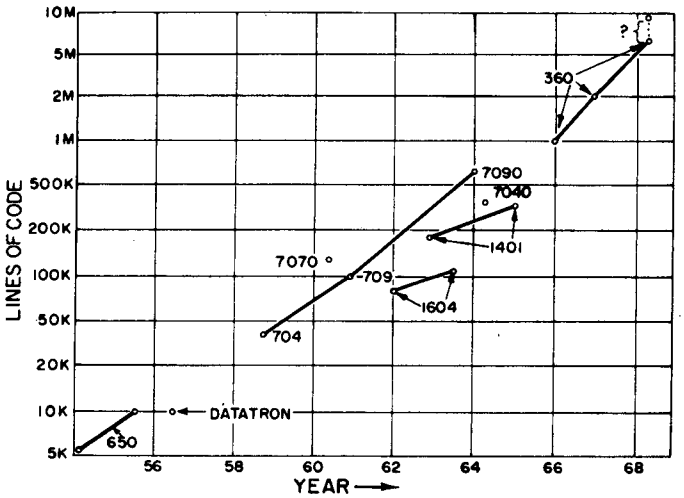


FIG. 7. Exponential growth of the software required by operating systems (McClure, 1969). In 10 years the amount of software associated with a typical scientific computing system has increased by a factor 200.

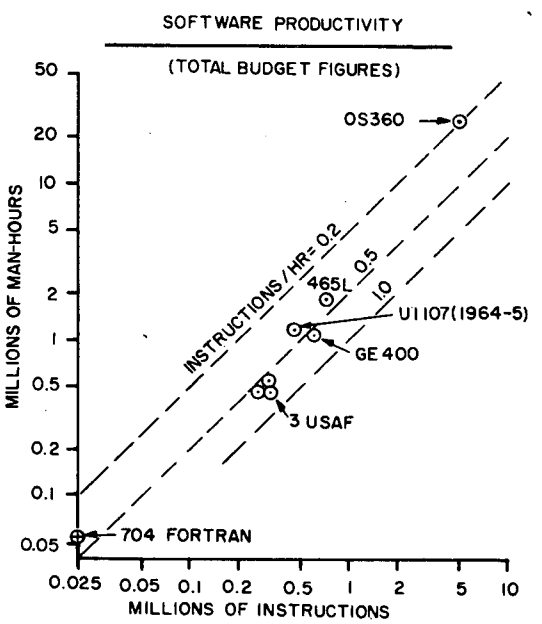


FIG. 8. Effort spent on software construction (Bemer, 1970). Because productivity has not increased with time, both the cost and the effort required for software construction are increasing exponentially at a similar rate to that of Fig. 7.

The first technique is that of open publication. The new journal Computer Physics Communications (North-Holland) has recently been founded to publish details of well-documented, refereed, tested physics programs. Associated with this is the International Physics Program Library at Queen's University, Belfast, which publishes the programs themselves in digital form. I have argued elsewhere the advantages of such a scheme (Roberts, 1969). One important advantage is that by 'exposing' the program listings to public criticism the standards are likely to be forced up. A primary reason why the standards in software engineering are so low is that there are so few models to work from, because programs are regarded as commercially valuable and are not therefore seen by more than a few people. The existence of a high-quality open scientific program literature should serve as a stimulus to the whole computing industry, just as the regular scientific and mathematical literature of books and journals does for technology.

The next technique is that of portability, which is the same as 'universality' in science and mathematics. Once a new program, subroutine package, compiler or scientific operating system has been developed and published, it should be possible to run it at any scientific laboratory or university throughout the world, just as one can read any journal article or textbook. There are two basic requirements for this:

(a) Scientific libraries must be persuaded to subscribe to the journal tapes, in the same way that they do to the regular scientific journals, and to make their contents as readily available as are books and papers.
(b) The published programs must be written in universally available languages.

At present, only the universal high-level languages FORTRAN and ALGOL are accepted by Computer Physics Communications. An important further requirement is a lower-level universal language in which compilers can be written and in which they can generate their output (Fig. 9). As soon as this is available, a single implementation of each new language will make it available on all machines, thus saving excessive duplication of effort and averting the danger of different implementations being out of step, as happens with FORTRAN and ALGOL at the present time.

This idea was proposed many years ago, in connection with the so-called Universal Computer Oriented Language or UNCOL (Mock et al., 1958). It enables N languages to be implemented on M machines with a total amount of effort N + M instead of NM. Another possible way of implementing the idea is by means of macro-processors (Poole and Waite, 1970). It seems unlikely that new scientific languages can be universally introduced except in some such way as this.

Figure 10 illustrates what I believe the structure of the scientific software literature Σ should eventually be; it has been drawn to parallel Fig. 1 which shows the structure of computational physics itself. Note that it includes the regular scientific literature, since I have assumed that in due course books and journals (or at least automatic indexes to them) will be made available in digital form. There is a significant danger here. In the past, the scientific literature has always been completely 'visible', even though it has been published, in large part, by commercial firms.

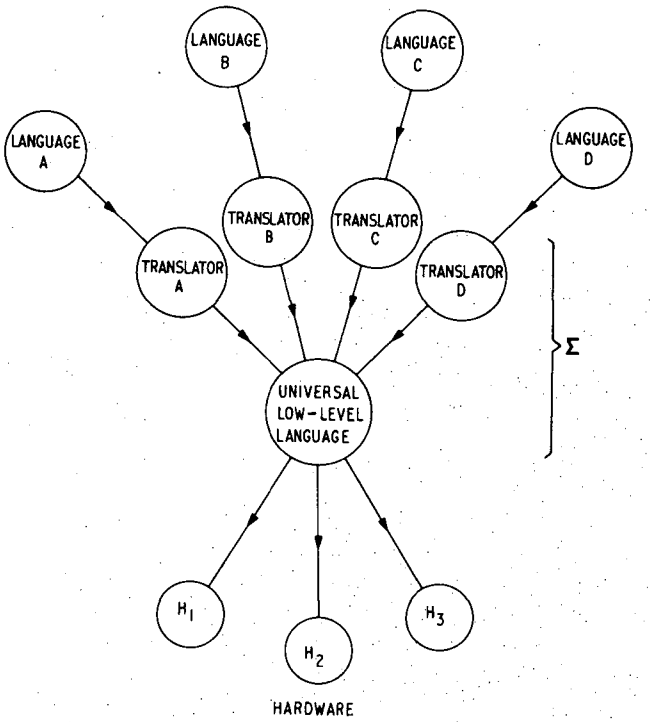


FIG. 9. It is now difficult to introduce new scientific programming languages because of the need to reach agreement on standardization and the cost of constructing a new compiler for each type of computer system. A better solution would be to construct just one compiler which would then be published. Both the compiler itself and its target code must be expressed in a suitable universal language so that they can be used on any system.

If it ever gets transferred to proprietary data banks which can be automatically consulted, for a fee, but can never be openly inspected by the scientific community as a whole, then there is a great danger that it will become corrupted. Even the standard FORTRAN library functions often contain mistakes.

Once established, Σ may be expected to increase steadily with time like the regular scientific literature and to be equally permanent; there are already programs that have been in use for more than 10 years and which have been run on a whole series of machines. At any given epoch, Σ will be run on a variety of different hardware types H_1, H_2, H_3 .... As Σ expands, it will be less and less economically practicable to recode even major portions of it for each new hardware system H_α, and this is why portability is essential. The most that can be done will be to recode certain replacement modules R_1, R_2, R_3. ... (Fig. 10) which are executed with very high frequency and so occupy a substantial fraction of the computer time.

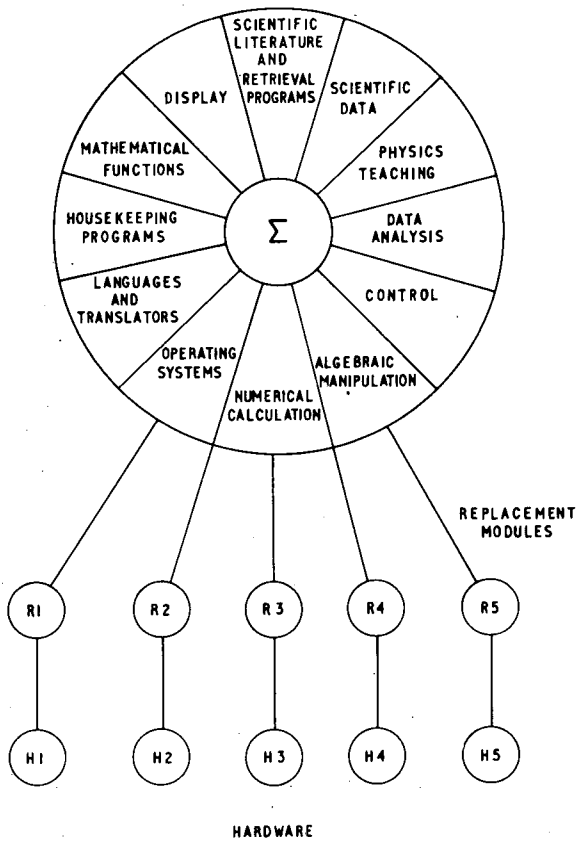


FIG.10. Programs and data of significance to science should be published in digital form to ensure that they are freely available and that their efficiency and reliability can be checked. This 'digital literature' Σ should include not only scientific applications programs but also operating systems and compilers. Most of it should be expressed in universal form so that it can be used on any machine. Replacement modules R_i written in assembly language are used in the interests of efficiency for those modules which have a high execution frequency.

The third important technique is modularity, which is the same as the 'Principle of Abstraction' in mathematics. Theoretical physics works by developing a number of separate tools, e. g. vector algebra, tensor calculus, group theory, Green's functions, Laplace's equation, and then combining them together in many different ways. This means that when a new branch of theoretical physics has to be mapped out, much of the necessary mathematics is already available (as with Schrödinger's equation and quantum mechanics in 1926). It also means that theoreticians can often move freely from one field to another because they recognize the language.

Suppose that we build a set of program modules of n different types, with m modules of each type. Then, by combining these together in all possible ways, the number of complete programs we can form is of order m^n, while the work required is only of order mn. Even allowing for many non-viable combinations, this is still a considerable advantage. To put it in another way, suppose that a single new module is developed of one particular type; then m^{n-1} new programs can in principle be constructed from it, an amplification factor of Nm^{n-1}.

Typical examples might be the introduction of a new type of co-ordinate system (e. g. spherical polars), or a new graphical display package. Provided that the existing programs are properly constructed, many of them can quickly make use of these with little further effort.

6. THE POWER AND LIMITATIONS OF COMPUTERS

I have stressed the organizational problem at some length because this is the single most important practical task facing computational physics at the present time. There are many algorithms in the literature which are not being exploited because of the effort needed to code them. There are many good programs that can only be used in one or two major laboratories (notably the Los Alamos hydrodynamics codes), and others which have gone out of use because their originators moved on to other work. Also, there are large numbers of significant research languages which have not moved very far from the computer science departments where they were developed; meanwhile, FORTRAN has been frozen since 1964. However, these are all problems which can be solved by persuasion and good planning, along the lines I have already indicated. A more basic problem is whether or not there are any fundamental limitations on the use of computers in physics.

It is sometimes thought that computers will eventually kill theoretical physics; all that one will need to do is to program the equations and press the button in order to get a numerical answer. This is very far from being the case. Consider an assembly of N particles, interacting via Newton's laws of motion and gravitation. If N is small (say equal to the number of planets together with the sun), then it is indeed possible to solve the equations rather accurately over long epochs using the computer, and in this sense one might say that much of the analytic work done in the 18th and 19th centuries on the classical few-body problem in astronomy was not strictly necessary. Fortunately, computers were not available then because the modules developed during the course of this work (e. g. Lagrangian and Hamiltonian mechanics and perturbation theory) turned out to be of great use in other fields such as quantum mechanics and statistical mechanics.

Because the number of elementary interactions between N particles increases as N^2, straightforward computational techniques become impracticable as soon as the number of particles greatly exceeds 100 or, at most, 1000. Statistical mechanics is difficult to apply because of the infinite potential energy that can be released when two gravitating particles approach each other, and the two lines of attack, theoretical and computational, must support one another.

Theoretical physics relies to a large extent on finding adequate approximations. Often this is a question of separating the various time-scales in a problem. For example, the Born-Oppenheimer method used in molecular theory treats the nuclei as fixed when calculating the electron energies and wave-functions from which one obtains a potential function to be used in solving the motion of the nuclei themselves. Time-scales are equally important in computational physics because if a naive approach is adopted, the cost of the calculation will be determined by the shortest time-scale t_{min} of the problem and will rise to astronomical values if the ratio of this to the largest time-scale t_{max} becomes too great.

In the case of an assembly of N gravitating stars the shortest time-scale is likely to be determined by the orbital periods of close binaries which can decrease without limit. These must somehow be decoupled from the problem, e.g. by treating the motion analytically until the perturbations due to nearby stars become too great. The N^2 difficulty might be removed by replacing the effect of the interactions outside a given distance by that of a mean field, so that the amount of calculation increases only as N.

Research of this type often proceeds in one of two ways:

(a) Theoretical approximations are devised to remove difficulties encountered in the computation, and then these approximations are verified using the computer.
(b) The numerical calculations turn up unexpected and striking results, which can then be given a simple analytic explanation.

Thus the theoretical and computational approaches are complementary to one another.

One instance where computers could have been of great assistance during the 19th century is in the solution of the Navier-Stokes equations for viscous flow. If these equations had been solved numerically in two dimensions at moderate Reynolds numbers, boundary layers of finite thickness would have automatically developed in the neighbourhood of solid surfaces, and the interpretation of this phenomenon should have led to the discovery of boundary layer theory and an understanding of the problem of flight much earlier than actually occurred. Shocks and Karman vortex streets would have automatically turned up in a similar way.

It is interesting to notice another complementarity between the theoretical and computational approaches, since theory finds it easier to deal with thin boundary layers, while computers find it easier to deal with thick ones (covering several space steps).

6.1. Partial differential equations

When we turn to partial differential equations the limitations of computers become even more apparent. Excluding high-energy physics for which the equations themselves are not well defined but their number seems to be infinite, we find three situations in decreasing order of complexity, as indicated below.

Numbers of dimensions

Schrödinger	Configuration space	3N
Vlasov	Classical phase space	6
Navier-Stokes	Real space	3

Vlasov's equation describing the phase space motion of particles interacting via long-range fields is important in plasma physics, while the Navier-Stokes equations of hydrodynamics are a prototype for many similar sets of coupled partial differential equations in magnetohydrodynamics, astrophysics, geophysics and other fields.

Assuming that we need at least 100 space points in each direction to achieve good accuracy (i.e. 25 Fourier modes with ≧ 4 points/mode), the amount of storage needed is 100^{3N} for Schrödinger's equation, 100^6 for Vlasov's equation, and 100^3 for hydrodynamics.

It is now just becoming practicable to compute with 10^6 mesh points using machines such as the CDC STAR-100 so that three-dimensional hydrodynamics problems should shortly be fairly routine provided that the Reynolds number is not too high. To achieve the same accuracy with Vlasov's equation requires a further factor of 10^6 in storage capacity and speed which is difficult to envisage at the present time, although a factor of 10^3 can perhaps be anticipated. But this method of solving Schrödinger's equation is out of the question for all but the simplest situations.

One is again led to the need for making adequate approximations before putting a problem on to the computer and this of course is done in quantum mechanics, for example, by the method of molecular orbitals (Clementi, 1970). In general, insight is likely to come not only from the numerical results themselves but also from studying the accuracy of the various approximations and trying to understand why they work as they do.

6.2. Turbulence problem

It has recently been pointed out by Emmons (1970) that a straightforward numerical attack on the problem of hydrodynamic turbulence in three dimensions is doomed to failure, since to solve the simplest turbulent pipe flow problem would require 10^{10} mesh points and 10^{14} operations altogether for a Reynolds number $R_e = 5 \times 10^3$, occupying perhaps 100 years on existing computers (or 10^{22} operations and the full age of the universe at $R_e = 10^7$). Here again one must look for a combination of more subtle computational techniques combined with physical insight and good theoretical approximations.

7. DISPERSION RELATIONS

When a partial differential equation is solved on a computer, one effect is to change the dispersion relations of linearized perturbations or small-amplitude waves. This happens because derivatives are replaced by differences, so that, for example,

$$df/dx \rightarrow \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \qquad (2)$$

$$d^2f/dx^2 \rightarrow \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2} \tag{3}$$

The result is that algebraic dispersion relations are replaced by more complex trigonometric ones, since

$$k \rightarrow \frac{\sin k\Delta x}{\Delta x} \tag{4}$$

$$k^2 \rightarrow \frac{2(\cos k\Delta x - 1)}{(\Delta x)^2} \tag{5}$$

and so on.

Depending on the difference scheme, on the equations themselves and on the ratio of the 'mesh speed' $\Delta x/\Delta t$ to the various characteristic speeds of the problem (where Δt is the time-step), this replacement can cause stable waves to become unstable or damped, and non-dispersive waves (e. g. sound-waves) to become dispersive. A good part of the papers of these Proceedings is concerned with such problems, the situation being quite analogous to the replacement of a continuum by a discrete lattice in solid-state physics.

This analogy with solid-state physics might usefully be exploited further. In particular, since there is a maximum wave-number k_{max} that can be represented on a lattice with finite spacing Δx, when two waves $\vec{k}_1$, $\vec{k}_2$ interact to give a new wave $\vec{k} = \vec{k}_1 + \vec{k}_2$ with $|k| > k_{max}$, this energy must be diverted to some other mode $|k'| < k_{max}$ by a type of umklapp process which is known in computational physics as 'aliasing'. This leads to errors in turbulence investigations, and the energy at high wave numbers must be removed by some form of artificial damping before it can cause damage.

The simplest example of numerical dispersion is given in the solution of the one-dimensional advective equation

$$\frac{\partial f}{\partial t} + v\,\frac{\partial f}{\partial x} = 0 \tag{6}$$

where v is a constant. This evidently describes a wave moving with uniform velocity v, thus preserving its shape unchanged, and the dispersion relation is

$$\omega = kv \tag{7}$$

Making the replacement (4) but keeping Δt small, we find

$$\omega = \left(\frac{\sin k\Delta x}{k\Delta x}\right) kv \tag{8}$$

This means that disturbances of short wavelength propagate more slowly and that for

$$k\Delta x = \pi \tag{9}$$

there is no propagation at all. A pulse can leave a train of waves behind it which may be misinterpreted as a real physical phenomenon, and a function (such as the density or temperature), which according to the differential equations must remain everywhere positive, can take negative values in the numerical calculation.

Equation (6) is significant because it is the prototype of all the hydrodynamic equations, in which the left-hand side occurs as the Eulerian derivative. It is also closely associated with Vlasov's equation. Advective errors are of importance in meteorology where v represents the speed with which disturbances are carried by the wind.

8. SYMBOLIC PROGRAMMING

I discussed earlier the possibility of finding a generalized language which would be suitable not only for the formulation and discussion of algorithms, but also for programming the computer itself. I should like to finish this paper by mentioning how this has been very largely achieved for one particular field, namely the solution of classical field equations for initial-value problems (Roberts and Boris, 1971; Roberts and Peckover, 1971; Kuo-Petravić, Petravić and Roberts, 1971). The method is known as Symbolic ALGOL and is described in detail in papers SMR-9/22 and SMR-9/24 in these Proceedings.

FORTRAN, and more particularly, ALGOL 60, were designed for this dual purpose but have two major weaknesses; firstly, they do not include much of the notation that theoretical physicists normally use, and secondly, they have no power of extension other than through the use of subroutines or procedures.

We have, however, been able to show that by writing ALGOL programs in a particular way, they can be brought into very close correspondence with the notation of vector analysis. For example, the magnetic diffusion equation

$$\frac{\partial B}{\partial t} = \mathrm{Curl}\,(V \times B) - \mathrm{Curl}(\eta\,\mathrm{Curl}\,B) \qquad (10)$$

can be programmed in Symbolic ALGOL I as

```
AB[C1,Q] := B+DT*(CURL(CROSS(V,B))-CURL(ETA*CURL(B))); (11)
```

independently of the co-ordinate system and of the number of dimensions. Most of the notation is obvious but it should be explained that the prefix 'A' denotes 'array', C1 stands for the current component (or the first component of a tensor), while Q represents the local mesh point at which B is being evaluated.

Modularity has been achieved because the same statement (11) will work just as well for spherical polars as for a Cartesian system, if one simply 'plugs in' or 'switches on' a different definition of the CURL operator. The definition of CURL in Cartesian co-ordinates is

```
real procedure CURL(A); real A;
    CURL := RP(DEL(RP(A)))-RM(DEL(RM(A)));                    (12)
```

which gives some idea of the conciseness of Symbolic ALGOL I as well as of its similarity to the notation of theoretical physics. Here RP, RM are mutually inverse rotation operators, permuting the vector components (123) in the positive (231) and negative (312) directions respectively, while DEL is a vector finite difference operator.

Symbolic ALGOL I executes quite slowly because of a large number of nested procedure calls. To get around this problem, we have shown that statements such as (11) can be converted either automatically or by hand into an equivalent form called Symbolic ALGOL II, which when executed will automatically generate an optimized program in any desired output language. For this purpose they are plugged in to an ALGOL program called the Petravić Generator which is supplied with modules analogous to (12) in order to define the difference scheme, co-ordinate system, target language and so on, which are required for the particular job.

The target code is about as fast as well-written FORTRAN, and an added advantage is likely to be that code can equally well be produced for computers for which no compiler is yet available, or even for which FORTRAN is not particularly suitable, such as the new CDC STAR-100 which is able to process complete vectors in one operation without using a DO loop.

What we are doing here is to use the computer itself to write the program, instead of writing it by hand. Since much of the work is tedious and mechanical, this is a very natural development, but the interesting point is that the instructions which must be fed to the computer to make it carry out this task are in virtual one-to-one correspondence with the original mathematical statement of the problem. This is a situation which is reminiscent of both quantization and second quantization, in which the equations always seem to remain the same but get interpreted in different ways. If it can be exploited further, we may be able to use much of the formalism of mathematical physics itself as the algorithmic language for programming computers.

In this sense I believe that one of our immediate aims should be to weld mathematical and computational physics into a coherent whole.

BIBLIOGRAPHY

BALL, W.W.R. (1908). A Short Account of the History of Mathematics, reprinted, Dover Publications, New York (1960).

BARBE, P. (1970), Software Engineering (TOU, J.T., Ed.), 2 Vols, Academic Press, New York and London, Vol. 1, p. 151.

BEMER, R.W. (1970), ibid., p. 121.

BETCHER, R., CRIMINALE, W.O. (1967), Stability of Parallel Flows, Academic Press, New York and London.

CLEMENTI, E. (1970), in Computers and their Role in the Physical Sciences* (FERNBACH, S., TAUB, A.H., Eds), Gordon and Breach, New York, p. 437.

* Referred to as CRPS.

DAVID, E.E., Jr. (1969), Software Engineering, Report on a Conference sponsored by the Nato Science Committee, Garmisch, October 1968, p.62.

ELGOT, G., ROBINSON, A. (1964), J. Ass. comput. Mach. 11, 365.

EMMONS, H.W. (1970), Critique of Numerical Modelling of Fluid-Mechanics Phenomena, Annual Review of Fluid Mechanics, Vol.2 (VAN DYKE, M., VINCENTI, W.G., WEHAUSEN, J.V., Eds), Annual Reviews Inc., Palo Alto, Calif., p.15.

FERNBACH, S., TAUB, A.H., Eds (1970), Computers and their Role in the Physical Sciences, Gordon and Breach, New York.

GOLDSTINE, H.H. (1970), CRPS, p.51.

GOLDSTINE, H.H., NEUMANN, J. von (1963), in John von NEUMANN, Collected Works, Vol.5, Pergamon Press, Oxford.

HARLOW, F.H. (1970), Numerical Methods for Fluid Dynamics, An Annotated Bibliography, Los Alamos Rep. LA-4281.

IBEN, I., Jr. (1970), CRPS, p.595.

KOWARSKI, L. (1970), CRPS, p.479.

KUO-PETRAVIĆ, G., PETRAVIĆ, M., ROBERTS, K.V. (1971), Automatic Optimization of Symbolic Algol Programs, I. General Principles (submitted to J. comput. Phys.).

McCLURE, R.M. (1969), Software Engineering, Report on a Conference sponsored by the Nato Science Committee, Garmisch, October 1968, p.66.

MOCK, O., OLSZTYN, J., STEEL, T., STRONG, J., TEITTER, A., WEGSTEIN, J. (1958), Communs Assoc. comput. Mach.

MORSE, P.M., FESHBACH, H. (1953), Methods of Theoretical Physics, 2 Vols, McGraw Hill, New York.

PASTA, J.R. (1970), CRPS, p.203.

POOLE, P.C., WAITE, W.M. (1970), Software Engng 1, 167.

ROBERTS, K.V. (1969), Comput. Phys. Communs 1, 1.

ROBERTS, K.V., BORIS, J.P. (1971), The Solution of Partial Differential Equations using a Symbolic Style of Algol, J. comput. Phys. 8, 1.

ROBERTS, K.V., PECKOVER, R.S. (1971), Symbolic Programming for Plasma Physicists, Culham Laboratory Preprint CLM-P257.

SNYDER, J.N. (1970), CRPS, p.463.

TURING, A.M. (1937), Proc. Lond. math. Soc. (2) 42, 230.

THE IMPACT OF COMPUTERS ON NUCLEAR SCIENCE

L. KOWARSKI
CERN,
Geneva, Switzerland

Abstract

THE IMPACT OF COMPUTERS ON NUCLEAR SCIENCE.

Nuclear physicists were the first large-scale scientific users of computers. As computer techniques developed in the 1950s and 1960s, they became indispensable in an ever growing variety of tasks of pure and applied nuclear science. The following categories of tasks are considered: (1) Theory and mathematical preparation of experiments; (2) Experimentation at its various stages – before, during and after the collection of data; (3) Simulation and "computer experiments"; (4) Operation and control of nuclear machines; (5) Documentation. This survey is given in a historical perspective with some emphasis on initial difficulties and gradual adaptation. Near-future prospects, especially in high-energy physics, are discussed, leading to the formulation of a conflict between the threat of dehumanization and a humanistic hope.

1. DEFINITION OF A COLLISION

Any kind of physical research, whether pure or applied, is today largely a computer-aided activity, but this is essentially a very recent state of things, hardly half a generation old. Physics, of course, is a much older activity, with its established way of life and thinking. And established ways do not always combine very kindly with new ways. That is why the intrusion of computers into physics was so much of a sudden event, best described in terms of a collision, and that is the meaning of the word "impact" in the title of this paper.

Nowhere has the impact been as dramatic as in nuclear physics and in its spectacular applications, first military and later on industrial. This whole domain of knowledge should be called nuclear science, because it is not only physics. But there the physicists played a dominant role, and also they were the first and the most important users of computers, so in this paper, there is not much difference between talking of nuclear science or of nuclear physics.

To describe the impact and its consequences, I shall proceed in a historical way. I come before you as a witness, as one who was active in nuclear science before the computers came and who was well situated to watch what happened when they did come. And as an observer of this sequence of events I may perhaps allow myself to make a few guesses as to how it is going to develop. Also, as a witness, I may be forgiven for talking at some length about things I know a little better and skipping briefly over those of which I know less.

2. DOMAINS OF PENETRATION

2.1. Introductory survey

Let us start at a definite time-point, say 1950. In the public mind this is the "atomic age", still at the zenith of its glory. We nuclear scientists

are the scientific wizards, ours is the most prominent science. We have heard that "electronic calculators" are being developed somewhere. They may be important for statistics or finance, but not for us, except perhaps for certain unusually intricate things like some particularly messy partial differential equations. Anyhow, radio tubes are clumsy, memories are even clumsier, it all will take time, it does not concern us at present.

Yet, we were drawn in pretty quickly. One simple reason: the computer development took money. It started in Princeton, then it was taken up by Remington Rand (not IBM yet at that time!) but "the atom" was the wealthiest prospective user and so the first user-motivated computers were built in places like Argonne, and in particular in Los Alamos, where Stanislas Ulam and his pupils played a considerable role in these early beginnings.

Occasions for using computers gradually spread all over nuclear science. At first, they arose only in the nuclear scientist's office, at his blackboard, where theories are made and leisurely calculations performed before and after the experiment – not yet at the moment when the scientist is in actual contact with the nuclear phenomenon, not actually in the laboratory or at the nuclear factory. At this stage, the computers are useful for what might in army use be called staff work, not for field work.

Then the computers began to get closer to experimentation and to the laboratory itself. Historically, this happened first in connection with handling the data left after the actual contact with the nuclear phenomenon (data analysis); at a later stage, computers became involved directly in this contact itself, and thus completed the invasion of another domain of nuclear science – the research laboratory.

After the laboratory study of nuclear phenomena comes the application – nuclear machines and factories. This constitutes the third domain. Computers came finally even to the library, "no place to hide", so to speak. After we have gone through this picture of a complete encirclement of nuclear scientists by computers, we shall try to see how the nuclear scientist reacted to this increasing involvement, what made him happy or unhappy and how this, not always easy, relationship may be expected to evolve.

2.2. At the blackboard: mathematics and theories

Let us now look again at a scientist in 1950 at his blackboard, away from the nuclear phenomenon. He may have been using a desk calculator, the cog-wheel type; if the infant electronic computers can provide only the same kind of service, then they are hardly necessary and if they can do something else – but the nuclear scientist sees no need for anything else. This is the usual vicious circle which, at first, is very effective in keeping the atom and the computer apart.

Yet gradually it dawns on our scientist that there may have been cases in his past when an attempt to solve a problem with a desk calculator had to be abandoned because of the sheer volume of work. Electronic computation, being faster, can go further in this direction, and so the realization comes that fast arithmetic can help in solving algebraic equations, differential equations, partial differential equations (a whole new world opens there), matrix problems, complicated functions, etc., etc. The computer then becomes the indispensable tool in a spreading diversity of uses.

Through the arithmetic of Cartesian co-ordinates, the way is open to the use of computers for handling geometry – first through the printing plotters and more recently on the cathode-ray displays. A physicist could hardly appreciate this roundabout way as long as he had to stay away from the computer operation. But once it became possible to put the computer on-line to the physicist's mental approaches to the problem, computerized geometry was recognized as a valuable technique, known nowadays under the name of "interactive graphics".

We have just had the first glimpse of the on-line concept – a computer timed to operate in step with another operation outside itself, in this case a human thinking process. We shall return to this many times and in various contexts. The first problem here is that of mismatched speeds: a computer working as sluggishly as the human mind is a computer largely wasted, and only when computers became cheaper per unit of operation, this waste became economically possible; that is why this particular on-line use is such a recent development.

I mention only for the sake of completeness such extensions of the computer use as formal logic, non-numerical algebra, games theory, artificial intelligence, etc. They have so far had rather little relation to anything nuclear, except perhaps the recognition of visual patterns, on which we shall comment below, and some very recent applications of symbolic algebra, reported in paper SMR-9/27 in these Proceedings. In particular, the symbolic treatment of Feynman diagrams is relevant to all of high-energy physics.

All this can be seen as generalized mathematics and to all this can be applied the remark made some ten years ago by J. T. Schwartz of the New York university: "Mathematics has always sought to reduce the unlimited natural complexity of facts and ideas to a humanly manageable size; it is like mining of diamonds from the surrounding rocks. But the increasingly cheap power of machines enables us to manage far greater masses of irreducible complexity; it is still an extraction process, but it is more comparable to the other useful form of carbon – the mining of coal". This analogy illustrates the difference between the spirit of mathematics and that of computer science and helps us to realize that being a computational physicist, or a computational nuclear chemist, or what not, is not at all the same thing as being a mathematical physicist and so on, so that, in fact, a new way of life in nuclear science has been opened. So much for the impact at the blackboard.

2.3. Computers in the laboratory: before the run

Now comes the laboratory, the contact with the nuclear phenomena. I would like to mention here that our forefathers seem to have been more careful about describing our various ways of dealing with the phenomena. When a physicist measures, using a known technique, some third decimal in the lifetime of a nucleus or a particle, which otherwise is well familiar and well behaved, he hardly should be called an experimenter. And when a junior physicist looks at a bubble film from a chamber built years ago and from a run recording the overall effects of a given beam on a given target, he does not experiment, he observes. Yet, all this is now called experimentation, so we shall have to use that word a little loosely.

In any nuclear "experiment", the crucial moment is that when the animal is actually being killed, I mean when the nucleus or the photon or whatnot is actually striking the detector. That is the phase of data collection; before it comes the preparation, the setting-up of the apparatus, and after it comes the handling of data and the arrival at meaningful scientific conclusions. Computers are now essential to all of these three stages — before, during and after, which I have once called Class 1, 2 and 3 of computer use in experimentation. My next remarks apply chiefly to a vast variety of nuclear experiments using the electronic kind of detection which is now current in high-energy physics as well as in low-energy physics, nuclear chemistry, etc., and — as you will see — a closer look at these applications will lead us to some slight changes in the definition of these three classes.

The setting-up of an experiment really starts with some thinking at the blackboard (we have already dealt with that) and, nowadays, increasingly often with preliminary computations which try to simulate the future experimental situation — we shall return to that below. After this more or less theoretical preamble, the experimenter may need the computer for such Class-1 work as checking the apparatus, testing its performance on blank runs and trial runs, adjusting the electronic logic, etc.

Let me quote here two fairly recent opinions from eminent physicists on the relevance of Class-1 computer use. An opinion from M.I.T.: "If more than 30-40 counters are used in the same experiment, then a computer is necessary in the setting-up stage". Another, from Princeton: "We try to choose our experiments so that the apparatus is simple enough to be set without using a computer. If it is not, then there are too many interfaces, the flexibility is lost". The prevailing trend seems to be away from this happily austere ideal.

A computer in Class 1 must be fast enough to follow the fast working of the apparatus in "real time". This is another kind of on-line use — on-line to the apparatus. It may require a rather large computer, e.g. a PDP-6 or -10, but in most cases a smaller size (such as PDP-9) seems to be adequate; still smaller computers, like the PDP-8, are hardly useful anymore in this connection. The same not too big computer may be useful when the experiment is actually going on, for the purpose of collecting the incoming data and to put them on a single record such as a magnetic tape. It is convenient to consider this particular function as belonging still to the Class 1, although it already takes place during the actual kill.

2.4. During the run

Class 2 is the most modern and the hottest problem, because it raises all sorts of issues about the required minimum size of the computer, its location and the wasteful use of its available time. The kill, usually called the run, may sometimes go on for days or weeks. The scientist wants to know if his incoming results are worthwhile; therefore he has to extract the meaning from the data as they arrive, if not all of them, at least from samples. This extraction very often requires the active intervention of the scientist's mental processes, if any; therefore the computer must be available on-line both to the nanoseconds timescale of the apparatus and to the seconds timescale of the thinker. Also, full processing up to the

physics-meaningful conclusion often requires definitely a large computer – a 6400, a 360/65 or more – to be made available to every interested user right on the site where his experiment is actually taking place. Class 2 is essential: according to its results the physicist may have to modify the experimental run or even to stop it altogether.

It is amusing to see how modest were the experimentalist's Class-2 requirements only a few years ago. A leading Columbia low-energy physicist, Dr. Lidofsky, reported in 1964 that the currently used system involved recording the output of his multichannel analyser on paper tape, then sending the tape by messenger to a distant small computer which would produce magnetic tape, then using another messenger to bring that tape to a 7090 computer. The elimination of the first messenger – that is, putting a PDP-4 on-line to the analyser – was at that time in progress. The possibility of using a large computer on-line, via data links, was mentioned as a utopian ideal. To-day the tendency is to build in a not too small computer, such as a PDP-9 (the same as in Class 1), and if this is not enough for Class 2, to have a practically on-line access, via data links and time-sharing, to a really big computer. How necessary this is going to be will be seen from two illustrations: according to P. Egelstaff (Harwell) a typical analyser of 1956 had 100 channels. By 1960 he got to a few thousand. Much more recently, a report on Dr. Ghiorso's radiochemical investigations in Berkeley mentioned an analysing set-up involving three variables (the chemical nature of the sample, time intervals, the height of the pulse) and a total of several hundred thousand channels.

2.5. After the run

And now, the run is over and we are left with the bulk of the recorded data, already partly processed through the Class-1 and -2 computers, already sampled as to their physical meaning. What comes now is Class 3. For a large category of experiments, mostly in high-energy physics (all of bubble chamber experiments and several kinds of those with spark chambers), all computer use is Class 3, because the results of the detecting run are at first recorded visually on film without any recourse to a computer. But then visual data have to be processed through film-measuring and pattern-recognizing stages before they are reduced to a set of computable numerical data. On the whole, it can be said that the use of a film-measuring machine re-introduces all the headaches which have been avoided during the initial detection, that is the film-taking stage. Now, to handle the film information, we face again the need for a smallish computer on-line to the measuring machine, a bigger computer on-line to a monitoring physicist, etc. There is also the problem of insertion of human help to overcome the occasional weaknesses of the information processing by machine.

The final domain of data handling, or the Class 3 proper, consists in the reduction of the raw computable data to their meaningful scientific conclusion. Dr. Lidofsky (Columbia university) quoted, as a typical example of the 1964 era, an experiment on gamma rays from some excited levels of a nucleus; there the computable data amounted to over two million bits and the final conclusion was expressed in about 100 bits. In 1970, Dr. Macleod (CERN) mentioned in a lecture cases of nearly 10 000 recorded bits per second of effective run. We are progressing, aren't we!

The last reduction stage in bubble experiments is the SUMX program, about which Lohrmann reports in paper SMR-9/28 in these Proceedings. The closer you get to the intelligent conclusion, the more important is the active participation of the scientist; all the problems of interactive operation, properly planned time-sharing, on-line access to a big computer and so on, have to be tackled as an increasingly accepted part of a physicist's life.

About ten years ago this was not foreseen at all. Data collection and reduction, recognition of visual patterns, etc., were thought to be amenable to a complete automation; the scientist would only come in to muse over the final print-out.

Systems of bubble film analysis invented around 1960, such as HPD and PEPR, were inspired by this "ideal" but did not succeed in achieving it fully. In the much simpler task of analysing spark-chamber film, the elimination of human intervention has been pushed farther, but even there not to the very end. Since the mid-1960s the style of using computers in nuclear experiments began to shift towards a proper combination of the human operator with the machine, instead of trying to do away with the human altogether; older bubble-film processing devices began to be exploited in a systematically interactive mode, and newer devices (e. g. those known as Polly and Sweepnik) explicitly embodied the principle of man-machine symbiosis.

2.6. Simulation

We are now going to take the scientist out of his laboratory and let him proceed further, but before that we must return him once more to the blackboard. Having got used to the computer in connection with the incoming physical data, he noticed that he can learn a lot about the phenomenon without taking any data at all. Instead of using experimental data as input, he can use an inflow of random numbers – and let the computer process them according to all the rules he already knows to be valid for that particular category of physical phenomena. This kind of study is known as simulation. Strictly speaking, it is not a branch of experimental science at all, and yet it looks very much like one, and one often hears nowadays of "computer experiments". In its purest form computer simulation studies problems which are fully known in their basic principles but too complex to allow the derivation of concrete numerical results from such principles – a situation which often occurs when dealing with many-body problems, liquids, plasmas. Then there is the mixed form which takes in actually observed data and then uses them as input for computer simulation of their subsequent evolution; meteorology and astrophysics, in particular, make use of this method. "Computer experiments" are mentioned in the papers by Hockney and by Roberts in these Proceedings.

Simulation is also useful in Class-1 work as a means of studying the performance of instruments from an input of random numbers rather than the physical data. Design and trial of electronics; testing of photographic optics and their distortions in the new, very complex bubble chambers; testing of bubble processing programs by trying them on simulated track co-ordinates – all these are suitable domains for simulation by computer.

2.7. Computers in nuclear factories and machines

After this excursion back to the blackboard, we now can turn to the scientist who has proceeded from the laboratory to the application, where his scientific work is concerned with actually running a definite device or machine. In the history of introduction of computers into nuclear sciences and their applications, one of the first domains considered was the control of nuclear reactors. Reactors are fairly quiet and steady devices, but if not enough attention is given to the evolution of radioactivity or of neutron fluxes within them, they may tend to get astray from the equilibrium position chosen for their functioning. Therefore, the operator in charge of the reactor has at his disposal various sensor devices which show the neutronic state inside the reactor. According to these indications, he takes his decisions and intervenes, if necessary, to change the working conditions of the reactor, so as to bring the chain reaction to a desired level of intensity.

All this, in principle, could be done by a computer. The sensors could transmit their readings to the computer which would digest the data, arrive at a logical decision and intervene according to this decision. This could be done with a speed and an accuracy far greater than those obtainable from a human operator, which would be useful for ensuring a safety feature that may yet prove to be very valuable for some reactor types in some critical stages of their functioning.

These prospects have been aired in many theoretical discussions; I was involved in some of them back in 1963. Five years later, at a conference held in Norway, I could see that the discussions had remained largely theoretical and not much use was made of them in practice. Some beginnings in this respect were reported from Canada and Japan; I do not know how much has been done since then.

Another field of nuclear technology where computers can make effective decisions, is the running of particle accelerators which is of direct interest to us in high-energy physics. In the experimental equipment around the accelerator there are many more tasks for controlling computers; they are now used extensively for handling the primary and secondary beams and, increasingly, in the functioning of very big bubble chambers or electronic experimental set-ups (the latter also in low-energy nuclear experiments). All these computers act as a switch-board mechanism which becomes necessary when the logic of switching is enormously complex and/or has often to be changed.

2.8. Computerized documentation

As the last domain in which computers have invaded the nuclear scientist's life, we have to mention the library, documentation and scientific information in a larger sense. The first way of using computers in these activities was the computerized production of catalogues. Instead of inscribing catalogue entries in time-honoured folios or card files, they can be put on computer-readable tape. The computer can then perform certain standard operations suitable for arranging the catalogue data in a desired form, and print out the results so that the printed catalogue can be made available in as many copies as required and not just one card file in the

central library. If the catalogue entry contains not only the classical indications such as the authors' names, the title of the document, the year of its publication and so on, but also a few keywords or other "descriptors" to give an idea of what the subject matter is about, then the computer can be used to make searches through large sets of entries and to print out a list of documents in a given collection which, by their content, correspond to some formulated demand. For a precise characterization of a document by keywords, help from a human documentalist has so far been necessary. Schemes have been proposed for the computerized extraction of keywords from a given text by making it optically accessible to the computer, which would single out the keywords on the basis of their frequency in the document or of their appearance in some specified context. Many theoretical studies and experiments of this sort have been performed, but the practical results have, so far, been rather meagre; as in bubble-chamber analysis, the "ideal" of eliminating all human intervention may soon be replaced by the deliberate development of an optimum combination of man and machine.

3. RESPONSE TO THE INVASION

I have said enough about what tasks the nuclear scientists are now considering as legitimate fields for using computers, and how the growing importance of computers and of their on-line interaction with scientific thinking has induced the nuclear scientist – has forced him – to become at least a computer technician if not always a computer scientist. In terms of a well-worn piece of political wit, those of the atomic era have been dragged into the computer era screaming and kicking. After the complete aloofness of 1950 they gradually came to view the computer as an occasionally useful blackbox, almost impossible to understand and not much worth the effort of understanding, accessible only through an esoteric language – there was no practical FORTRAN in those early years, because computer time was too precious to be wasted in such crude ways. Only professional applied mathematicians could approach the blackbox and with these, for the next ten years or so, the nuclear scientist had to co-operate willy-nilly, most often nilly.

The invasion of the laboratory was probably started around 1955 by Luis Alvarez and his group, with computerized measuring machines for bubble-chamber pictures. A new kind of interface had to be designed, placed between the film-recorded data and the computer; soon afterwards, another such interface appeared in low-energy physics between the pulse analyser and the computer. The nuclear scientist tended to design and build these intermediate devices all by himself, helped only by his faithful, modest electronic technicians. They sometimes would build a whole custom-built specialized computer, without even realizing it. It was the time of what I used to call tricky hardware – very ingenious, very rigid. Equivalent tricks by software were already thinkable, but nuclear men were shy of them.

Not for long. Nuclear physicists began to learn FORTRAN. Since physics is defined as what physicists do (it is quite official, not just my joke), computers did duly become a part of physics. It seems almost incredible today that only eight years ago, on the highest directing level

of CERN, I was told (in full view of an approaching crisis of CERN computing facilities) that a "crash effort" could be approved only in physics but not in computers. Today such a deliberate opposition would sound rather unnatural.

As scientists get used not only to writing their own programs, but also to sitting on-line to an operating computer, as the new kind of nuclear scientist develops — neither a theoretician, nor a data-taker, but a data-processor specialized in using computers — they become too impatient to sit and wait while their job is being attended to by computer managers and operators. Turn-around time vexations can ultimately be solved only in two ways: one is to provide each experimental group with its own computer, available for on-line inclusion into its experimental set-up or for its processing operations; and the other is time-sharing of bigger computers, with remote consoles. Both of these solutions have been adopted in the current practice.

4. PROSPECTS AND VALUES

4.1. Liberation in space

Storage of raw data for subsequent processing begins to be noticeable as a means of resolving some contradictions imposed by geographical separation between the accelerator and its average user. A university physicist can then perform a very significant part of his experimental work without leaving his own campus, especially if data transmission by telephonic links is available to supplement the conveyance of data by mail or messenger. Perhaps, when links as comprehensive as those used in television become available at long distance, there will be even less reason for the user to spend a lot of his time on the site where his physical events are being produced. This may even abolish the kind of snobbery which decrees today that only those may be considered as physicists who are bodily present at the kill, that is at the place and time when the particle is actually coming out of the accelerator and hitting the detector.

Today the campus user tends to use his own computer only in Class 1 and 3 of his activities, in terms of the classification explained above, and to commute for the Class 2 of his processing work. As the remaining dependence on site facilities diminishes, this almost complete liberation from the geographical dilemma can acquire a direct political significance. We can envisage a state of things when there will be only one biggest machine in the whole world available for some problems in high-energy physics or in high-flux lower-energy nuclear research. It will be unthinkable to concentrate all the leading experimentalists of the world around this machine: decentralization of experimentation is necessary in order to keep alive the teaching by prominent scientists.

4.2. Liberation in time

In addition to this increasing liberation in space, computers can already give freedom from several kinds of time limitations. We are no longer obliged to conform ourselves to the nanosecond region of speeds in which the events are actually happening: we can record them in their real time,

and then process and study the records in our own timescale. The other form of time liberation is the possibility of reprocessing an event in some new ways long after its original handling. The events can be stored in archives, where we can return and browse, and now and then, perhaps, extract some new meaning from an old record.

The third way is the liberation from the tyranny of the Big Machine's time schedule. Because the physicist is no longer compelled to be present when the physical event is actually happening, he can organize his work in a better equilibrium with his teaching duties.

4.3. Dangers and hopes

The vision of these huge and costly machines, spitting their particles into almost as huge and no less costly assemblies of detecting apparatus, every day of the year and every millisecond of the day — the yield has to justify the investment, you know! — is in a way terrifying. The era of the ingenious scientist, who sets up his apparatus to catch Nature behaving this or that way, seems to be past. The machine will have to run just "because it is there", and according to its own rules. And from each run — there will be not much sense in calling them experiments any more — there will be a rich harvest of recorded data, like a deep-sea dredge coming up with its load of pebbles and fishes. A nuclear scientist will prepare the run and sift the harvest, following the example of today's oceanographers, selenologists and archeologists.

There will be a lot of attempts to judge such new situations by old value criteria. What is a physicist? What is an experimenter? Is simulation an experiment? Is the man who accumulates print-outs of solved equations a mathematical physicist? And the ultimate worry: are we not going to use computers as a substitute for thinking? There is an interesting recent precedent for this increased reliance on computers. In the early 1950s hydrogen bombs were developed simultaneously in the Soviet Union and in the United States of America, and a little later in the United Kingdom. They were successfully brought to the point where they could really work, I mean explode. The French scientists, attacking the same problem in the 1960s, felt that they needed access to the biggest scientific computers available at that time and which certainly were not available in the 1950s to their American, Soviet or British colleagues. The United States government embargoed some of the biggest American computers when it learned that they were to be used in a French hydrogen-bomb project. The French scientists protested that this made their work as good as impossible. We are led to believe that with their methods as they had developed by that time, this work did indeed become impossible in the absence of very big computers, and to take this episode as a striking illustration of the irreversible evolution of working habits in nuclear science, due to the advent of computers. It is, of course, only fair to assume that the standards of desirable accuracy had gone up in the intervening decade so as to ensure more completely reliable predictions of performance than those which were available in the early 1950s.

In high-energy physics itself it is noticeable that bubble-chamber technique has got further towards what we might call too much dependence on computers than its other branches of experimentation. The contemplation

of this and other such trends leads us to the formulation of a dehumanized utopia in which nuclear scientists will produce very fine measurements and will make no discoveries. In some sense we may be obliged to say that this situation is already largely prevailing in high-energy physics, especially in Europe. But, of course, the full dehumanization is only an extrapolation limit. Moreover, we must never forget that the computers are also available for constructing a humanistic utopia: they offer us the possibility of browsing over stored data with all intact human ingenuity to give a new sense to the already processed events; the overall direction of experimentation by a creative human; the fruitful symbiosis between man and machine. William Miller formulated in 1964 an ideal: "We must decide what to do and the machine will maybe tell us how to do it". I might add: how to do it by operating the machine. But as long as we decide, and not the machine, there is hope.

PART II: CLASSICAL COMPUTATIONAL PHYSICS

OCCURRENCE OF PARTIAL DIFFERENTIAL EQUATIONS IN PHYSICS AND THE MATHEMATICAL NATURE OF THE EQUATIONS

D.E. POTTER
Department of Plasma Physics,
Imperial College,
London, United Kingdom

Abstract

OCCURRENCE OF PARTIAL DIFFERENTIAL EQUATIONS IN PHYSICS AND THE MATHEMATICAL NATURE OF THE EQUATIONS.

This paper is introductory in discussing how partial differential equations arise in physics and in reviewing some simple properties which are nevertheless important when considering methods of solution on the computer. The importance of principles of conservation in physics, and in particular in classical physics, is stressed and they are used to derive some important examples of partial differential equations. The use of Fourier analysis and the derivation of dispersion relations is reviewed for waves, advection, diffusion and potential problems. The time-scales of interest in initial-value problems are isolated, as a preamble to the concepts of stability and accuracy in difference solutions. The essential non-linearity of the few- and many-body problem is discussed. The general initial-value problem is defined and the concepts of a discrete mesh in space and time are introduced. A comparison is made between a continuous function and a mesh vector by Fourier analysis, illustrating that the essential approximation on a difference mesh is the limitation to long-wavelength modes. Integration of the initial-value problem in time over finite time-steps is introduced.

1. INTRODUCTION

The power of differential calculus has led to the very wide application of the concepts of continuous media and continuous fields. In using such concepts, continuous functions in space and time are defined which describe the properties of the medium, and on applying the quantitative principle of physics, partial differential equations are obtained which couple the properties of the medium in space and time.

To illustrate the very wide application of this concept, and consequently the very wide occurrence of partial differential equations, a few fields which rely on such a concept are listed: in classical electrodynamics, Maxwell's equations are formulated by defining continuous electromagnetic fields and continuous source functions; solids are frequently treated for simplicity as a continuum, though we know otherwise; a great variety of fluids (liquids, gases, plasmas, the galactic fluid) may be treated with most ease as a continuum; other examples are phase fluids and "continuous fields of force" in classical and quantum mechanics.

Although it is clear that the partial differential equations of physics arise in a great variety of ways, and from very different problems, nevertheless such equations and systems of equations repeatedly take the same form or similar ones. This is essentially because a great deal of the philosophy of physics and in particular classical physics has been formulated in

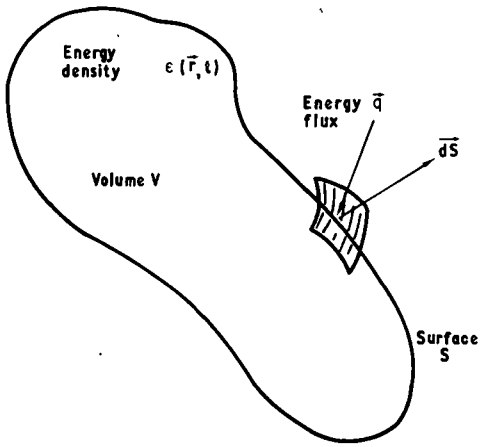


FIG. 1. Conservation of energy.
The heat conduction (diffusion) equation is obtained by applying the principle of the conservation of energy to an arbitrary volume, V. The energy in the volume V may only alter by a flux of energy $\vec{q}$ across the surface S of V.

terms of principles of conservation. Some simple but vital examples come readily to mind: mass cannot be created or destroyed; momentum is conserved; total electric charge is an invariant; etc. To stress this point and to develop some systems of partial differential equations with which we will be concerned, some particular examples may be considered [1].

1.1. Conservation of energy in a solid

Classically, energy is transported in a solid by conduction. Since the solid is a rigid stationary body, the variable energy density (ϵ) in the solid is given by the thermal energy or temperature. Hence the principle of conservation of energy is invoked and must be satisfied when considering the energy in a finite volume V of the solid of surface S (Fig. 1). By the principle of the conservation of energy, the rate of change of energy in V must be equal to the flux of energy $\vec{q}$ across the surface S of V:

$$\text{Energy in volume V} = \iiint \epsilon(\vec{r},t)\,d\tau$$

$$\text{Flux across S} = -\oiint_S \vec{q}\cdot d\vec{S}$$

$$\frac{\partial}{\partial t}\iiint_V \epsilon(\vec{r},t)\,d\tau + \oiint_S \vec{q}\cdot d\vec{S} = 0$$

Applying the divergence theorem to the second term and for a constant volume V,

$$\iiint_V \frac{\partial \epsilon}{\partial t} d\tau + \iiint_V \nabla \cdot \vec{q}\, d\tau = 0$$

The energy density is proportional to the temperature T, and experimentally it is found that the heat flux $\vec{q}$ depends on the gradient of the temperature. Hence, defining a proportionality constant, the conductivity K,

$$\frac{\partial T}{\partial t} - \nabla \cdot K \nabla T = 0$$

The essential principle described by the diffusion equation in this case is then the principle of the conservation of energy.

1.2. Conservation of electric charge

Another informative example, in electromagnetic theory, is the principle of the conservation of electric charge. Again, therefore, in a volume V of surface S, the change rate of charge in V must be equal to the flux of charge (current $\vec{j}$) across the surface. If ρ is the charge density $\rho(\vec{r}, t)$ then,

$$\frac{\partial}{\partial t} \iiint_V \rho d\tau = - \oiint_S \vec{j} \cdot d\vec{S} \qquad (1)$$

and applying the divergence theorem,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{j} = 0 \qquad (2)$$

This is of course just one of Maxwell's equations, since using Gauss' law ($\nabla \cdot \vec{D} = \rho$, where D is the electric field),

$$\nabla \cdot \left(\frac{\partial \vec{D}}{\partial t} + \vec{j} \right) = 0$$

and on integrating and noting that $\nabla \cdot (\text{curl}\, \vec{X}) = 0$,

$$\frac{\partial \vec{D}}{\partial t} + \vec{j} = \nabla \times \vec{X} \qquad (3)$$

where $\vec{X} = \vec{H}$ is the magnetic field by Ampere's law.

1.3. Conservation of magnetic flux

Similarly, Faraday's law is an expression of the conservation of magnetic flux: magnetic flux cannot be created or destroyed, since the change rate of the total flux through a surface S is related only to the

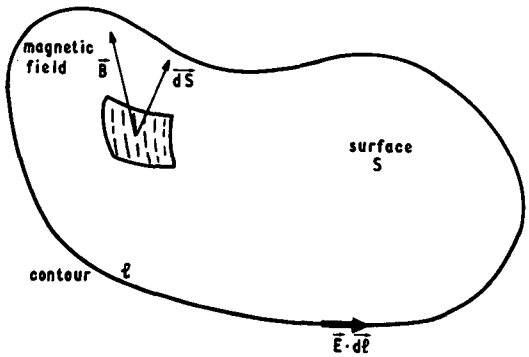


FIG. 2. Conservation of magnetic flux.
Faraday's law is a statement of the conservation of magnetic flux.

electric field around the boundary ℓ of S (Fig. 2),

$$\frac{\partial}{\partial t} \iint_S \vec{B} \cdot d\vec{S} = - \oint_\ell \vec{E} \cdot d\vec{\ell}$$

and applying Stoke's theorem,

$$\frac{\partial \vec{B}}{\partial t} + \nabla \times \vec{E} = 0 \tag{4}$$

1.4. Conservation of mass and momentum in a fluid

In a fluid, the basic classical principles of the conservation of mass, the conservation of momentum (Newton's third law) and the conservation of energy are used to derive equations to describe the dynamics of a fluid. Defining a variable $\rho(\vec{r}, t)$, the density of the fluid, we invoke the first principle to state that the change rate of mass in the volume V must equal the mass flux crossing the surface S of V (Fig. 3). The mass flux through any surface element $d\vec{S}$ is just $\rho\vec{v} \cdot d\vec{S}$. Hence,

$$\frac{\partial}{\partial t} \iiint_V \rho d\tau = - \oiint_S \rho\vec{v} \cdot d\vec{S}$$

and using the divergence theorem, a differential equation for the conservation of mass is obtained,

$$\frac{\partial}{\partial t} \rho + \nabla \cdot \rho\vec{v} = 0 \tag{5}$$

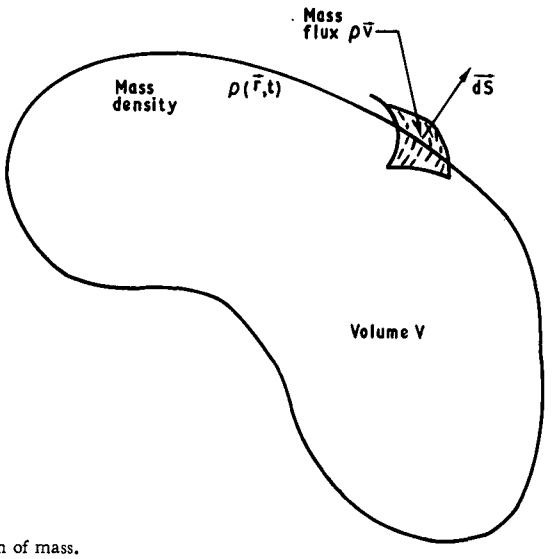


FIG. 3. Conservation of mass.
The continuity equation in hydrodynamics is obtained by applying the principle of the conservation of mass. The total mass in the volume V may only be altered by the existence of a net mass flux across the surface S of V.

Similarly, by demanding that momentum be conserved, an equation of motion in the fluid is obtained. Consider the conservation of momentum in the X-direction (Fig. 4). The total X-momentum in the volume V is

$$\iiint_V \rho v_x \, d\tau$$

The X-component of momentum of the fluid in the volume V is increased in time, by momentum being convected across the surface S,

$$- \oiint_S \rho v_x \, \vec{v} \cdot d\vec{S}$$

and by the sum of the pressures (p) in the X-direction on the surface S,

$$- \oiint_S p\hat{\vec{e}}_x \cdot d\vec{S}$$

Hence conservation of X-momentum yields the equation

$$\frac{\partial}{\partial t} \iiint_V \rho v_x \, d\tau = - \oiint_S (\rho v_x \vec{v} + p\hat{\vec{e}}_x) \cdot d\vec{S}$$

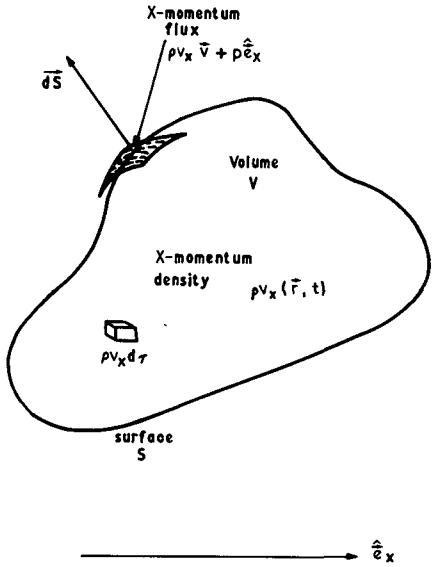


FIG. 4. Conservation of momentum.
The vector momentum equations in hydrodynamics are obtained by applying the classical principle of the conservation of momentum. Both the thermal pressure and the centre-of-mass kinetic momentum contribute to the flux across the surface.

By the divergence theorem,

$$\frac{\partial}{\partial t} \iiint_V \rho v_x \, d\tau = - \iint_V \nabla \cdot (\rho v_x \vec{v} + p\hat{\vec{e}}_x) \, d\tau$$

therefore,

$$\frac{\partial}{\partial t} \rho v_x + \nabla \cdot (\rho v_x \vec{v} + p\hat{\vec{e}}_x) = 0 \tag{6}$$

Similarly, the equations of motion are obtained for the y and z directions. We summarize these three equations as

$$\frac{\partial}{\partial t} \rho \vec{v} + \nabla \cdot (\rho \vec{v} \vec{v} + p \vec{\vec{I}}) = 0 \tag{7}$$

where $\vec{\vec{I}}$ is the unit tensor. These equations are the hydrodynamic equations describing the motion of a compressible fluid.

To summarize, I have illustrated how partial differential non-linear equations governing a variety of physical systems are obtained from simple

fundamental principles of conservation. The differential equations are "conservative" and they take the general form

$$\frac{\partial \vec{u}}{\partial t} + \nabla \cdot \vec{\vec{f}} = 0$$

where $\vec{u}$ is a vector of the dependent variables and $\vec{\vec{f}} = \vec{\vec{f}}(\vec{u})$.

2. FOURIER ANALYSIS AND THE DISPERSION RELATION

Partial differential equations couple points in space and time and many of the essential and simple properties of a partial differential equation or a system of partial differential equations can be described by the behaviour of a wave in space and time. When considering such simple or general properties of an equation it is useful to assume periodic boundary conditions and to apply the techniques of Fourier analysis. If $u(x,t)$ is periodic over X, then, provided u satisfies simple conditions (Dirichelet conditions) (Ref. [2], Vol. 1), u can be expanded as an infinite Fourier series:

$$u(x,t) = \sum_{k=0}^{\infty} \hat{u}_k(t) \exp\left(\frac{i2\pi kx}{X}\right) \tag{8}$$

where

$$\hat{u}_k(t) = \frac{1}{X} \int_X u(x,t) \exp\left(\frac{-i2\pi kx}{X}\right) dx \tag{9}$$

It is stressed that for a continuous medium the function u must be described by an infinite set of modes.

Let us consider four types of processes which may be described by rather simple partial differential equations but which in more complex forms keep recurring in interesting physical problems. In particular, we shall be interested in the time-scales τ of these processes.

2.1. Waves and the wave equation

The phenomenon of waves and wave motions occurs so frequently that it is not necessary to enumerate some examples. Consider the particular case of a wave on a stretched string, where the displacement $\xi(x,t)$ of the string is described by a wave equation

$$\frac{\partial^2 \xi}{\partial t^2} - V_s^2 \frac{\partial^2 \xi}{\partial x^2} = 0$$

The parameter V_s is in this case given by the tension T in the string and the mass per unit length m of the string, $V_s = \sqrt{T/m}$. If L is a characteristic

length along the string, then we can define a characteristic time τ as the time for a travelling wave to propagate over the length L,

$$\tau \sim \frac{L}{V_s}$$

In a more sophisticated way, we consider a Fourier mode on the string

$$\xi(x,t) = \hat{\xi} \exp(i(\omega t - kx)) \tag{10}$$

Inserting the mode in the wave equation, for a given wave number k, ω must satisfy

$$-\omega^2 + k^2 V_s^2 = 0 \tag{11}$$

where ω is the angular frequency associated with the wave, k is the wave number, $k = 2\pi/\lambda$, and λ is the wavelength. Thus a characteristic time-scale τ may be associated with the wave

$$\tau = \frac{2\pi}{\omega} = \frac{2\pi}{V_s k} = \frac{\lambda}{V_s}$$

Equation (11) is the dispersion relation of the partial differential equation. In obtaining difference solutions to such equations we will be interested in the characteristic times which may be associated with the physical process. It is also noteworthy that, if a velocity $v = \partial\xi/\partial t$ of the displacement and the angular displacement $\theta = \partial\xi/\partial x$ are defined, the second-order wave equation may be written as two coupled first-order equations

$$\frac{\partial v}{\partial t} - V_s \frac{\partial \theta}{\partial x} = 0 \tag{12}$$

$$\frac{\partial \theta}{\partial t} - V_s \frac{\partial v}{\partial x} = 0 \tag{13}$$

2.2. Advective equation

The advective equation is related to the wave equation and arises when properties of a fluid are advected (or convected) by the fluid. The term has already come up in the equations of hydrodynamics (Eqs 5, 7). The conservation of fluid mass may be written as

$$\frac{\partial \rho}{\partial t} + \vec{v} \cdot \nabla \rho + \rho \nabla \cdot \vec{v} = 0$$

$$\frac{d\rho}{dt} + \rho \nabla \cdot \vec{v} = 0$$

where $d/dt = \partial/\partial t + \vec{v}\cdot\nabla$ is the total time derivative or Lagrangian derivative. The property of fluid density is a local property of a fluid element, and as the element moves in the fluid, its density is advected with it. In the incompressible case but with a variable-density fluid, the conservation-of-mass equation is

$$\frac{d\rho}{dt} = \frac{\partial \rho}{\partial t} + \vec{v}\cdot\nabla\rho = 0 \qquad (14)$$

the advective equation. It is clear that when obtaining equations for any intensive property of any fluid in the initial-value problem, the advective terms will arise in the equation describing that property.

Again it is important to assign a time-scale to the process of advection: obviously the time-scale of interest in this case is simply the time for a point in the fluid to move over the characteristic distance L,

$$\tau = \frac{L}{|\vec{v}|}$$

$\vec{v}$ is now the centre-of-mass velocity and not a phase velocity. More precisely, we obtain the dispersion relation for the advective equation for a Fourier mode

$$\rho = \hat{\rho} \exp(i(\omega t - kx))$$

Consequently,

$$i(\omega - \vec{k}\cdot\vec{v}) = 0 \qquad (15)$$

$$\tau = \frac{2\pi}{\omega} = \frac{2\pi}{\vec{v}\cdot\vec{k}} = \frac{\lambda}{|\vec{v}|}$$

2.3. Diffusion equation

The diffusion equation is very familiar and arises in a multitude of problems in physics. In one dimension and in the simplest case the equation takes the form

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x} K \frac{\partial u}{\partial x} = 0 \qquad (16)$$

where $u(x,t)$ is a dependent variable and K is a diffusion coefficient. In more complex forms the equation can include inhomogeneous or source terms on the right-hand side and it becomes non-linear when the conductivity

or diffusion coefficient is a function of the dependent variable K = K(u). The time-scale of interest here is the diffusion time

$$\tau = \frac{L^2}{K}$$

More specifically, we can consider again the effect of the diffusion equation on a Fourier mode

$$u = \hat{u}_k \exp(i(\omega t - kx))$$

and consequently, for a constant conductivity K,

$$i\omega + Kk^2 = 0$$

$$\omega = iKk^2 \tag{17}$$

The dispersion relation for the simple diffusion equation is obtained. The angular frequency ω is now imaginary and hence the mode decays in time. The time-scale for this decay is

$$\tau = \frac{2\pi}{\omega} = \frac{2\pi}{Kk^2} = \frac{\lambda^2}{2\pi K} \tag{18}$$

2.4. Elliptic equations

Finally, the elliptic equation, arising from boundary value problems, might be included as a fourth example. Again, such equations are common and familiar in physics: Laplace's equation and Poisson's equation are examples,

$$\nabla^2\phi = 0 \tag{19}$$

$$\nabla^2\phi = -\rho \tag{20}$$

The dependent variable ϕ might be an electrostatic or gravitational potential, while the inhomogeneous term or known "source function", ρ, might be a charge or mass density. These equations result from considering static solutions or alternatively in cases where it has been assumed that information is transported instantaneously. If the analogy with the previous three processes of wave propagation, advection and diffusion is maintained, in the case of the elliptic equation the angular frequency of a Fourier mode is effectively infinite, and the time-scale for information to propagate over a scale length L is effectively zero.

$$\omega \to \infty$$

$$\tau \to 0$$

3. FORMAL CLASSIFICATION OF PARTIAL DIFFERENTIAL EQUATIONS

We have considered four types of commonly occurring partial differential equations in physics. In their linear forms we see that three of them are each a special case of the general second-order two-dimensional equation

$$a\frac{\partial^2\phi}{\partial x^2} + b\frac{\partial^2\phi}{\partial x\partial y} + c\frac{\partial^2\phi}{\partial y^2} + d\frac{\partial\phi}{\partial x} + e\frac{\partial\phi}{\partial y} + f\phi + g = 0 \qquad (21)$$

a, b, c, d, e, f, g may be functions of the independent variables x, y and of ϕ (non-linear). We classify formally second-order partial differential equations as:

hyperbolic: when $b^2 - 4ac > 0$

parabolic: when $b^2 - 4ac = 0$

elliptic: when $b^2 - 4ac < 0$

For a detailed discussion of the properties of partial differential equations and analytic methods of solution, see Ref. [2], Vol. 2.

4. NON-LINEAR PHENOMENA

In Sections 1-3, the occurrence of partial differential equations in physics and some simple linear properties of those equations have been surveyed. These properties are familiar since analytic theory is particularly successful in describing linear phenomena. However, apart from the simplest problems, analytic mathematics breaks down in describing non-linear phenomena, while computational physics relies on no such property, and the subject can therefore give us a considerable understanding of such properties. Particularly in the few- and many-body problems the essential property with which we must deal is the non-linear coupling between the force field and the particles. As an example, though a general one, if we describe a classical many-particle distribution by the distribution in phase space $f(\vec{p}, \vec{q}, t)$, then the distribution of particles in phase space might satisfy the time-dependent equation

$$\frac{\partial f}{\partial t} + J(f, H) = 0$$

where J is a Jacobian with respect to $\vec{p}$, $\vec{q}$, and H is the Hamiltonian of the system, which depends on the particular force law between the particles. Since the Hamiltonian will in general be a function of the distribution, the problem is non-linear. As an example of such a problem, we could describe

a galaxy of stars by the distribution of stars in phase space, f, which satisfies the Vlasov equation

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_r f - \vec{\nabla}\phi \cdot \nabla_v f = 0$$

$$\nabla^2 \phi = 4\pi G \vec{m} \int_{-\infty}^{+\infty} d\vec{v}$$

ϕ is the gravitational potential, G the gravitational constant and m the mass of a star. In this problem the distribution defines the potential, while the potential between the stars defines the time-evolution of the distribution and the problem is strongly non-linear. It is this essential coupling between the Hamiltonian and the distribution which ensures that all few- or many-body problems, in whatever particular assembly they arise, are non-linear: examples occur in Vlasov systems in hydrodynamics, in atomic and molecular physics (c.f. Hartree-Fock approach) [3], in meteorology, in oceanography, and in the structure of stars or galaxies.

As a particular example of a non-linear phenomenon, the case of shocks in a compressible gas may be considered.

We frequently analyse small-amplitude disturbances or waves on some equilibrium configuration (for example, sound waves in the atmosphere), and the question arises, "what is the effect when the waves no longer have a small amplitude relative to the equilibrium?". In this instance, long-wavelength modes tend to dissipate their energies to shorter wavelengths. We might ask the question then, "what is the final distribution of energies among the modes?". Such a problem arises in describing shocks or large-amplitude disturbances in compressible hydrodynamics or plasmas, and it would seem that simulation on the computer would provide a valuable approach to the problem.

5. INITIAL-VALUE PROBLEM

Many systems of partial differential equations of interest are time-dependent, and it is useful, before considering methods of solution on the computer, to define the general initial-value problem. This problem occurs in every branch of physics, and, involving as it does the idea of prediction, it is of prime interest in computational physics. It is particularly applicable to solution on the computer, since we can evolve the solution in the real time of the computer.

Given a system defined by the state vector $\vec{u}$, in the space domain R, if $\vec{u}$ is defined at time t = 0 in R, $\vec{u} = \vec{u}^0$, and if $\vec{u}$ is defined on the surface S of R for all t $\vec{u}_R$, we wish to determine $\vec{u}$ for all t in R. Such a specification of $\vec{u}$ is obtained as solution to the initial-value equation

$$\frac{d}{dt}\vec{u} = L\vec{u} \qquad (22)$$

where L is an algebraic operator for ordinary differential equations or a spatial differential operator for partial differential equations; $\vec{u}^0$ are the initial conditions and $\vec{u}_R$ are the boundary conditions.

As implied by this formulation, any high-order (in the time derivative) partial differential equation (say of order n) may be written as n first-order (in the time derivative) partial differential equation. Such an example is illustrated in the case of waves on a stretched string (Section 2.1), where by defining two dependent variables of translational velocity of displacement and the local angle to the equilibrium position of the string, two coupled first-order equations are obtained from the wave equation. Another simple example for illustrating the approach is the equation of motion of particles in, say, a gravitational field. The position of a star in space may be determined from Newton's second law of motion

$$\frac{d^2\vec{x}}{dt^2} = -\vec{\nabla}\phi(\vec{x}, t)$$

where ϕ is the gravitational potential through which the star moves. The equation is reduced to two first-order equations by defining the velocity $\vec{v}(t)$ of the particle

$$\frac{d\vec{x}}{dt} = \vec{v}$$

$$\frac{d\vec{v}}{dt} = -\vec{\nabla}\phi(\vec{x}, t)$$

6. DISCRETE ARITHMETIC ON THE COMPUTER: INTEGRATION OF THE INITIAL-VALUE PROBLEM ON A TIME MESH

The essential property of the description of physical systems on the computer is that they be discrete and finite. This is demanded in the first instance because the memory bank of a computer is finite: only a finite number of variables may be stored in the machine (typically, on large present-day machines, the central memory may have variable storage locations of the order of 128000 "words"). However, there is a further limitation; since the computer performs arithmetic in a finite amount of time, only a finite number of arithmetic operations may be performed (e.g. on the CDC 6600, of the order of 10^7 operations may be performed per second).

Thus we may not describe a continuum in its entirety. Approximations must be employed to discretize continuous functions, and the partial differential equations of physics. Let us consider the initial-value problem and attempt to obtain a discrete representation of the solution in time. This may be achieved most readily by defining a mesh or lattice in time, by dividing the time dimension into a set of small time intervals Δt^n, and by describing the evolution of the system in time by a representative set of solutions at each point in time between the time intervals

$$t^n = \sum_{m=0}^{n} \Delta t^m$$

Given the state of the system at time t^n, $\vec{u}^n$, the solutions may be obtained at the next time-step t^{n+1} over the interval Δt^{n+1} by integrating the time-dependent equations (Eq. (22))

$$\int_{t^n}^{t^{n+1}} \frac{d\vec{u}}{dt}\, dt = \int_{t^n}^{t^{n+1}} L\vec{u}\, dt$$

Consequently,

$$\vec{u}^{n+1} = \vec{u}^n + \int_{t^n}^{t^{n+1}} L\vec{u}\, dt$$

The integral on the right-hand side may not be evaluated exactly since the state u is not known for all times t in the intervals $t^n \leq t \leq t^{n+1}$. The essential approximation is now employed. We assume that if the time interval Δt^{n+1} is small, the integral may be approximated by some time average between t^n and t^{n+1}

$$\vec{u}^{n+1} = \vec{u}^n + (1-\epsilon)\Delta t^{n+1}(L\vec{u})^n + \epsilon \Delta t^{n+1}(L\vec{u})^{n+1} \qquad (23)$$

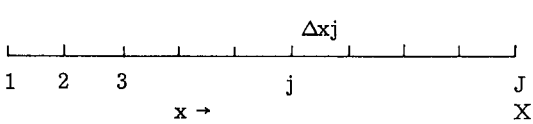
where the parameter ϵ is an interpolation parameter, $0 \leq \epsilon \leq 1$. Therefore,

$$(I - \epsilon \Delta t^{n+1} L^{n+1})\vec{u}^{n+1} = (I + (1-\epsilon)\Delta t^{n+1} L^n)u^n$$

I is the unit operator and, in a non-linear problem, $L^n = L(\vec{u}^n)$. In general, a matrix equation is obtained at each time-step relating the solution, $\vec{u}^n$, at the old time, t^n, to the solution at the new time-step, $\vec{u}^{n+1}$. Initial conditions define the state $\vec{u}^0$ at $t = t^0$. Hence we may proceed over many successive time-steps obtaining the solutions $\vec{u}^n$ successively. In the case where $\epsilon = 0$, the method is said to be explicit, since the new state $\vec{u}^{n+1}$ is defined explicitly by the equations; otherwise, the method is said to be implicit. The requirements, properties and examples of such difference solutions are dealt with in paper SMR-9/14b in these Proceedings.

7. SPACE MESH

The concept of a time mesh has been introduced to discretize the time dimension. It is clear that the space dimensions must be discretized as well, and a space mesh or lattice is introduced to approximate continuous functions. If x is a spatial independent variable, it is divided by a set of lattice points j, $1 \leq j \leq J$.



$$x_j = \sum_{k=1}^{j-1} \Delta x_k$$

Given a continuous function f(x), the function f may be represented by a vector $\{f_j\}$ such that

$$f_j = f(x_j)$$

A continuous function $f^*(x)$, which approximates f(x), may be obtained by interpolation.

For $x_j \leqq x \leqq x_{j+1}$

$$f^*(x) = \frac{(x_{j+1} - x)}{(x_{j+1} - x_j)} f_j + \frac{(x - x_j)}{(x_{j+1} - x_j)} f_{j+1}$$

Clearly, for the case where f(x) is a slowly varying function of x, $f^*(x)$ is a good approximation for f(x). This idea of a "good" or "poor" approximation to the function f(x) may be established quantitatively by expanding the function f(x) and the vector $\{f_j\}$ in Fourier series. The application of Fourier series is discussed in Section 2 where it is stressed that, in general, a continuous function, f(x), may be expanded over a periodic interval X into an infinite set of Fourier modes of wavelengths $X/n \to \infty$ for $n = 0$. The vector representation $\{f_j\}$ is of finite dimension J, and $\{f_j\}$ may only be expanded by the finite Fourier series of J modes

$$f_j = \sum_{k=0}^{J-1} g_k \exp\left(\frac{i2\pi kj}{J}\right)$$

where

$$g_k = \frac{1}{J} \sum_{j=1}^{J} f_j \exp\left(\frac{-i2\pi kj}{J}\right)$$

Thus, the representation $\{f_j\}$ includes only the long-wavelength modes belonging to the function f. The difference approximation therefore is a long-wavelength approximation. When the function f is rapidly varying (the amplitudes of the short-wavelength modes are large), the errors in a difference approximation will be very large and vice versa.

Having defined a space mesh, difference approximations to continuous derivatives may be formulated. Consider the first derivative df/dx. Clearly, a consistent approximation on the mesh to the first derivative is the difference derivative (Fig. 5)

$$\Delta_x^1 f_j = \frac{f_{j+1} - f_{j-1}}{\Delta x_{j+1} + \Delta x_{j-1}}$$

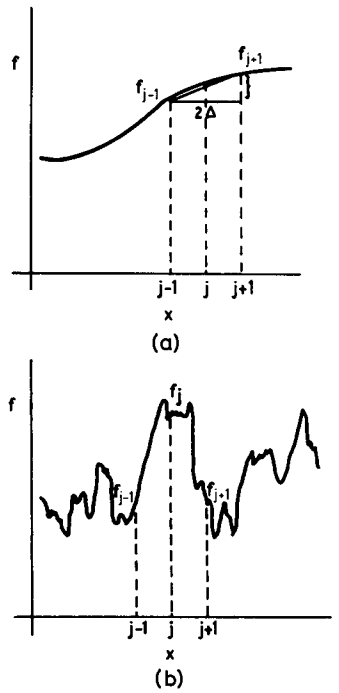


FIG. 5. (a) The conventionally used difference derivative $((f_{j+1} - f_{j-1})/2\Delta)$ as an approximation to the differential derivative df/dx at the point x_j. The difference method is a "good" approximation if functions are slowly varying (the amplitudes of the long-wavelength modes are large).
(b) The difference method is a "poor" approximation if functions are rapidly varying (the amplitudes of the short-wavelength modes are large).

Similarly, the second-space derivative is usually approximated by the difference derivative

$$\Delta_x^{11} f_j = \frac{f_{j+1} - 2f_j + f_{j-1}}{\Delta^2}$$

if $\Delta = \Delta x_j = \Delta x_{j-1}$. Again, these approximations are "good", in some sense, in the case where f is a slowly varying function. These ideas are quantified in paper SMR-9/14b in these Proceedings, where the extent of the difference approximation is evaluated and where particular difference schemes and algorithms for important systems of partial differential equations are developed.

REFERENCES

[1] POTTER, D.E., An Introduction to Computational Physics, to be published, John Wiley, London (1972).
[2] COURANT, R., HILBERT, D., Methods of Mathematical Physics, John Wiley (Interscience), New York (1953), Vol. 1, p. 69-82; Vol. 2.
[3] DAVYDOV, A.S., Quantum Mechanics, Pergamon Press, London (1966) 350.

DIFFERENCE SCHEMES AND NUMERICAL ALGORITHMS

D.E. POTTER
Department of Plasma Physics,
Imperial College,
London, United Kingdom

Abstract

DIFFERENCE SCHEMES AND NUMERICAL ALGORITHMS.

The methods of numerical analysis as applied to the differential equations and systems of partial differential equations in physics are introduced. The techniques of Fourier analysis are used to analyse difference algorithms. The initial-value problem is introduced and the requirements of a numerical integration scheme are discussed with particular reference to the consistency, stability, accuracy and efficiency of various methods. By considering time-dependent ordinary differential equations, the Euler, leapfrog, two-step, and implicit methods are introduced and subsequently applied to partial differential equations. The von Neumann stability condition is discussed with the aid of Fourier analysis and is applied, for various difference methods, to the diffusion and advective equations. The conditions for stability are related to the physical times of interest discussed in paper SMR-9/14a. Dispersion and diffusion on a difference mesh are illustrated and conservative schemes considered. Finally, methods of solving systems of parabolic equations and systems of hyperbolic equations are summarized.

1. INTRODUCTION

In paper SMR-9/14a in these Proceedings, the occurrence, the form and the simple properties of partial differential equations in physics are surveyed and it is shown how equations, continuous in space and time, must necessarily be formulated for the computer by finite representations on discrete lattices or meshes in space and time. It is suggested that the essential approximation of such difference methods is a long-wavelength approximation. In this paper, it is illustrated how the extent of the difference approximation may be evaluated quantitatively, and from this knowledge a variety of important numerical schemes and their properties may be enumerated.

First, the difference calculus is considered; subsequently, by considering ordinary differential equations, a number of important integration schemes for the initial-value problem are listed which are also essential for the solution of partial differential equations. Then, with the aid of Fourier techniques, an analysis of difference solutions to systems of partial differential equations is given.

2. DIFFERENCE CALCULUS AND THE EXTENT OF THE DIFFERENCE APPROXIMATION

A continuous function $f(x)$ may be represented by a vector $\{f_j\}$ on a difference mesh $1 \leq j \leq J$,[1] and, while for periodic boundary conditions $f(x)$ may

[1] Paper SMR-9/14a, these Proceedings.

be expanded by an infinite Fourier series, the vector $\{f_j\}$ may only be represented by a finite Fourier series in the long-wavelength modes. The first difference derivative may be compared with the first differential derivative by considering the effect of each operator, respectively, on a Fourier mode

$$u = \hat{u}\, e^{ikx}$$

The first differential derivative is

$$\frac{du}{dx} = iku \tag{1}$$

The first difference derivative has the form[1]

$$\begin{aligned}
\Delta_j^1 u &= \frac{u_{j+1} - u_{j-1}}{2\Delta} \\
&= \frac{\hat{u}}{2\Delta}\left(e^{ik(x_j+\Delta)} - e^{ik(x_j-\Delta)}\right) \\
&= \frac{\hat{u}\, e^{ikx_j}}{\Delta}\left(\tfrac{1}{2}\, e^{ik\Delta} - \tfrac{1}{2}\, e^{-ik\Delta}\right) \\
&= \hat{u}\, e^{ikx_j}\, \frac{i \sin k\Delta}{\Delta} \\
&= \frac{i \sin k\Delta}{\Delta}\, u_j
\end{aligned} \tag{2}$$

(3)

Clearly, in the limit of long wavelengths where $k\Delta$ is small, the difference derivative Δ^1 is a good approximation to the differential derivative d/dx. The right-hand side of Eq.(3) may be expanded in small $k\Delta$

$$\Delta^1 u = ik\left\{1 - \frac{(k\Delta)^2}{6} + O(k^4\Delta^4)\right\} u_j$$

$$\Delta^1 \equiv \left\{1 - \frac{(k\Delta)^2}{6} + O(k^4\Delta^4)\right\} \frac{d}{dx}$$

For the short wavelengths, where $k\Delta$ approaches one, the difference approximation is a poor one and the errors are large.

Similarly, we may compare the second difference derivative with the second differential derivative and estimate the approximation

$$\Delta_j^{11} u = \frac{u_{j+1} - 2u_j + u_{j+1}}{\Delta^2} \tag{4}$$

[1] Paper SMR-9/14a, these Proceedings.

For a Fourier mode $u = \hat{u}\, e^{ikx}$, of wave number k,

$$\frac{d^2u}{dx^2} = -k^2u$$

while

$$\Delta_j^{11}u = \frac{\hat{u}}{\Delta^2}\left\{e^{ik(x_j+\Delta)} - 2e^{ikx_j} + e^{ik(x_j-\Delta)}\right\}$$

$$= -u_j \frac{2}{\Delta^2}\left\{1 - \tfrac{1}{2}\left(e^{ik\Delta}+e^{-ik\Delta}\right)\right\}$$

$$= -u_j \frac{2}{\Delta^2}(1 - \cos k\Delta) \qquad (5)$$

For long wavelengths, where $k\Delta$ is small, we may expand the cosine, then,

$$\Delta^{11}u = -u\frac{2}{\Delta^2}\left\{1 - \left(1 - \frac{(k\Delta)^2}{2} + \frac{(k\Delta)^4}{12} + O(k\Delta^6)\right)\right\}$$

thus,

$$\Delta^{11}u = -k^2\left\{1 - \frac{k^2\Delta^2}{12} + O(k^4\Delta^4)\right\}u$$

or

$$\Delta^{11} \equiv \left\{1 - \frac{k^2\Delta^2}{12} + O(k^4\Delta^4)\right\}\frac{d^2}{dx^2} \qquad (6)$$

Again, for long wavelengths (small k) the approximation is a good one. The procedure illustrated here for first and second differences shows the general approach used in determining the accuracy of a difference algorithm and it may be applied readily to more complex schemes. Particularly for boundary value problems, the method may be applied in many dimensions and the sufficiency of the matrix representation of a differential operator may be determined. We shall not be concerned here further with the elliptic equation and boundary value problems, since these are analysed in some detail in papers SMR-9/13a, b, c in these Proceedings.

3. REQUIREMENTS OF A DIFFERENCE SOLUTION TO THE INITIAL-VALUE PROBLEM

The initial-value problem is formulated in Section 5 of paper SMR-9/14a in these Proceedings. The state of a system in some region R may be defined by the set of dependent variables which, written as a vector, $\vec{u}$, satisfy a set of first-order differential equations

$$\frac{d\vec{u}}{dt} = L\vec{u} \qquad (7)$$

where L in general is a spatial differential operator. Representing $\vec{u}$ on a difference mesh, L becomes a matrix operator and we could integrate this equation over small time-steps, Δt, to obtain solutions $\vec{u}^{n+1}$ at the new time-step $n+1$

$$(I - \epsilon \Delta t L)\vec{u}^{n+1} = (I + (1 - \epsilon)\Delta t L)\vec{u}^{n} \tag{8}$$

where $0 \leq \epsilon \leq 1$. Thus a difference scheme may be described by a difference operator C

$$\vec{u}^{n+1} = (I - \epsilon \Delta t L)^{-1}(I + (1 - \epsilon)\Delta t L)\vec{u}^{n} \tag{9}$$

$$\vec{u}^{n+1} = C\vec{u}^{n} \tag{10}$$

where the operator C is a function of the time-step and space-step, $C(\Delta t, \Delta)$. In choosing the operator C we must question what criteria must be satisfied for a particular difference scheme. The important properties of a difference approximation with which we shall be concerned may be summarized under the headings

consistency
accuracy
stability
efficiency

Clearly, the first requirement to be demanded is that in some manner our difference system approximates the differential system. Formally, the requirement of consistency may be specified as

$$\lim_{\substack{\Delta t \to 0 \\ \Delta \to 0 \\ \cdot\, \frac{\Delta t}{\Delta} \to \beta}} \frac{C(\Delta t, \Delta) - I}{\Delta t} = L \tag{11}$$

where β is some finite number; if this condition were not satisfied, the difference scheme would in no manner simulate the initial-value problem of interest.

Beyond this requirement, however, two sources of errors which affect the accuracy of the solution exist. The first of these is termed the truncation error which is caused by the approximation involved in simulating the differential equations by the difference equations. The essence of this approximation has been pointed out previously (Section 2), where we have seen that it arises from representing a continuous variable by a set of discrete points. We have seen that such errors are dependent on the mesh intervals in time and space, Δt, Δ, and we may readily determine the magnitude of the error. In choosing a difference scheme, we are required to minimize the truncation error.

A second source of errors (round-off) occurs because of the finite accuracy with which a particular variable is described in the memory of the computer. The arithmetic of the computer is not perfect. Obviously, if the calculation is pursued with i decimal places it will be less accurate than if it were executed with i + 1 decimal places. The error depends on how a number is "rounded off" in the computer, and to determine the total round-off error, a statistical analysis would have to be carried out. With modern digital machines, however, many decimal figures are or can be used and the cumulative round-off error is usually not serious.

It is nevertheless important to note that the arithmetic in the computer is not exact. This consideration leads us to the vital property of the stability of the difference method. A numerical method is stable if a small error at any step produces a smaller cumulative error. If this property were not demanded of a numerical method, an error occurring at any stage would grow without bound. For example, consider the case of a simple first-order ordinary differential equation and suppose that an error ϵ^n occurs at step n: we are interested in the amplification g of this error at step n + 1

$$\epsilon^{n+1} = g\epsilon^n \tag{12}$$

where g is the amplification factor which depends on the particular difference scheme, and we require for stability

$$|\epsilon^{n+1}| \leq |\epsilon^n|$$

consequently,

$$|g\epsilon^n| \leq |\epsilon^n|$$

$$|g| \leq 1 \tag{13}$$

Hence for a difference scheme, the condition (13) must be satisfied if the scheme is to be stable. This condition will be considered in more detail for partial differential equations.

The fourth property of the difference method which must be taken into account is the efficiency of the method. This may be defined as the total number of arithmetic and logical operations performed by the central processor of the computer to obtain a solution. On the one hand, the efficiency decreases with greater complexity of the particular difference method being applied. On the other hand, the accuracy of the solution can be increased with increasing complexity, and a compromise must be reached to obtain a viable method which is both accurate and efficient.

To illustrate these basic properties and to define four important methods of integrating in time, the case of a simple ordinary differential equation may be taken,

$$\frac{du}{dt} + f(u,t) = 0$$

As the simplest interesting example of such an equation, the decay equation may be given,

$$\frac{du}{dt} + \frac{u}{\tau} = 0$$

3.1. Euler method

If u is given at step n, namely u^n (u^0 is defined by initial conditions), we wish to determine u at step n+1. The simplest method is the Euler explicit first-order method (Fig. 1)

$$u^{n+1} = u^n - f(u^n, t^n)\,\Delta t \tag{14}$$

In the Euler method, the function f is evaluated only at time t^n, and hence the method is explicit and first-order accurate in the time-step Δt only. To investigate the stability of the method, it is assumed that a small error, ϵ^n, exists at step n and we question how the error is amplified to step n+1. To do so, the difference equation (14) is linearized about the small error ϵ^n

$$\epsilon^{n+1} = \epsilon^n - \left.\frac{\partial f}{\partial u}\right|_n \epsilon^n \Delta t$$

and using the definition for the amplification factor (Eq. (12)),

$$g = 1 - \left.\frac{\partial f}{\partial u}\right|_n \Delta t \tag{15}$$

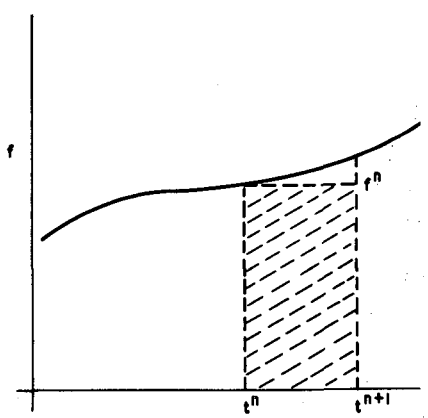


FIG. 1. Euler explicit method. Between time-steps t^n and t^{n+1}, only the first term in a Taylor expansion about time t^n is used to integrate the equations.

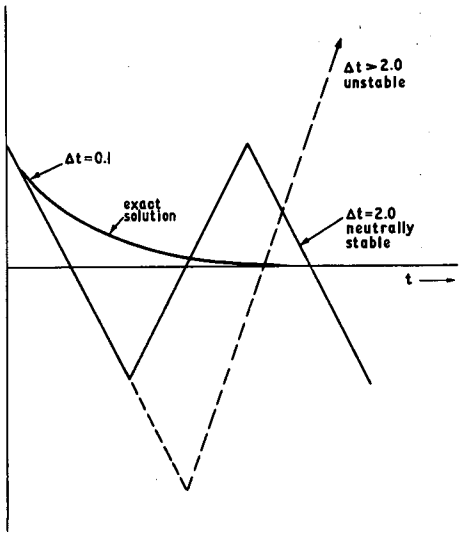


FIG.2. Example of stability in the Euler explicit solution to the decay equation (decay time $\tau = 1.0$). The three cases shown illustrate a stable, an unstable and a neutrally stable solution as the time-step Δt is varied about the stability condition ($\Delta t \leq 2.0$).

Clearly, for decay type equations, where $\partial f/\partial u \geq 0$, the stability condition is satisfied if the time-step is sufficiently small such that

$$\left.\frac{\partial f}{\partial u}\right|_n \Delta t \leq 2$$

$$\Delta t \leq \frac{2}{\left.\dfrac{\partial f}{\partial u}\right|_n} \tag{16}$$

For example, in the decay equation, with decay time τ, $\partial f/\partial u = 1/\tau$, we require for stability:

$$\Delta t \leq 2\tau \tag{17}$$

This is illustrated in Fig. 2, where the case of a stable, neutrally stable, and unstable solution occurs as the time-step, Δt, is varied according to the condition (7).

3.2. Leapfrog method

Second-order accuracy may be obtained by storing the variable u at two time-levels (Fig. 3):

$$u^{n+1} = u^{n-1} - f(u^n, t^n)2\Delta t$$

$$u^{n+2} = u^n - f(u^{n+1}, t^{n+1})2\Delta t \tag{18}$$

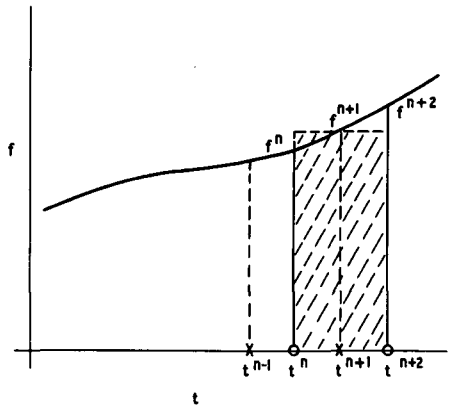


FIG. 3. Leapfrog scheme. Second-order accuracy is achieved by using a three-level formula at any step. The meshes (X and O) can become uncoupled.

The same method of stability analysis is applied as before, but now errors at three time-levels, ϵ^{n-1}, ϵ^n, ϵ^{n+1}, are related:

$$\epsilon^{n+1} = \epsilon^{n-1} - \left.\frac{\partial f}{\partial u}\right|_n 2\Delta t\,\epsilon^n \tag{19}$$

$$g^2 = 1 - \left.\frac{\partial f}{\partial u}\right|_n 2\Delta t g$$

Therefore,

$$g = -\frac{\alpha}{2} \pm \sqrt{\frac{\alpha^2}{4} + 1} \quad \text{where} \quad \alpha = \left.\frac{\partial f}{\partial u}\right|_n 2\Delta t \tag{20}$$

There are two roots for the amplification factor (corresponding to second-order accuracy) and for non-oscillatory equations, where the parameter α is not imaginary; the magnitude of the amplification factor for one of the roots is always greater than one. This arises because we have introduced an arbitrary computational mode, and the variables on the even mesh 2n are not coupled to the variables on the odd mesh 2n+1. This will be considered in more detail later, but clearly, if the computational mode can be removed, either because the equations are oscillatory (α is imaginary as in a pair of coupled equations) or by a filtering technique, the leapfrog scheme is particularly simple and has second-order accuracy.

3.3. Two-step explicit method

An extremely useful method with wide application is to "time centre" the integral, $\int_n^{n+1} f\,dt$, by a two-step process. The two-step method uses the

Euler explicit method as a first stage in evaluating the dependent variable, u, temporarily at time, $t = t^{n+\frac{1}{2}}$ (auxiliary calculation):

Auxiliary
$$u^{n+\frac{1}{2}} = u^n - f(t^n, u^n)\frac{\Delta t}{2} \tag{21}$$

Main
$$u^{n+1} = u^n - f(t^{n+\frac{1}{2}}, u^{n+\frac{1}{2}})\Delta t \tag{22}$$

The amplification factor g is

$$g = 1 - \alpha + \frac{\alpha^2}{2}, \text{ where } \alpha = \left.\frac{\partial f}{\partial u}\right|_n \Delta t$$

Again, for the case $\partial f/\partial u \geq 0$, stability is achieved for a sufficiently small time-step,

$$\Delta t \leq \frac{2.0}{\left.\frac{\partial f}{\partial u}\right|_n} \tag{23}$$

3.4. Implicit second-order method

In the three methods described above, the solution at each time-step is obtained in explicit form and a stability criterion for the time-step must always be satisfied. We may formulate an implicit second-order accurate method by evaluating the time integral $\int_n^{n+1} f\,dt$ by a time average (Fig. 4):

$$u^{n+1} = u^n - \tfrac{1}{2}\Delta t\left(f(u^n, t^n) + f(u^{n+1}, t^{n+1})\right) \tag{24}$$

The amplification factor, g, satisfies

$$g = 1 - \left.\frac{\partial f}{\partial u}\right|_n \frac{\Delta t}{2} - \left.\frac{\partial f}{\partial u}\right|_{n+1} \frac{\Delta t}{2} g$$

$$g = \frac{1 - \left.\frac{\partial f}{\partial u}\right|_n \Delta t}{1 + \left.\frac{\partial f}{\partial u}\right|_{n+1} \Delta t} \tag{25}$$

For the case $\partial f/\partial u \geq 0$, the magnitude of the amplification factor is always smaller than one, and the method is unconditionally stable, which is clearly a great advantage of an implicit method. We have not, however, obtained the new dependent variable, u^{n+1}, explicitly in terms of known quantities, and a possibly complex algebraic equation must still be solved at each time-step. For more sophisticated methods and a detailed discussion of ordinary differential equations, see Ref. [1].

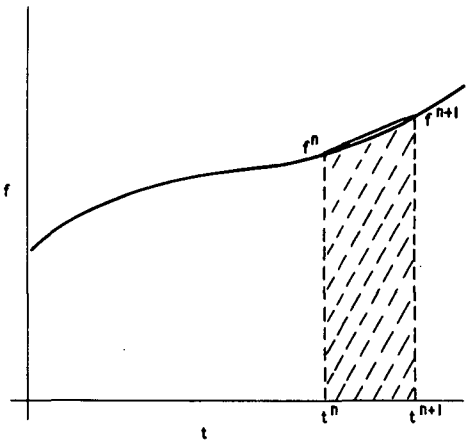


FIG.4. Implicit second-order scheme. Second-order accuracy is achieved by time-averaging the integration.

4. VON NEUMANN STABILITY CONDITION FOR PARTIAL DIFFERENTIAL EQUATIONS

The important properties of a difference solution to the initial-value problem have been enumerated and the nature of the difference approximation as a long-wavelength approximation in describing only the long-wavelength modes of the dependent variables has been described. For ordinary differential equations the important property of stability in the explicit case depends primarily on the magnitude of the time-step in comparison to the physical times of interest in the problem, and,by evaluating the dispersion relation of a system of partial differential equations,we have discussed the important time-scales of interest which occur in physical problems. These ideas may now be synthesized in evaluating the properties of a numerical solution to a system of partial differential equations.

For a system of partial differential equations the solutions are obtained at each time-step by operating a matrix $C(\Delta t, \Delta)$ on the solution at the previous time-step (see Eq.(10))

$$\vec{u}_j^{n+1} = C\,\vec{u}_j^{n} \tag{26}$$

To determine the stability or to obtain the dispersion relation of the difference scheme, the problem may be simplified by separately investigating the modes on the mesh. If C is constant (otherwise a linear approximation is applied) and the space interval is periodic, then for a Fourier mode

$$\vec{u}_j = \hat{\vec{u}}(k)\, e^{ikx_j}$$

the set of equations (Eq.(26)) may be transformed

$$\begin{aligned} \hat{\vec{u}}^{n+1} e^{ikx_j} &= C(\Delta t, \Delta)\,\hat{\vec{u}}^{n}(k)\, e^{ikx_j} \\ \vec{u}^{n+1}(k) &= G\,\hat{\vec{u}}^{n}(k) \end{aligned} \tag{27}$$

$G(\Delta t, k)$ is the amplification matrix of the difference scheme, for the Fourier mode k. For stability, we demand: if the amplitude of a Fourier mode is finite at time $t = 0$, then it must remain finite for all time-steps n. Namely,

$$|\hat{u}^n| < K|\hat{u}^0| \tag{28}$$

where K is a positive finite number. Using the definition of the amplification matrix,

$$\hat{u}^n(k) = G^n \hat{u}^0(k)$$

$$|G^n \hat{u}^0(k)| < K|\hat{u}^0(k)| \tag{29}$$

Therefore,

$$|G^n| < K$$

$$|G| < K^{\frac{1}{n}} \tag{30}$$

For large n, $K^{\frac{1}{n}} \to 1$. In general, G is a matrix of order I for a system of I first-order partial differential equations, and the magnitude of G is given by its determinant,

$$|G| = \prod_{i=1}^{I} g_i$$

We therefore make plausible the condition for stability

$$|g_i| \leq 1 \tag{31}$$

Von Neumann took into account the possible occurrence of a growing local term in the partial differential equation and he has shown that a necessary and sufficient condition for stability is (Ref. [2])

$$|g_i| \leq 1 + O(\Delta t) \qquad \text{for all i and k.} \tag{32}$$

The eigenvalues g_i can be complex:

$$|g_i| = +\sqrt{g_i^* g_i} \tag{33}$$

where g_i^* is the complex conjugate of g_i. In the partial differential equations of interest, the matrix C is in fact not a constant but might vary over the space and time lattice. The stability condition then reduces to a "local" condition, i.e. it must be satisfied everywhere on the mesh. The above discussion, generalizing the concept of stability in application to partial differential equations, is merely illustrative. For a more detailed and rigorous discussion, see Ref. [2].

5. EXPLICIT FIRST-ORDER SOLUTION TO THE DIFFUSION EQUATION

As a simple example to illustrate this approach to stability, we consider a first-order accurate (in the time-step) difference solution to the diffusion equation with constant conductivity, analogous to the Euler method. The diffusion equation

$$\frac{\partial u}{\partial t} - K \frac{\partial^2 u}{\partial x^2} = 0 \tag{34}$$

is simulated by the difference equation

$$u_j^{n+1} = u_j^n + \frac{K\Delta t}{\Delta^2}\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) \tag{35}$$

For a Fourier mode in space, $u_j^n = \hat{u}^n e^{ikx_j}$,

$$\hat{u}^{n+1}(k)\, e^{ikx_j} = \hat{u}^n e^{ikx_j} + \hat{u}^n \frac{K\Delta t}{\Delta^2}\left(e^{ik(x_j+\Delta)} - 2e^{ikx_j} + e^{ik(x_j-\Delta)}\right)$$

Consequently,

$$g = 1 - \frac{2K\Delta t}{\Delta^2}(1 - \cos k\Delta)$$

$$g = 1 - \frac{4K\Delta t}{\Delta^2}\sin^2\frac{k\Delta}{2} \tag{36}$$

Therefore, for $|g| \leq 1$,

$$\frac{4K\Delta t}{\Delta^2} \leq 2$$

$$\Delta t \leq 0.5\,\frac{\Delta^2}{K} \tag{37}$$

For stability, a maximum value of the time-step is obtained. This time corresponds to the largest diffusion time on the space mesh (Δ^2/K), and the result is hardly surprising. The accuracy of the method is first-order only in the time-step and second-order in the space-step, with errors e,

$$e = O(\Delta t) + O(\Delta^2)$$

6. ADVECTIVE EQUATION: EXPLICIT FIRST-ORDER INTEGRATION

We have seen how important the advective equation is, and we consider a possible difference solution in first order as for the diffusion equation. The advective equation

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0$$

is simulated on a difference mesh by

$$u_j^{n+1} = u_j^n - \frac{v\Delta t}{2\Delta}\left(u_{j+1}^n - u_{j-1}^n\right) \tag{38}$$

Considering a Fourier mode as before,

$$\hat{u}^{n+1} = \left(1 - \frac{v\Delta t}{\Delta} i \sin k\Delta\right)\hat{u}^n$$

$$g = 1 - \frac{v\Delta t}{\Delta} i \sin k\Delta \tag{39}$$

g is now complex and its magnitude in the complex plane is

$$|g|^2 = 1 + \left(\frac{v\Delta t}{\Delta} \sin k\Delta\right)^2 \tag{40}$$

Thus the von Neumann stability condition in this case cannot be satisfied for any time-step Δt, and the method is unconditionally unstable. This result leads to considerable difficulty in many fluid simulations.

We may, however, obtain a first-order explicit solution, with the loss of second-order accuracy in the space-step by replacing the algorithm (Eq.(38)) by

$$u_j^{n+1} = \tfrac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) - \frac{v\Delta t}{2\Delta}\left(u_{j+1}^n - u_{j-1}^n\right) \tag{41}$$

where the value of the variable at the old time-step has been replaced by a spatial average. Again, for a Fourier mode of wave number k, the amplification factor now becomes

$$g = \cos k\Delta - i \frac{v\Delta t}{\Delta} \sin k\Delta \tag{42}$$

Therefore,

$$gg^* = \cos^2 k\Delta + \left(\frac{v\Delta t}{\Delta}\right)^2 \sin^2 k\Delta$$

$$= 1 - \sin^2 k\Delta \left\{1 - \left(\frac{v\Delta t}{\Delta}\right)^2\right\} \tag{43}$$

and therefore the von Neumann stability condition (Eq.(32)) is satisfied if

$$\frac{|v|\Delta t}{\Delta} \leq 1$$

$$\Delta t \leq \frac{\Delta}{|v|} \tag{44}$$

This method has considerable importance in hyperbolic equations and is known as the Lax method. Again, the maximum permissible time-step is associated with the fastest speed on the mesh or the characteristic physical time described by the equations. This condition for the time-step for hyperbolic equations is known as the Courant-Friedrichs-Lewy condition [3].

7. DISPERSION AND DIFFUSION ON A DIFFERENCE MESH

The von Neumann stability condition has very wide applicability and permits us to obtain a stability criterion in the simplest way. It tells us little, however, of the more detailed properties of a particular difference solution. If mathematically the problem is not too complicated, ideally we may obtain the dispersion relation of the differential system. Let us consider the case of the Lax method applied to the advective equation. In one dimension, for a constant velocity v, as for the differential system[2], the dispersion relation for a Fourier mode in space and time is

$$\omega + vk = 0 \tag{45}$$

For the Lax difference scheme (Eq.(41)),

$$\hat{u} e^{i(\omega(t^n + \Delta t) - kx_j)} = \hat{u} e^{i\omega t^n} \left\{\cos k\Delta - \frac{v\Delta t}{\Delta} \sin k\Delta\right\} e^{ikx_j}$$

and the dispersion relation for the difference method is

$$e^{i\omega \Delta t} = \cos k\Delta - \frac{v\Delta t}{\Delta} \, i \sin k\Delta \tag{46}$$

---

[2] Section 2.2 of paper SMR-9/14a, these Proceedings.

In general, ω is complex. Let $\omega = \Omega + i\gamma$, and equating the real and imaginary parts of the dispersion relation (Eq.(46)) separately,

$$\tan \Omega\Delta t = \frac{v\Delta t}{\Delta} \tan k\Delta \tag{47}$$

$$e^{-2\gamma\Delta t} = \cos^2 k\Delta + \left(\frac{v\Delta t}{\Delta}\right)^2 \sin^2 k\Delta \tag{48}$$

In the special case where $v\Delta t/\Delta = 1$, $\gamma = 0$, and

$$\Omega\Delta t = - k\Delta$$

$$\Omega + vk = 0 \tag{49}$$

which is identically the dispersion relation of the differential system (Eq.(45)). In general, however, for a variable velocity v, or a non-linear problem, the parameter $v\Delta t/\Delta$ is smaller than one, at least on most points on the mesh. In this case, the imaginary part of the angular frequency exists, $\gamma > 0$, and

$$\Omega = \frac{1}{\Delta t} \tan^{-1}\left(\frac{v\Delta t}{\Delta} \tan k\Delta\right) \tag{50}$$

Thus, in general, the Lax method gives rise to diffusion on the mesh; unlike the differential system, Fourier modes decay on the mesh. In addition, $d\Omega/dk \neq v$ for all modes; different modes travel with different speeds on the mesh: the solution is dispersive. For long wavelengths ($k\Delta$ and $\Omega\Delta t$ small), an expansion of the difference dispersion relation (Eqs (47, 48)) shows agreement with the differential dispersion relation to first order. But to second order in the Lax method, particularly for short wavelengths, the effects of dispersion and diffusion can be very severe. This is a difficult problem in the simulation of hyperbolic equations that may only be minimized by turning to methods of second-order accuracy.

8. CONSERVATION ON A DIFFERENCE MESH

When a system of partial differential equations is non-linear, we may define a particular difference method, but there still remains a variety of ways of differencing non-linear terms. Since many of the partial differential equations of physics are conservative, as shown in paper SMR-9/14a, it would be useful to demand that the corresponding difference equations are themselves conservative. More specifically, we seek difference equations which identically conserve the energy, mass, momentum, and the magnetic flux of the system irrespective of the errors incurred by the finite difference lattice.

Consider, say, a rectangular region R bounded by the boundary B. The region R may be divided by a mesh into a set of elementary rectangular cells,

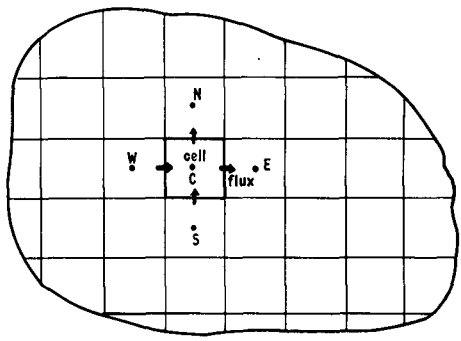


FIG.5. A conservative scheme employs integrated variables in each space cell, C, N, E, S, W. The fluxes are defined only on the surfaces which divide the cells. Hence, if a differential system is conservative, a difference scheme may be devised which identically conserves the same variables on the mesh.

each of volume $\Delta\tau$. In two dimensions, there are IJ rectangular boxes (Fig. 5). The system of conservative partial differential equations[3]

$$\frac{\partial \vec{u}}{\partial t} + \nabla \cdot \vec{\vec{f}} = 0 \tag{51}$$

may be integrated over each space-time box of volume $\Delta\tau\Delta t$ between the space-like surfaces t^{n+1}, t^n. At cell C,

$$\int_{t^n}^{t^{n+1}} dt \iint_C \frac{\partial \vec{u}}{\partial t} d\tau = -\int_{t^n}^{t^{n+1}} dt \iint_C \nabla \cdot \vec{\vec{f}}\, d\tau$$

Therefore,

$$\iint_C \vec{u}^{n+1} d\tau - \iint_C \vec{u}^{n}\, d\tau = \int_{t^n}^{t^{n+1}} dt \oint_{\substack{\text{Surface} \\ \text{of C}}} \vec{\vec{f}} \cdot d\vec{S} \tag{52}$$

where the left-hand side has been integrated over time, and on the right-hand side, the divergence theorem has been applied. On the mesh, instead of defining the intensive variable of, say, density or momentum density, the total mass or total momentum, respectively, in each box or cell may be defined. Thus,

$$\Delta\tau \vec{\tilde{u}}_{ij}^{\,n} = \iint_C \vec{u}_{ij}^{\,n}\, d\tau$$

[3] Section 1 of paper SMR-9/14a, these Proceedings.

In addition, the fluxes $\vec{\vec{f}} \cdot d\vec{S}$ are defined on the surfaces of each cell. For the cell ij, there are four fluxes (α = E, S, W, N),

$$\vec{F}_{\alpha_{ij}} = \int \vec{\vec{f}}_{ij} \cdot d\vec{S}$$

Therefore,

$$\tilde{\vec{u}}^{n+1}_{ij} = \tilde{\vec{u}}^{n}_{ij} - \int_{t^n}^{t^{n+1}} \frac{dt}{\Delta\tau} \sum_{\alpha} \vec{F}_{\alpha_{ij}} \qquad (53)$$

Although a particular difference scheme has not been defined, since we have not here defined the fluxes in time, this formulation has the considerable advantage that if the differential system conserves the variables $ud\tau$, the difference scheme also identically conserves these variables. For, if Eqs (53) are summed over all the boxes ij, in R, the fluxes cancel in pairs, since for example,

$$F_{E_{ij}} = - F_{W_{i+1,j}}$$

Therefore,

$$\sum_{i=1}^{I} \sum_{j=1}^{J} \tilde{\vec{u}}^{n+1}_{ij} = \sum_{i=1}^{I} \sum_{j=1}^{J} \tilde{\vec{u}}^{n}_{ij} - \sum_{B} \int \frac{dt}{\Delta\tau} F_{\alpha_{ij}}$$

Hence the variables $\int_R ud\tau$ are identically conserved in R except for the fluxes crossing the boundary B of R. By defining the fluxes $\vec{F}_\alpha$ in time, a large set of conservative difference schemes may be obtained.

9. SUMMARY OF METHODS FOR PARABOLIC EQUATIONS

With the techniques now available, a number of important and useful methods for solving parabolic difference equations are briefly summarized below. As an example, the simple diffusion equation in one dimension is used, but this is not to detract from the wide generality of the methods.

$$\frac{\partial u}{\partial t} - K \frac{\partial^2 u}{\partial x^2} = 0$$

9.1. Explicit first-order method

$$u_j^{n+1} = u_j^n + \frac{K\Delta t}{\Delta^2} \left(u_{j+1}^n - 2u_j^n + u_{j-1}^n \right) \qquad (54)$$

The amplification factor is

$$g(\Delta t, K) = 1 - \frac{2K\Delta t}{\Delta^2} (1 - \cos k\Delta)$$

for stability:

$$\Delta t \leq 0.5 \frac{\Delta^2}{K}$$

errors:

$$e = O(\Delta t) + O(\Delta^2)$$

9.2. Crank-Nicholson implicit method

Second-order accuracy in the time-step is obtained by using an implicit method:

$$u_j^{n+1} = u_j^n + \frac{K\Delta t}{2\Delta^2}\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) + \frac{K\Delta t}{2\Delta^2}\left(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n-1}\right) \tag{55}$$

The amplification factor is

$$g = \frac{1 - \frac{K\Delta t}{\Delta^2}\sin^2\frac{k\Delta}{2}}{1 + \frac{K\Delta t}{\Delta^2}\sin^2\frac{k\Delta}{2}}$$

and the scheme is therefore unconditionally stable with errors

$$e = O(\Delta t^2) + O(\Delta^2)$$

We are still left with the problem of solving a matrix equation for u_j^{n+1} on the mesh at each time-step.

9.3. Leapfrog method for the diffusion equation

A seemingly consistent method, as for ordinary differential equations, is the two-time-level leapfrog method

$$u_j^{n+1} = u_j^{n-1} + \frac{2K\Delta t}{\Delta^2}\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) \tag{56}$$

The amplification factor for $\alpha = \frac{4K\Delta t}{\Delta^2}(1-\cos k\Delta)$ is

$$g = \frac{-\alpha}{2} \pm \sqrt{\frac{\alpha^2}{4} + 1}$$

g can be less than minus one for all α and the method is therefore unconditionally unstable.

9.4. Dufort-Frankel method

By slightly altering the leapfrog scheme (Eq.(56)), the Dufort-Frankel scheme is obtained:

$$u_j^{n+1} = u_j^{n-1} + \frac{2K\Delta t}{\Delta^2}\left(u_{j+1}^n - \left(u_j^{n+1} + u_j^{n-1}\right) + u_{j-1}^n\right) \tag{57}$$

For $\alpha = \frac{2K\Delta t}{\Delta^2}$:

$$u_j^{n+1} = u_j^{n-1}\left(\frac{1-\alpha}{1+\alpha}\right) + \frac{\alpha}{1+\alpha}\left(u_{j+1}^n + u_{j-1}^n\right)$$

The scheme is explicit, and the amplification factor is

$$g = \frac{1}{1+\alpha}\left\{\alpha \cos k\Delta \pm \sqrt{1 - \alpha^2 \sin k\Delta}\right\}$$

The modulus of the amplification factor g is always less than unity, so that the method is unconditionally stable, with errors

$$e = O(\Delta t^2) + O(\Delta^2) + O\left(\left(\frac{\Delta t}{\Delta}\right)^2\right)$$

10. SUMMARY OF METHODS FOR HYPERBOLIC EQUATIONS

To define these methods, the set of hyperbolic conservative equations in one dimension are taken as an example

$$\frac{\partial \vec{u}}{\partial t} + \frac{\partial}{\partial x}\vec{F} = 0$$

where $\vec{F} = \vec{F}(\vec{u})$. It is found that stability criteria (Courant-Friedrichs-Lewy condition [3]) are obtained

$$\Delta t \leq C\frac{\Delta}{|v|} \tag{58}$$

where C is a constant of the order of unity and v is the largest velocity on the mesh. For the advective equation, the fluxes are defined by $\vec{F} = v\vec{u}$, and v is some centre-of-mass velocity. Alternatively, in wave-like equations, v is a phase velocity.

10.1. Lax first-order scheme

$$\vec{u}_j^{n+1} = \tfrac{1}{2}\left(\vec{u}_{j+1}^n + \vec{u}_{j-1}^n\right) - \left(\vec{F}_{j+1}^n - \vec{F}_{j-1}^n\right)\frac{\Delta t}{2\Delta} \tag{59}$$

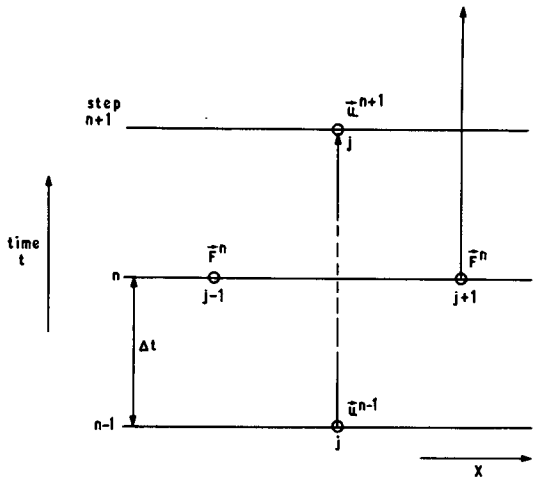


FIG.6. The conservative leapfrog scheme, which may be applied to conservative hyperbolic equations. A three-time-level formula is used (n-1, n, n+1). The time derivatives are defined between the levels n-1 and n+1, and the fluxes are defined at the intermediate space-time points.

where, for the advective equation, the amplification factor g is

$$g = \cos k\Delta - i \frac{v\Delta t}{\Delta} \sin k\Delta$$

and, therefore, for stability,

$$\Delta t \leq \frac{\Delta}{|v|}$$

$$e = O(\Delta t) + O(\Delta)$$

10.2. Leapfrog scheme

Three time levels are used to obtain the difference equations (Fig.6)

$$\vec{u}_j^{n+1} = \vec{u}_j^{n-1} - \frac{\Delta t}{\Delta}\left(\vec{F}_{j+1}^{n} - \vec{F}_{j-1}^{n}\right) \tag{60}$$

and the amplification factors g are

$$g = i\alpha \pm \sqrt{-\alpha^2 + 1}$$

where

$$\alpha = \frac{v\Delta t}{\Delta} \sin k\Delta$$

and, for a sufficiently small time-step, $\Delta t \leq \Delta/|v|$, stability is achieved with the magnitude of the amplification factor equal to one. Hence no diffusion arises, and the errors are of the order

$$e = O(\Delta^2) + O(\Delta t^2)$$

Decoupling between the meshes can occur, however.

10.3. Two-step (Lax-Wendroff) method

A method of very wide applicability for hyperbolic equations is the two-step method or Lax-Wendroff method. Just as for ordinary differential equations (Section 3.3), we evaluate temporary or auxiliary variables at the half time-step to time-centre the equations. The auxiliary step uses the Lax method:

$$\vec{u}_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \tfrac{1}{2}\left(\vec{u}_{j+1}^{n} + \vec{u}_{j}^{n}\right) - \frac{\Delta t}{2\Delta}\left(\vec{F}_{j+1}^{n} - \vec{F}_{j}^{n}\right) \tag{61}$$

which is used to define the fluxes $\vec{F}_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \vec{F}\left(\vec{u}_{j+\frac{1}{2}}^{n+\frac{1}{2}}\right)$ in the main step:

Main step:

$$\vec{u}_{j}^{n+1} = \vec{u}_{j}^{n} - \frac{\Delta t}{\Delta}\left(\vec{F}_{j+\frac{1}{2}}^{n+\frac{1}{2}} - \vec{F}_{j-\frac{1}{2}}^{n+\frac{1}{2}}\right) \tag{62}$$

The condition for stability is

$$\Delta t \leq \frac{\Delta}{|v|}$$

$$e = O(\Delta t^2) + O(\Delta^2)$$

As with the leapfrog method, diffusion only occurs to high order in $k\Delta$, but in this case no extraneous computational mode is introduced.

REFERENCES

[1] SMITH, G.D., Numerical Solution of Partial Differential Equations, Oxford University Press, Oxford (1968).
[2] RICHTMYER R.D., MORTON, K.W., Difference Methods for Initial-Value Problems, John Wiley (Interscience), New York (1967).
[3] COURANT, R., FRIEDRICHS, K.O., LEWY, H., Math. Annln 100 (1928) 32.

PLASMA PHYSICS, SPACE PHYSICS AND ASTROPHYSICS

D. E. POTTER
Department of Plasma Physics,
Imperial College,
London, United Kingdom

Abstract

PLASMA PHYSICS, SPACE PHYSICS AND ASTROPHYSICS.

Fluid problems which occur in the structure of stars, in the solar system and in laboratory and fusion plasmas are surveyed. These topics are described by the hydrodynamic equations with the inclusion of self-consistent long-range forces: in gravitational hydrodynamics, used to study the evolution, structure and behaviour of stars, the self-consistent gravitational field must be included; in magnetohydrodynamics, the self-consistent electromagnetic fields must be included. Systems of equations for the gravitational and electromagnetic cases are developed, and some simple properties of such equations are analysed. In particular, the important frequencies associated with the gravitational frequency and with Alfvén waves are stressed. The difference solutions of two particular problems are used to illustrate the general approach. In the gravitational case, a one-dimensional model of stellar pulsation is discussed and an implicit solution to the non-linear equations described. The approach is also applicable to one-dimensional problems in magnetohydrodynamics. In the electromagnetic case, explicit models for two-dimensional problems are briefly outlined. Such multidimensional problems are of importance in thermonuclear fusion physics, in solar flares and the magnetosphere.

1. INTRODUCTION: FLUID EQUATIONS AND LONG-RANGE FORCES IN LABORATORY AND ASTROPHYSICAL PLASMAS

The scope of the title of this paper is extremely broad, covering, as it does, a wide range of fluid objects from laboratory plasmas to stars and galactic fluids. To define the subject of the paper more precisely, we are interested in describing many-particle (that is nuclear or star-like) assemblies, interacting by long-range forces (gravitational or electromagnetic), by nuclear forces and by radiation transfer. We shall be concerned only with fluid descriptions of such assemblies, since the evolution of distributions in phase-space, as described by, say, the Vlasov equation, is discussed in papers SMR-9/13a, b, c in these Proceedings.

The hydrodynamic equations, which describe fluids interacting under their thermal pressure, as occurs in, say, the earth's atmosphere, are introduced in papers SMR-9/14a and SMR-9/17 in these Proceedings. We may describe many problems in fusion physics, in stars, and in the solar system by the inclusion of other forces and other energy transfer processes. In describing the structure and evolution of stars, the self-consistent gravitational field produced by the stellar mass must be included. In laboratory plasmas or the magnetosphere, the self-consistent magnetic fields must be incorporated and the plasma is described by magnetohydrodynamic (MHD) equations. These problems are strongly non-linear, since the occurrence of the fluid produces the field, which in turn interacts back as a force on the fluid, and apart from the simplest problems, solutions may

only be obtained computationally. In addition, experimentally, both stellar objects and laboratory plasmas are very difficult to study, firstly because they are both hot and destroy any probes, and secondly because both scale times and scale lengths are either very short (laboratory plasmas) or very long (stellar and space objects). A considerable understanding of such fluid problems may be obtained most readily by simulating interesting problems on the computer.

To illustrate the range of physics involved and to introduce some important problems, some example problems are listed which are described by gravitational hydrodynamics or magnetohydrodynamics.

1.1. Gravitational hydrodynamics

1.1.1. Evolution of stars

Once a star is formed by condensation from the galactic hydrogen matter, it is of interest to evolve the history of the star over long time-scales. A radial equilibrium is described by the expansive thermal pressure of the stellar matter and the contractive self-gravitational field of the star. Our dominant concern here is the equilibrium of energy, by production through nuclear reactions in the star interior on the one hand, and, on the other hand, by loss through outward radiation [1].

1.1.2. Stellar atmospheres

When investigating stars experimentally, it is the outer atmosphere which is observed. The gravitational hydrodynamic equations are solved in planar geometry, but to describe fully the stellar atmosphere, complex radiation phenomena must be studied, including many-species partially ionized phenomena [2].

1.1.3. Pulsation of stars

The full time-dependent gravitational hydrodynamic equations must be included in the non-linear problem when a star's radial equilibrium is unstable. Stars may pulsate radially over time-scales typically of 10^5 seconds in non-linear modes [3].

1.1.4. Gravitational collapse of stars

General relativistic hydrodynamic equations are used to describe the rapid collapse, prior to supernova explosions of old stars [4].

1.2. Magnetohydrodynamics

1.2.1. Laboratory plasmas and pinches

The macroscopic behaviour of dense laboratory plasmas is described by the interaction of plasma and magnetic pressure, produced by currents in the plasma [5].

1.2.2. Fusion plasmas

Plasma is trapped in very strong externally produced magnetic fields. The small fields produced by the plasma are nevertheless of great importance in defining the length of containment. These problems are essentially three-dimensional [5].

1.2.3. Magnetosphere

The magnetosphere is a three-dimensional problem in the interaction between the solar wind and the dipole magnetic field of the earth.

1.2.4. Solar flares

Also of interest in astrophysics are problems which involve both the gravitational field and the electromagnetic field in the hydrodynamic equations. In particular, one might include as examples of such problems the structure of pulsars and the magnetic field of galaxies.

2. HYDRODYNAMIC EQUATIONS WITH SELF-CONSISTENT FIELDS

2.1. Introduction to the equations

Classically and in three dimensions, we may describe a fluid hydrodynamically by a set of five time-dependent equations in the fluid density (ρ), momentum density ($\rho\vec{v}$), and internal energy density ($\rho\epsilon$) [6, 7].

Mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \qquad (1)$$

Acceleration:

$$\rho \frac{d\vec{v}}{dt} = - \nabla \cdot \vec{\vec{P}} + \vec{F} \qquad (2)$$

Internal energy:

$$\frac{\partial}{\partial t} (\rho\epsilon) + \vec{\vec{P}} \cdot \nabla \vec{v} + \nabla \cdot (\rho\epsilon \vec{v} + \vec{q}) = S(\rho, \epsilon) \qquad (3)$$

The pressure $\vec{\vec{P}}$ is, in general a tensor (with trace 3p - the scalar pressure) which might include viscous effects, and $\vec{q}$ is an energy flux which might include heat conduction or radiation diffusion. The gravitational or electromagnetic interactions may be included by defining additional forces $\vec{F}$ in the acceleration equation (2) and supplementing the hydrodynamic equations with equations for the fields. In the gravitational case,

$$\vec{F} = \rho \vec{g} = - \rho \nabla \phi \qquad (4)$$

where ϕ is a gravitational potential, determined self-consistently by Poisson's equation:

$$\nabla^2\phi = 4\pi G\rho \tag{5}$$

In studying stars, their evolution and structure, the energy flux $\vec{q}$ is a radiation diffusion term, and additional source terms, $S(\rho, \epsilon)$, may occur, for example, through nuclear reactions.

If L is a scale-length, these equations define in the first place the advection of the fluid (frequency $\omega = |\vec{v}|/L$) and secondly they describe sound waves (frequency $\omega_s \sim (1/L)\sqrt{\gamma p/\rho}$, where γ is the ratio of specific heats). In addition, gravitational oscillations occur with frequency $\omega_g = \sqrt{4\pi G\rho_0}$, and these couple to sound waves and lead, for example, to the phenomenon of pulsating stars. It is these characteristic frequencies we must follow in time-integrating the equations[1].

In the magnetohydrodynamic case, the force $\vec{F}$ is the magnetic force

$$\vec{F} = \frac{1}{c}\vec{j} \times B \tag{6}$$

where $\vec{B}$ is the magnetic field and $\vec{j}$ the current density, given by Ampère's law,

$$\vec{j} = \frac{c}{4\pi}\nabla \times \vec{B} \tag{7}$$

Unlike the gravitational case, the magnetic field, in general, must be described by time-dependent equations as defined by Faraday's law,

$$\frac{\partial \vec{B}}{\partial t} + c\nabla \times \vec{E} = 0 \tag{8}$$

The electric field to be used in this equation depends on the properties of the particular magneto-fluid being studied. In the simplest case, the Lorentz electric field (in the frame of the moving fluid) is defined by Ohm's law [5]

$$\vec{E} + \frac{1}{c}\vec{v} \times \vec{B} = \eta\vec{j} \tag{9}$$

Again, in the magnetohydrodynamic case, the characteristic frequencies associated with advection and sound waves occur. But, in addition, the magnetic field acts as a pressure on the fluid and consequently Alfvén or magnetosonic waves occur, with a characteristic frequency $\omega_A \sim L/V_A$, where V_A is the Alfvén speed,

$$V_A = \frac{B}{\sqrt{4\pi\rho}} \tag{10}$$

The resistive electric field in Faraday's law leads to the diffusion of the magnetic field, and correspondingly the source term, S, in the energy

---

[1] See papers SMR-9/14a and SMR-9/14b, these Proceedings.

equation (3), includes Joule heating. These equations are supplemented by an equation of state relating specific internal energy density to the pressure of the plasma, ,

$$\epsilon = \epsilon\,(p,\ \rho,\ \gamma) \tag{11}$$

2.2. Simple properties of the equations

Unlike hydrodynamic problems, the physical systems described above include gravitational energy ($\rho\phi$) and magnetic energy ($(1/8\pi)B^2$) respectively. Consequently, field energy and thermal energy are interchangeable and great variations in the fluid parameters can occur. This leads to strongly non-linear problems, and characteristic frequencies (gravitational frequency or Alfvén frequency) can vary very considerably over regions of the problem. For example, in star structure problems, densities between the inner core and the outer regions of stars vary by many orders of magnitude, and, consequently, the gravitational frequency may be very large in the core of a star but very small in the atmosphere of a star. In explicit calculations such large variations can lead to difficulties if the time-step is not to become too small.

Similarly, in the magnetohydrodynamic case, plasma densities may vary by three or more orders of magnitude and the Alfvén speed becomes very large in tenuous plasma (since $V_A = B/\sqrt{4\pi\rho}$). Thus the Courant-Friedrichs-Lewy condition[2] on an explicit time-step,

$$\Delta t \leq \frac{\Delta}{|\vec{v}| + V_A}$$

where Δ is the mesh space-step, can lead to very small time-steps. Approximations must be introduced to avoid small time-steps in such circumstances.

3. ONE-DIMENSIONAL GRAVITATIONAL CASE: STELLAR PULSATION

Some of the phenomena which arise from the study of stars and which have been investigated by fluid simulation have been outlined in the previous section. These problems are very diverse, but the essential approach may be illustrated particularly by describing the method of solution for problems in stellar pulsation. Up to the present time, these problems have been studied in one dimension and the possibly important processes of convection from the centre of stars have therefore not been described.

Stellar pulsation has been observed experimentally in certain stars for some time, particularly by the use of Doppler shift measurements from the surface of such stars. Typical frequencies of the pulsation are of the order of 10^5 s, associated with surface velocities of 50 km $\cdot$ s^{-1}. The method of simulation of this problem, which is discussed here, has been devised and applied by Christy [3, 8, 9].

---

[2] See paper SMR-9/14b, these Proceedings.

In one dimension the use of a Lagrangian mesh and the rapid solution of tridiagonal matrix equations [10], which facilitate the use of implicit difference methods, admit a relatively simple formulation of the problem on the computer. This is equally true for magnetohydrodynamic problems in one dimension where the same techniques are applied [11].

3.1. System of equations

If r is an independent variable describing a radial shell in the star, a Lagrangian mesh may be defined,

$$\frac{dr}{dt} = v \tag{12}$$

where v is the centre-of-mass fluid velocity. It is convenient to define as a variable the mass with the shell of radius r,

$$M(r) = \int_0^r 4\pi s^2 \rho(s)\, ds \tag{13}$$

From the continuity equation (Eq. (1)), in the Lagrangian frame,

$$\frac{dM}{dt} = 0 \tag{14}$$

Clearly, the mass contained within each spherical Lagrangian shell is a constant. Thus, in this simple geometry, Poisson's equation (5) may be integrated immediately to define the local gravitational acceleration,

$$g = -\nabla\phi = -\frac{GM(r)}{r^2} \tag{15}$$

and assuming a scalar pressure, $p\vec{\vec{I}} = \vec{\vec{P}}$, the acceleration equation (Eq. (2)) becomes

$$\frac{d\vec{v}}{dt} = -\frac{GM(r)}{r^2} - \frac{1}{\rho}\frac{\partial p}{\partial r} \tag{16}$$

It proves useful to use as the independent Lagrangian space variable the contained mass, M, rather than the radius, r (M is a scalar single-valued function of r). The acceleration equation takes the form:

$$\frac{d\vec{v}}{dt} = -\frac{GM}{r^2} - 4\pi r^2 \frac{\partial p}{\partial M} \tag{17}$$

and $r = r(M, t)$. The final equation for the energy or temperature of the fluid may be taken from the equation for the internal energy density (Eq. (3)),

$$\frac{\partial \epsilon}{\partial t} + p\frac{\partial\left(\frac{1}{\rho}\right)}{\partial t} + \frac{\partial}{\partial M}(q) = 0 \tag{18}$$

where a scalar pressure has been assumed, and the energy flux q is dominated by radiation diffusion. Alternatively, this equation may be combined with the equation for the kinetic energy and gravitational energy of the fluid to derive an equation for the conservation of total energy,

$$\frac{d}{dt}\left(\tfrac{1}{2}v^2 - \frac{GM}{r} + \epsilon\right) + \frac{\partial}{\partial M}(4\pi r^2 pv + q) = 0 \qquad (19)$$

The conserved quantity in the time derivative includes, respectively, kinetic energy, gravitational energy, and internal or thermal energy. Frequently, and as has been discussed in papers SMR-9/14a, SMR-9/14b (these Proceedings), it is preferable to use an equation for the conserved quantity (Eq.(19)) rather than the equation for the unconserved quantity, in this case the specific internal energy density (Eq.(18)). However, in this particular problem, large variations occur between the gravitational energy and thermal energy in different regions of the star, and in particular, there exist regions of low density on the surface of the star, where the calculation of the temperature from the total energy equation would yield large errors from the small thermal energy term. Hence in this problem the temperature is determined directly from the thermal energy equation (Eq.(18)).

The radiation flux term q depends essentially on the opacity, $\kappa(\rho, \epsilon)$, of the local stellar material. We apply Stefan's law and obtain an expression for the energy flux,

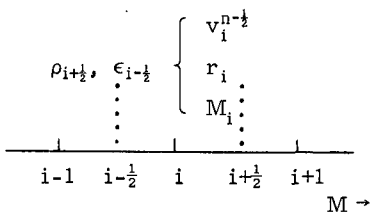
$$q = -4\pi r^2 \frac{4\sigma}{3}\frac{1}{\kappa}\frac{\partial T^4}{\rho \partial r} \qquad (20)$$

σ is the Stefan-Boltzmann constant. The inclusion of this term makes the internal energy equation (17) parabolic and describes the emission and absorption of radiation energy by the macroscopic coefficient, $1/\kappa$, which must be determined by considering the particular microscopic processes for a given problem: Bremsstrahlung radiation and absorption and indeed atomic phenomena. In the interior of stars where particles are fully ionized, a very good functional approximation for the opacity κ is $\kappa \sim \rho/T^3$, but in the stellar atmosphere, particularly through the "ionization front", κ may have a more complex form. Finally, we define the specific internal energy density by an equation of state (for example, in the fully ionized region $\epsilon = p/[\rho(\gamma-1)]$) which may be complex through the ionization front. The problem has now been reduced to three first-order, one-dimensional time-dependent equations (Eqs (11, 15, 17)) in the three dependent variables, $r(M,t)$, $v(M,t)$ and $\epsilon(M,t)$. The equations are essentially hyperbolic, describing advection (frequency ω), sound waves (ω_s) and gravitational oscillations (ω_G), but the inclusion of the radiation term makes the energy equation parabolic with a diffusion 'frequency'

$$\omega_R = \frac{\Delta^2}{\frac{4\sigma T^3}{3\rho\kappa}} \qquad (21)$$

3.2. Difference solution

The Lagrangian independent variable M is discretized by spherical shell boundaries i, which define mesh cells $i-\frac{1}{2}$,



The dependent variables of radius and velocity are defined at the mesh boundaries r_i^n, $v_i^{n-\frac{1}{2}}$, with the velocities defined at the half-time-steps $t^{n-\frac{1}{2}}$. All other variables (the intensive variables) may be defined as cell quantities at $i-\frac{1}{2}$ and at integer time-steps. Thus, given the radius and pressure, at integer times, the acceleration equation is integrated on a time-centred, space-centred scheme:

$$v_i^{n+\frac{1}{2}} = v_i^{n-\frac{1}{2}} - \Delta t^n \frac{(GM_i)}{(r_i^n)^2}$$

$$\times \frac{-\Delta t\ 4\pi (r_i^n)^2}{\frac{1}{2}(\Delta M_{i+\frac{1}{2}} + \Delta M_{i-\frac{1}{2}})} (p_{i+\frac{1}{2}}^n - p_{i-\frac{1}{2}}^n) \tag{22}$$

and, consequently, the new radial positions, at integer times, of the cell boundaries are determined,

$$r_i^{n+1} = r_i^n + v_i^{n+\frac{1}{2}} \Delta t^{n+\frac{1}{2}} \tag{23}$$

The local density of each cell is determined

$$\rho_{i-\frac{1}{2}}^{n+1} = \frac{M_i - M_{i-1}}{\frac{4}{3}\pi\left(\left(r_i^{n+1}\right)^3 - \left(r_{i-1}^{n+1}\right)^3\right)} \tag{24}$$

It remains to solve the equation for the energy density. The essential problem here is the non-linear diffusion equation, which is most readily solved implicitly by the stable Crank-Nicholson method, and a stability criterion related to the diffusion frequency (ω_R) is therefore avoided. Consider the equation for the internal energy density ϵ (Eq. (18)) which may be written as

$$\frac{\partial(\epsilon_0 T)}{\partial t} + p \frac{\partial\left(\frac{1}{\rho}\right)}{\partial t} - \frac{\partial}{\partial M} R \frac{\partial T}{\partial M} = 0 \tag{25}$$

where R is the radiation diffusion coefficient,

$$R = (4\pi r^2)^2 \cdot \frac{4\sigma}{3} \cdot \frac{4T^3}{\kappa\rho} \tag{26}$$

and we have written the internal energy density, $\epsilon = \epsilon_0 T$. The second term is explicitly known, but the radiation diffusion coefficient and the specific internal energy density are functions of the unknown temperature T. A solution may be obtained by iterating (over steps p) around the non-linear terms and using the Crank-Nicholson method[2] to time-difference the equation:

$$\epsilon_{0_{i+\frac{1}{2}}}^{p,n+1} T_{i+\frac{1}{2}}^{p+1,n+1} - \epsilon_{0_{i+\frac{1}{2}}}^{n} T_{i+\frac{1}{2}}^{n} - \frac{\Delta t}{2\Delta M_{i+\frac{1}{2}}} \left\{ \frac{(R_{i+\frac{3}{2}}^{p,n+1} + R_{i+\frac{1}{2}}^{p,n+1})}{\Delta M_{i+1}} \right.$$

$$\times \left(T_{i+\frac{3}{2}}^{p+1,n+1} - T_{i+\frac{1}{2}}^{p+1,n+1} \right) - \left. \frac{(R_{i+\frac{1}{2}}^{p,n+1} + R_{i-\frac{1}{2}}^{p,n+1})}{\Delta M_i} \left(T_{i+\frac{1}{2}}^{p+1,n+1} - T_{i-\frac{1}{2}}^{p+1,n+1} \right) \right\}$$

$$= S_{i+\frac{1}{2}}^{n} \qquad (27)$$

The equation is time-centred and space-centred and the term on the right-hand side is explicitly known. The left-hand side is a tridiagonal set of equations in the temperature T at each time-step n + 1 and at each iteration step p + 1. When the tridiagonal equations are simply inverted [5], improved coefficients $\epsilon_{0_{i+\frac{1}{2}}}^{p+1,n+1}$ and $R_{i+\frac{1}{2}}^{p+1,n+1}$ are determined over the mesh, and the linear tridiagonal equations are solved again to convergence. The time-step is then completed.

Because the gravitational frequency is an increasing function of the density and the time-step must be chosen smaller than the time of this frequency, the inner boundary is not taken at r = 0, but over a small, rigid, radiating sphere at the centre of the star. Equilibrium initial conditions are imposed. Christy [3] has obtained solutions for a wide range of equilibria. Non-linear steady-state oscillations (of amplitude typically 10%) are obtained illustrating the fundamental properties obtained experimentally. The detailed results are too extensive to discuss here.

The dominant modes found from the simulations are either the fundamental or first overtone (one node), and after several periods of oscillation, only one mode exists. Particularly of interest, however, is the result that, given the same equilibrium but perturbing the initial state with either a large-amplitude fundamental or a large-amplitude overtone, different steady-state oscillations may be obtained in either the fundamental or first overtone. Thus the final state "remembers" the time-history of the star [3].

4. TWO-DIMENSIONAL MAGNETOHYDRODYNAMIC SYSTEM – THE COAXIAL Z-PINCH

Unlike stellar problems, problems in magnetohydrodynamics, both in fusion and in the solar system, are of particular interest in two or three dimensions (toroidal fusion plasmas; the magnetosphere; solar flares). In two dimensions, even on present-day computers, the problem is fairly severe: for example, on a 64 × 64 mesh, typically we must solve six time-dependent

---

[2] See paper SMR-9/14b, these Proceedings.

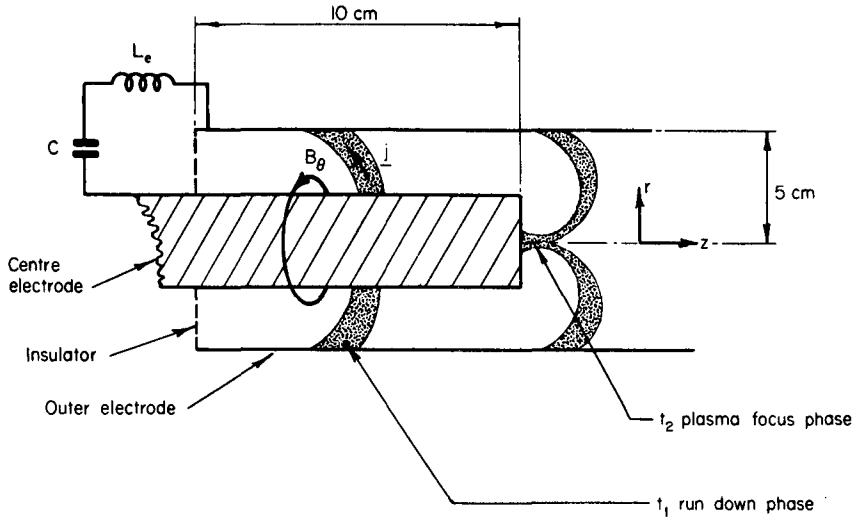


FIG. 1. Schematic of the plasma focus experiment. A capacitance at typically 40 kV discharges a current approaching 1 MA between coaxial electrodes. A two-dimensional supersonic shock is produced (time t_1) which collapses at the end of the centre electrode to form a thermonuclear plasma (time t_2). The calculation simulates the experiment in the (r, z) plane to which the magnetic field (B_θ) is always perpendicular.

coupled equations over probably one thousand time-steps, producing then about 3×10^7 dependent variables. Again, the variety of problems is immense, and the basic approach shall be illustrated by using as an example a two-dimensional explicit simulation of a laboratory non-cylindrical supersonic pinch, which occurs between coaxial electrodes and which produces a thermonuclear plasma (Fig. 1).

We may write the magnetohydrodynamic equations most conveniently in conservative form.

Mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{v} = 0 \tag{28}$$

Momentum:

$$\frac{\partial \rho \vec{v}}{\partial t} + \nabla \cdot \left(\rho \vec{v}\,\vec{v} + \vec{P} + \frac{B^2 \vec{I}}{8\pi} - \frac{\vec{B}\,\vec{B}}{4\pi} \right) = 0 \tag{29}$$

Magnetic flux:

$$\frac{\partial \vec{B}}{\partial t} + \nabla \cdot \left(\vec{v}\,\vec{B} - \vec{B}\,\vec{v} \right) = \nabla \times \eta \frac{c^2}{4\pi} \nabla \times \vec{B} \tag{30}$$

These equations are the same as given in Section 2, except that the magnetic force term has been included within the momentum flux tensor (Eq. (29)) to

obtain conservative equations. In general, in fully ionized plasmas it is necessary to define separate electron and ion internal energy densities, ϵ_e, ϵ_i, the equations for which are non-conservative.

Electron thermal energy:

$$\frac{\partial}{\partial t}\rho\epsilon_e + p_e \nabla\cdot\vec{v} + \nabla\cdot(\rho\epsilon_e\vec{v} + \vec{q}_e) = \eta j^2 + \rho\frac{\epsilon_i - \epsilon_e}{\tau_{eq}} \tag{31}$$

Ion thermal energy:

$$\frac{\partial}{\partial t}\rho\epsilon_i + p_i \nabla\cdot\vec{v} + \nabla\cdot(\rho\epsilon_i\vec{v} + \vec{q}_i) = -\vec{\vec{V}}:\nabla\vec{v} + \rho\frac{\epsilon_e - \epsilon_i}{\tau_{eq}} \tag{32}$$

$\vec{q}_e$ and $\vec{q}_i$ are the electron and ion heat conduction terms respectively. $\vec{\vec{V}}$ is the ion viscous tensor and τ_{eq} is the time for equipartition between electrons and ions. In general, in three dimensions, these equations define a set of nine dependent variables $\vec{u} = (\rho, \rho\vec{v}, \vec{B}, \rho\epsilon_e, \rho\epsilon_i)$ where equations of state relate $\epsilon_e = \epsilon_e(p_e, \rho)$, $\epsilon_i = \epsilon_i(p_i, \rho)$. In the absence of resistivity η, viscosity μ, and heat conductivities κ_e and κ_i, the equations are hyperbolic and describe advection ($\omega \sim \Delta/|\vec{v}|$), sound waves ($\omega \sim \Delta/v_s$), and Alfvén waves ($\omega \sim \Delta/v_A$), and in the explicit case the time-step is limited by a Courant-Friedrichs-Lewy condition,

$$\Delta t \leq \frac{\Delta}{|\vec{v}| + \sqrt{\frac{\gamma(p_e+p_i)}{\rho} + \frac{B^2}{4\pi\rho}}} \tag{33}$$

Unlike the one-dimensional case, matrix equations resulting from two- or three-dimensional problems are extremely difficult to solve, particularly for the non-linear or variable coefficient case. In the special case of Poisson's equation, the eigenvectors of a resulting difference matrix are known and a variety of exact methods of solution are available. This is not the case for the general elliptic equation with variable coefficients, and time-consuming iterative methods, such as the Alternating-Direction-Implicit or Chebyshev methods, would have to be applied at each time-step for the implicit solution of the above equations.

For multidimensional problems, explicit methods have been most successful. Certainly for high-beta supersonic problems (beta is the plasma pressure compared to the magnetic pressure), the diffusion processes are small but important and may also be included explicitly. To avoid large numerical diffusion, we use a second-order method in the time-step to integrate the equations; the Lax-Wendroff method is particularly appropriate. We may write the equations as

$$\frac{\partial\vec{u}}{\partial t} + \nabla\cdot\vec{\vec{F}} = \vec{S} \tag{34}$$

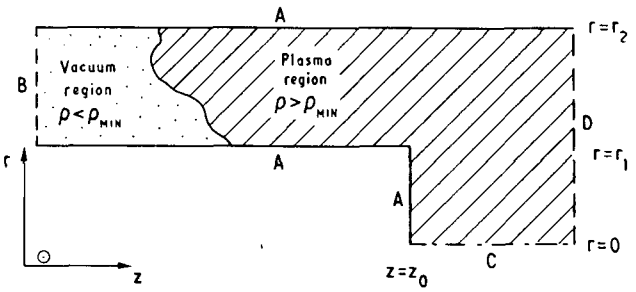


FIG. 2. In gravitational and electromagnetic hydrodynamic problems, large interchanges between field energy and thermal energy occur. The diagram illustrates the problem in the case of a magnetohydrodynamic simulation (plasma focus, Fig. 1) where in the vacuum region (large magnetic energy) the Alfvén speed becomes very large and a small time-step must be avoided. The calculation region is divided into a plasma region and a vacuum region: the time-independent electromagnetic equations must be solved in the two-dimensional vacuum region with a moving boundary. In the plasma region, the full magnetohydrodynamic equations are solved.

where the source terms $\vec{S}$ arise only in the case of the internal energy equations. Intermediate values in time and space are found by the Lax method. In two dimensions, at the point $(x_i, y_i) = C$, at time t^n:

Auxiliary step

$$\vec{u}_{C^1}^{\,n+\frac{1}{2}} = \frac{1}{4}\left(\vec{u}_{N^1}^{\,n} + \vec{u}_{S^1}^{\,n} + \vec{u}_{E^1}^{\,n} + \vec{u}_{W^1}^{\,n}\right) - \frac{\Delta t}{2\Delta}\left(\vec{F}_{x_{E^1}}^{\,n} - \vec{F}_{x_{W^1}}^{\,n} + \vec{F}_{y_{N^1}}^{\,n} - \vec{F}_{y_{S^1}}^{\,n}\right) \tag{35}$$

Main step

$$\vec{u}_{C^1}^{\,n+1} = \vec{u}_{C^1}^{\,n} - \frac{\Delta t}{\Delta}\left(\vec{F}_{x_E}^{\,n+\frac{1}{2}} - \vec{F}_{x_W}^{\,n+\frac{1}{2}} + \vec{F}_{y_N}^{\,n+\frac{1}{2}} - \vec{F}_{y_S}^{\,n+\frac{1}{2}}\right) \tag{36}$$

where $\vec{F}^{\,n+\frac{1}{2}} = \vec{F}(\vec{u}^{\,n+\frac{1}{2}})$ and the subscripts N, S, E, W refer to adjacent compass points.

The diffusion terms are also included explicitly by the same approach, but the source terms in the internal energy equations, of viscous heating, joule heating, equipartition and adiabatic compression, may be included implicitly.

One of the major problems in magnetohydrodynamic simulations is the occurrence of low-density regions where magnetic pressure replaces plasma pressure and Alfvén speeds become very large. In the case of a coaxial shock a "vacuum region" exists behind the shock, and we must avoid a very small time-step as defined by the Courant-Friedrichs-Lewy condition (Eq. (33)). Here the magnetic field is in the azimuthal direction; in the vacuum region (Fig. 2) we permit no currents to flow, so that the vacuum magnetic field is given by Ampere's law,

$$B_\theta \sim \frac{2I}{cr}$$

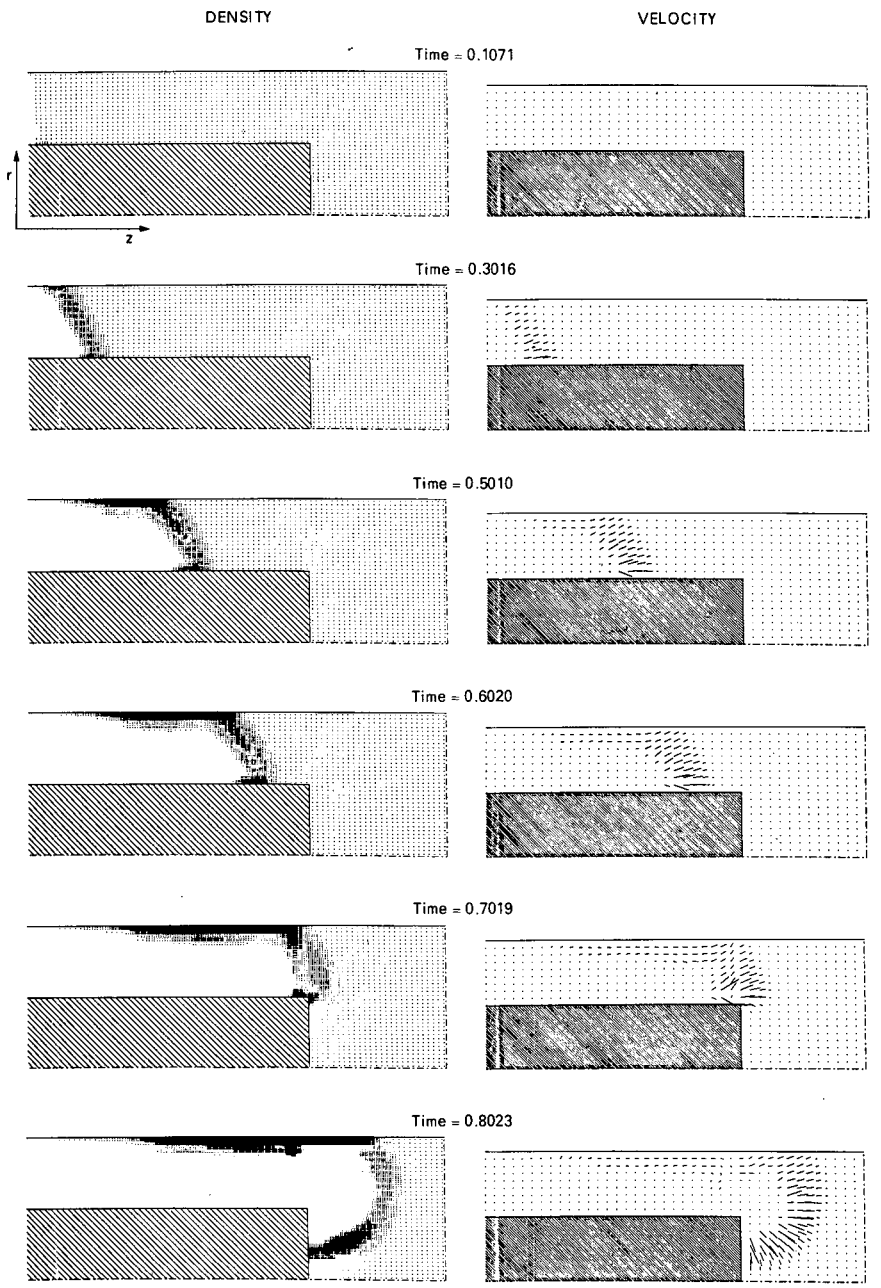


FIG. 3. Two-dimensional simulation of the plasma focus experiment. The diagrams illustrate, for successive times, the density in a moving shock and the associated flow velocity vectors.

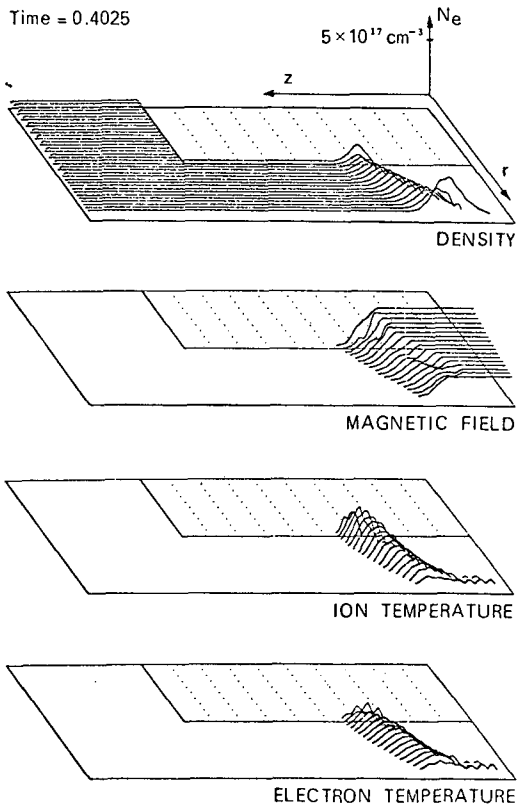


FIG. 4. Dependent variables in the plasma focus shock, drawn above the two-dimensional (r,z)-plane. Behind the shock, the large magnetic energy and low density, giving rise to very large Alfvén speeds, can be seen.

where I is the total current flowing in the circuit. When the magnetic fields exist in the plane of the calculation, it is necessary to solve Laplace's equation for the magnetic vector potential in the vacuum region with varying boundaries,

$$\nabla^2 \vec{A} = 0 \tag{37}$$

In laboratory plasmas and in the case of the coaxial gun, the calculational plasma domain is coupled with an external circuit which provides the electromagnetic energy,

$$\frac{d}{dt}(LI) = Q/C$$

$$L = \int_S \vec{B} \cdot d\vec{S}$$

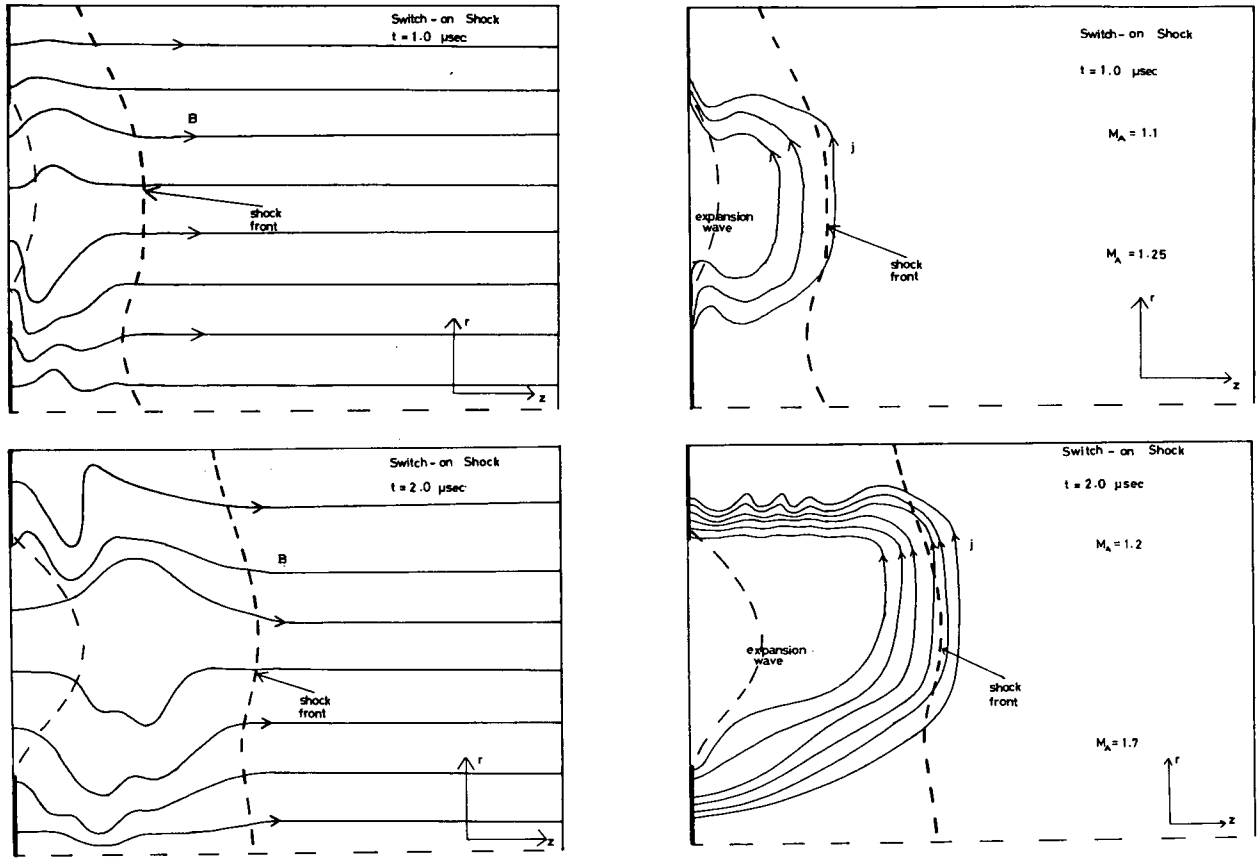


FIG. 5. Two-dimensional magnetohydrodynamic calculation with a three-dimensional magnetic field. The diagrams show, for two times, the motion of a "switch-on" shock parallel to an initial axial magnetic field. Behind the shock, an azimuthal component of magnetic field is switched on. In fusion problems and in the magnetosphere, it is essential to describe anisotropic phenomena, associated with a vector magnetic field, on multidimensional computational lattices.

where I is the current, L the inductance of the system, C a capacitance and Q the charge on the capacitance (Fig. 1).

As an example of such a simulation, numerical solutions of the motion of a two-dimensional coaxial shock and the consequent collapse to form a thermonuclear plasma on the axis of the system are illustrated in Figs 3 and 4. The shock travels with a speed $v_s = 2 \times 10^7$ cm · s^{-1} and temperatures of 1.5 keV are obtained in the hot contained pinch.

Another interesting example is the motion of a shock parallel to magnetic field lines, which switches on a third component of the magnetic field (Fig. 5). The reverse phenomenon (switch-off shock) is believed to be of importance in annihilating the magnetic field in solar flares.

In magnetohydrodynamics, the anisotropy introduced by the magnetic field leads to a variety of wave phenomena (Alfvén and Whistler waves) which travel preferentially along the magnetic field. In addition, the coefficients of resistivity, viscosity, and heat conductivities are tensor quantities associated with the magnetic field [12], and, in future simulations in both space physics and fusion physics, these anisotropic MHD phenomena must be simulated in two and three dimensions for the full elucidation of the subject [13].

REFERENCES

[1] KIPPENHAHN, R., WEIGERT, A., HOFMEISTER, E., in Methods in Computational Physics (ALDER, B., FERNBACH, S., ROTENBERG, M., Eds), Academic Press, New York, 7 (1967) 129.
[2] MIHALAS, D., in Methods in Computational Physics (ALDER, B., FERNBACH, S., ROTENBERG, M., Eds), Academic Press, New York, 7 (1967) 1.
[3] CHRISTY, R.F., Astrophys. J. 144 (1966) 108.
[4] MAY, M.M., WHITE, R.H., in Methods in Computational Physics (ALDER, B., FERNBACH, S., ROTENBERG, M., Eds), Academic Press, New York, 7 (1967) 219.
[5] ROBERTS, K.V., POTTER, D.E., in Methods in Computational Physics (ALDER, B., FERNBACH, S., ROTENBERG, M., Eds), Academic Press, New York, 9 (1970) 339.
[6] COWLING, T.G., Magnetohydrodynamics, John Wiley (Interscience), New York (1957).
[7] KILLEEN, J., "Difference Methods in Fluid Dynamics, with Applications", paper IAEA-SMR-9/17, these Proceedings.
[8] CHRISTY, R.F., Rev. mod. Phys. 36 (1964) 555.
[9] CHRISTY, R.F., in Methods in Computational Physics (ALDER, B., FERNBACH, S., ROTENBERG, M., Eds), Academic Press, New York, 7 (1967) 191.
[10] FOX, L., An Introduction to Numerical Linear Algebra, Oxford University Press, Oxford (1964).
[11] HAIN, K., HAIN, G., ROBERTS, K.V., ROBERTS, S.J., KÖPPENDORFER, W., Z. Naturf. 15a 12 (1960) 1039.
[12] SHKAROFSY, I.P., BERNSTEIN, I.B., ROBINSON, B.B., Physics Fluids 6 (1963) 40.
[13] POTTER, D.E., "Multidimensional magnetohydrodynamic calculations", Proc. Second Conf. Computational Physics, The Institute of Physics and the Physical Society, London (1970).

PLASMA, GRAVITATIONAL AND VORTEX SIMULATION

R. W. HOCKNEY
Computer Science Department,
Reading University, Reading,
United Kingdom

Abstract

PLASMA, GRAVITATIONAL AND VORTEX SIMULATION.

Particle models are first compared with fluid MHD-models and the time-stepping procedure described. A plasma model is described in detail, and it is shown how this may be re-scaled to become a gravitational or vortex simulation. Examples are given of the use of such models in the study of electron devices, galactic evolution and fluid flow stability.

In this paper, we shall discuss the use of particle/field calculations in the simulation of plasmas, galaxies and vortex flows. We shall describe the simulation in terms of the plasma case and show that by suitable scaling the same model can be used for the case of galaxies and vortices.

THE QUASI-ELECTROSTATIC PARTICLE MODEL

In a particle model, the plasma is represented by calculating the motion of a "large number" (usually 10^4 to 10^5) of representative ions and electrons as they move through the fields of all the other particles. The motion is governed by Newton's laws and the only field we consider in this model is the electrostatic field of the assembly (except perhaps for a fixed external magnetic field).

The model is termed "quasi-electrostatic" because, although the particles move dynamically according to Newton's laws, the field is at all times the electrostatic field due to the present positions of all particles. That is to say, we consider only non-relativistic velocities when the velocity of transmission of the field (i. e. the velocity of light) is very much greater than the velocities of the particles.

In contrast to an MHD-plasma, the main properties of such an electrostatic plasma arise from charge separation and inertia. These lead to:

(a) Plasma oscillations at a frequency ω_p given by $\omega_p^2 = 4\pi ne^2/m$ and a characteristic time of $\tau_p = 2\pi/\omega_p$. Here n, e and m are the density, charge and mass of the particle, respectively.

(b) Charge separation over distances of the order of the Debye length, λ_D given by $\lambda_D^2 = kT/(4\pi ne^2)$. The Debye length is a characteristic length of the system. Here k is the Boltzmann constant, and T the absolute temperature.

The above behaviour is in sharp contrast to the behaviour of an MHD-plasma which may be more familiar to many readers. In the MHD-plasma one is interested in time scales much greater than τ_p and distance scales much greater than λ_D. On this time and space scale, the system

may be considered to be neutral and all the effects present in the quasi-electrostatic plasma are ignored. Furthermore, in the MHD-plasma one's interest is focused on the magnetic field produced by currents flowing in the system and the effect of this field on the motion of the plasma. In the electrostatic plasma, the magnetic field of a moving charge is ignored.

The electrostatic particle model is useful in the study of

(a) small-scale, high-frequency micro-instabilities which give rise to anomalous diffusion and resistivity effects.

(b) Velocity-space instabilities, two-stream instability, Landau damping, Debye shielding.

(c) Problems involving many inter-penetrating streams.

However, because the magnetic field of a moving charge is ignored, the model is only applicable to low densities for which $\beta = nkT/(B_0^2/8\pi) \ll 1$, where B_0 is the strength of some external magnetic field imposed on the problem, for example, the containing field in the case of a problem in controlled nuclear fusion.

All computer models of a plasma are wrong in one way or another. A particle model is wrong in the first instance because we represent a real system of say 10^{15} particles by a model with at most about 10^5. We return to the question of the effect particle number on the properties of the model later on. We may also, as in the electrostatic model, ignore many of the fields actually present in the physical system.

An MHD-fluid model is wrong because the velocity distribution function of the particles, f(v), is not allowed to change and the model cannot cope with inter-penetrating streams. If, on the other hand, we go to a solution of Vlasov's equations which may be said to simulate a system of an infinite number of particles, to overcome the problems of particle number, then we swop errors due to using too few particles for errors due to having too few spatial and velocity mesh points on which to solve the Vlasov equation.

THE TIMESTEP LOOP

The simulation proceeds in a series of timesteps during which the field is assumed to remain constant. We store in the computer the co-ordinates $(x, y)^t$ and $(V_x, V_y)^{t-\frac{1}{2}DT}$ of all the particles, where t is the time and DT the timestep, and we find it convenient in the differencing to regard the velocity as given one half timestep before the positions. The positions and velocities are advanced stepwise in time as follows:

(a) Find the field on each particle due to the present positions $(x, y)^t$ of all particles.

(b) Accelerate each particle for a short time, DT according to Newton's laws;

$$m \frac{d\vec{V}}{dt} = q\left(\vec{E} + \frac{\vec{V} \times \vec{B}_0}{c}\right)$$

$$\frac{d\vec{x}}{dt} = \vec{V}$$

to obtain revised positions and velocities

$$(x, y)^{t+DT},\ (V_x\ V_y)^{t+\frac{1}{2}DT}$$

DIFFERENCING OF NEWTON'S EQUATIONS

The simplest differencing scheme is used which is centred in time

$$m\left(\frac{\vec{V}^{t+\frac{1}{2}DT} - \vec{V}^{t-\frac{1}{2}DT}}{DT}\right) = q\left\{\vec{E} + \left(\frac{\vec{V}^{t+\frac{1}{2}DT} + \vec{V}^{t-\frac{1}{2}DT}}{2}\right) \times \frac{\vec{B}_0}{c}\right\}$$

$$\frac{\vec{x}^{t+DT} - \vec{x}^{t}}{DT} = \vec{V}^{t+\frac{1}{2}DT}$$

where $\vec{x}$, $\vec{V}$, $\vec{B}_0$ are vectors.

This scheme is implicit in the new velocity $\vec{V}^{t+\frac{1}{2}DT}$; however, the equations are linear in the velocity and an explicit expression can be obtained for the new velocity. We note also that if the velocity is eliminated the resulting difference equation is the same as that obtained by differencing Newton's equation in the form

$$m\,\frac{d^2x}{dt^2} = \text{force}$$

with the second derivative represented as

$$\frac{d^2x}{dt^2} = \frac{x^{t-DT} - 2x^{t} + x^{t+DT}}{DT^2}$$

The integration in time is explicit and we may expect there to be a stability requirement limiting the size of the timestep. In fact, one may show that for stability $\omega_p DT < 2$, where ω_p is the highest plasma frequency, which is due to the electrons. This is a severe limitation on the size of the time-step and the magnitude of physical time that can be studied in this type of simulation.

FINDING THE FIELD

Two methods are available for finding the field in step (a), the method of "action at a distance" and the "mesh method".

Action at a distance

For small numbers of particles, $N < 1000$, it is possible to find the field on each particle by adding up contributions from all the other (N-1)

particles. Since this must be done for each particle the number of arithmetic operations and hence the computer time used will be proportional to N(N-1) or N^2 for large N.

The field on the i-th particle due to all others is

$$E_i = \sum_{\substack{j=1 \\ i \neq j}}^{N} \frac{q_j \vec{r}_{ij}}{|\vec{r}_{ij}|^2}$$

which would take about 10 N arithmetic operations to compute. To find the field on all N particles therefore takes about 10 N^2 operations.

To see the meaning of this in practice, let us consider a computer which takes about 1 μs for an arithmetic operation (e.g. CDC 6600, IBM 360/91). Then the time for the field calculation in each timestep is:

N = 100	timestep = 0.1 s
= 1000	= 10 s
= 10000	= 15 min
= 100000	= 1 d

Since a useful computer experiment will include at least 1000 timesteps, it is clear that at most 1000 particles can be moved in a simulation using the "action at a distance" method. Simulations of this type are performed in studies of clusters of a few hundred stars. For plasma and galactic simulations, however, one must seek an alternative method which will allow one to move many thousands of particles. To do this a method in which the number of operations is proportional to N, and not N^2, must be found.

The mesh method

In the mesh method the region of calculation is divided into a regular array of cells as shown in Fig. 1. At the centre of each cell, there is a mesh point at which the values of variables applying to that cell are calculated. In the present context, these variables are the charge density and the electrostatic potential. The field in the region is then determined by solving the appropriate differential equation by finite-difference methods operating on the variables given at the mesh points. To conserve storage it is usual to solve for the potential and then derive the fields by differencing as follows:

(1) examine (x, y) for each particle and assign a unit of charge to the appropriate mesh point. This gives a charge distribution $\rho(x, y)$ on the mesh.

(2) Solve the Poisson equation

$$\nabla^2 \phi = -4\pi\rho$$

to give the electrostatic potential. The 5-point difference approximation may be used

$$\phi_{i,j-1} + \phi_{i,j+1} + \phi_{i-1,j} + \phi_{i+1,j} - 4\phi_{i,j} = -4\pi\rho_{ij} H^2$$

where H is the mesh interval.

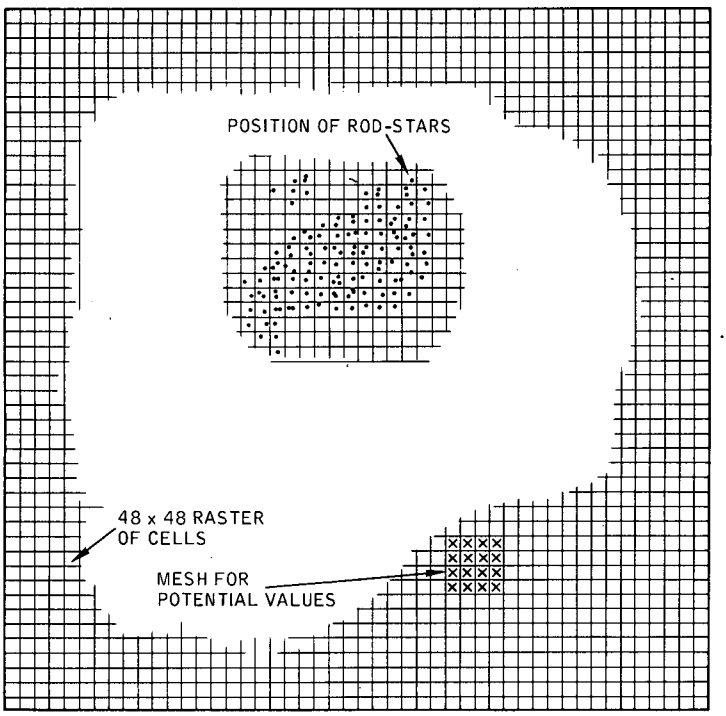


FIG. 1. The mesh method – showing the division of space into cells and the mesh points used in the solution of Poisson's equation. A typical problem showing "stars" on a 48 × 48 mesh. [from HOCKNEY, R. W., Astrophys. J. 150 (1967) 797, Chicago University Press].

(3) Derive the field in each cell by differencing

$$\vec{E} = -\text{grad } \phi$$

The simplest finite-difference approximation for the field in the (i, j) cell is

$$E_x = (\phi_{i-1,j} - \phi_{i+1,j})/2H$$

$$E_y = (\phi_{i,j-1} - \phi_{i,j+1})/2H$$

In the above scheme let us suppose there are on average p particles per mesh cell, then there will be N/p mesh points. It will be shown in another paper by the author in these Proceedings that, in simple geometries, the number of arithmetic operations required to solve Poisson's equation [1] is proportional to the number of mesh points (neglecting a slowly varying logarithmic dependence). Hence the number of operations for stage (2) above is proportional to the number of particles N. Stages (1)

and (3) also require a fixed number of operations for each particle and hence the total number of operations required by the mesh method is proportional to N. We find in a typical case that 100 000 particles can be moved one timestep in 5 s on a CDC 6600 or IBM 360/91 which is something like 100 times the number of particles that can be moved using action at a distance.

NGP approximation [2]

Particle models differ in the manner in which charge is assigned to the mesh in stage (1) above and the manner in which the field is obtained in stage (3). We have described above the simplest scheme, which is known as the nearest-grid-point approximation (NGP) in which all the charge of a particle is assigned to the nearest mesh point as shown in Fig. 2. In stage (3) the field is taken to be the same for all particles in the same cell and is given by the simplest difference given above.

The NGP is very simple computationally and can be reduced by appropriate scaling to a program like:

For every particle compute:

```
I = X
J = Y
EX = PHI(I-1,J) - PHI(I+1,J)
EY = PHI(I,J-1) - PH(I,J+1)
DX = DX + EX
X = X + DX
DY = DY + EY
Y = Y +DY
```

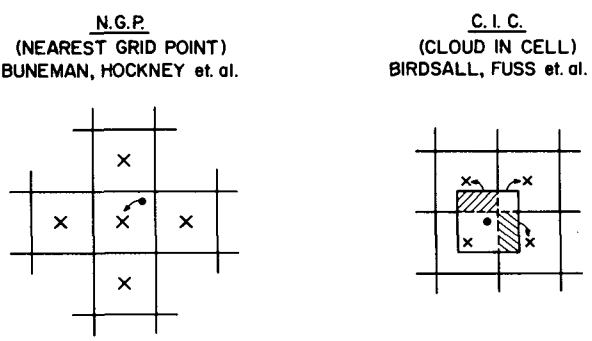


FIG. 2. The relation between cells, mesh (or grid) points and clouds in the NGP and CIC models. The region is divided into cells by straight lines. There is a mesh point shown by a cross at the centre of each cell. In CIC this is the centre of a square cloud. The shaded areas show the portion of the cloud in each cell, and the arrows the mesh points with which the parts are associated. [from HOCKNEY, R.W., Meth. Comput. Phys. 9 (1970) 135, Academic Press].

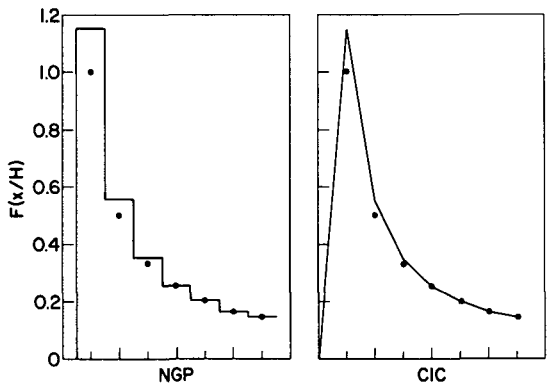


FIG. 3. The force F(x/H) between two particles with separation x/H for the NGP and CIC models. The dots show the $(x/H)^{-1}$ interaction between point charges. [from HOCKNEY, R.W., Meth. Comput. Phys. 9 (1970) 135, Academic Press].

such a loop has six operations per particle and is probably short enough to use the "loop mode" or instruction stack facilities of computers like the CDC 6600 or IBM 360/91. The charge assignment loop is particulary simple:

$$I = X$$

$$J = Y$$

$$Q(I, J) = Q(I, J) + CH$$

where CH is the unit of change.

The disadvantage with the NGP scheme is that the force between two interacting particles varies like a staircase with separation, as is shown in Fig. 3. The sharp changes at the steps lead to a high level of noise in the system and a poor conservation of energy (a few per cent conservation over 1000 steps is considered good).

CIC approximation [3]

To overcome the noise problems associated with NGP the cloud-in-cell method (CIC) was devised. In this method, the co-ordinate of a particle is regarded as the centre of a square cloud of change of uniform density. In assigning the density in stage (1) the charge of the cloud is apportioned to the four neighbouring mesh points according to the proportion of the cloud that lies in each of the four cells associated with the mesh points. In stage (3) the NGP field is calculated in each of the four neighbouring cells. The total field on the cloud is then found as the weighted average of the neighbouring fields, using as weights the areas of the cloud

in each cell. A program for the above method using an obvious notation might look like:

```
I = X
J = Y
```

Now work out the weights

```
U = X - I
V = Y - J
A = (1 - U)
B = (1 - V)
```

Now assign charge to the mesh points

```
Q01 = Q01 + A.V.CH
Q11 = Q11 + U.V.CH
Q00 = Q00 + A.B.CH
Q10 = Q10 + U.B.CH
```

where 00 refers to the bottom left mesh point in Fig. 2:
10 " " " " right " " "
01 " " " top left " " " "
11 " " " top right " " "
For the field calculation one has the NGP field in all cells:

```
E00 = PHI (left) - PHI (right) about 00 cell
E10 =      similarly            "   10  "
E11 =      similarly            "   11  "
E01 =      similarly            "   01  "
```

and similarly for the y-field; then the total field is

```
ET0TX = A.V.E01 + U.V.E11 + A.B.E00 + U.B.E10
```

and similarly for ET0TY.

Since the weights are too numerous to store between stages (1) and (3) they must be recalculated and the total number of operations per particle for CIC is about 34 compared with 7 for NGP. The disadvantage of the CIC method is that it may take 4 to 5 times longer to computer per particle than NGP. However, the CIC model is perhaps ten times less noisy and is the only method that can be used in some cases.

The area-weighting technique used in CIC is the same as a bilinear interpolation and hence we find in Fig. 3 that the resulting force law is like a linear interpolation between the step values of the NGP force law. It is clearly a much more accurate and smoother approximation to the exact r^{-1} force law between interacting line charges which is given by the dots in Fig. 3.

GRAVITATIONAL ANALOGY

The electrostatic particle model which has just been described can be applied to a gravitational problem by re-scaling. We note that the force law between interacting charges is

$$F = - \frac{q_1 q_2}{r^2}$$

and between interacting masses is

$$F = + G \frac{m_1 m_2}{r^2}$$

Hence, if we use an electrostatic model with a charge-to-mass ratio q/m equal to $\sqrt{G}$, the square root of the gravitational constant, and in addition change the sign somewhere in the loop, then we have a gravitational simulation.

There is, however, a more difficult problem associated with the absence of gravitational electrodes. It is necessary to modify the boundary conditions of the potential problem to those associated with a gravitational problem. This usually means that the gravitational potential should decay away to infinity correctly, without the potential being specified anywhere. In contrast, a typical electrostatic problem has the potential specified on a near boundary. Consequently, considerable revision is likely to be necessary in the potential solving routine; a method for doing this is described in Ref. [4].

If a two-dimensional electrostatic particle simulation is converted to a gravitational simulation, one has a simulation of infinitely-long rod-like stars. Such a two-dimensional model may be interesting in its own right and have some realism in simulating long cigar-like galaxies such as NGC 2685. However, the simulation of a thin-disk galaxy is much more interesting as it is a good model for the study of the origin and evolution of spiral structure. For this one needs a model of point (in contrast to rod) stars moving in a plane and the potential calculation must reproduce correctly the r^{-1} potential of interacting point masses (and not the log(r) interaction of line-masses). This cannot be achieved by solving Poisson's equation in two dimensions, but Fourier transform techniques are available and described in Ref. [1].

THE VORTEX ANALOGY

In an electrostatic simulation with a high external magnetic field B_0 the motion of the gyrating particles can be accounted for adequately by the guiding-centre approximation, which gives the velocity directly from the electric field

$$\vec{V} = c \frac{\vec{E} \times \vec{B}}{\vec{B}^2}$$

or, in component form,

$$V_x = \frac{c}{B_z} E_y = -\frac{c}{B_z}\frac{\partial \phi}{\partial y}$$

$$V_y = -\frac{c}{B_z} E_x = \frac{c}{B_z}\frac{\partial \phi}{\partial x}$$

where ϕ is given by

$$\nabla^2 \phi = -4\pi \rho(x, y)$$

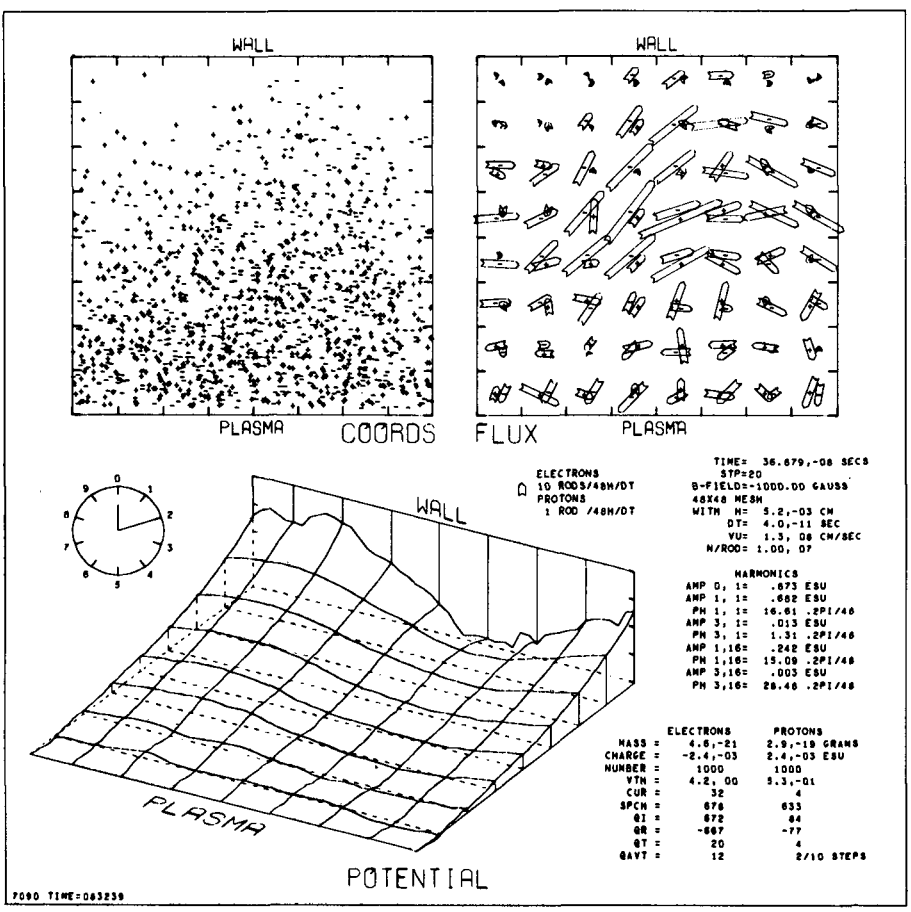


FIG. 4. A simulation of plasma flow across a strong magnetic field, showing top left – particle positions, top right – flux arrows, and bottom – electrostatic potential. [from HOCKNEY, R. W., Physics Fluids 9 (1966) 1826, A.I.P.].

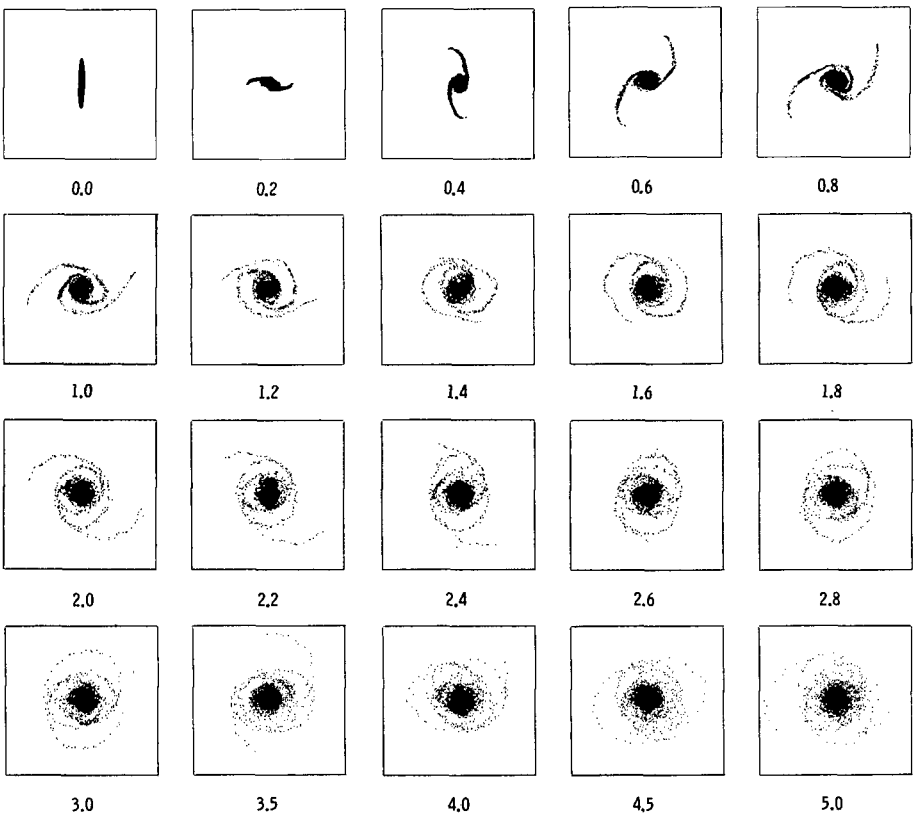


FIG. 5. The development of an elliptical bar galaxy under differential rotation. Time in units of rotations of the outer stars. [from HOCKNEY, R. W., Publ. Astronom. Soc. Pacific 80 (1968) 662].

An analogous situation arises in two-dimensional incompressible and inviscid flow. In this case the velocity is derived from a stream function ψ by

$$V_x = -\frac{\partial \psi}{\partial y}, \quad V_y = \frac{\partial \psi}{\partial x}$$

The vorticity, ξ, is then defined as the curl of the stream function. Hence

$$\xi = \text{curl } V = \frac{\partial}{\partial x} V_y - \frac{\partial}{\partial y} V_x = \nabla^2 \psi$$

Comparing these formalisms one can see that the stream function and vorticity are analogous to the electrostatic potential and charge distribution. In this analogy the moving charges must be thought of as moving elements of vorticity.

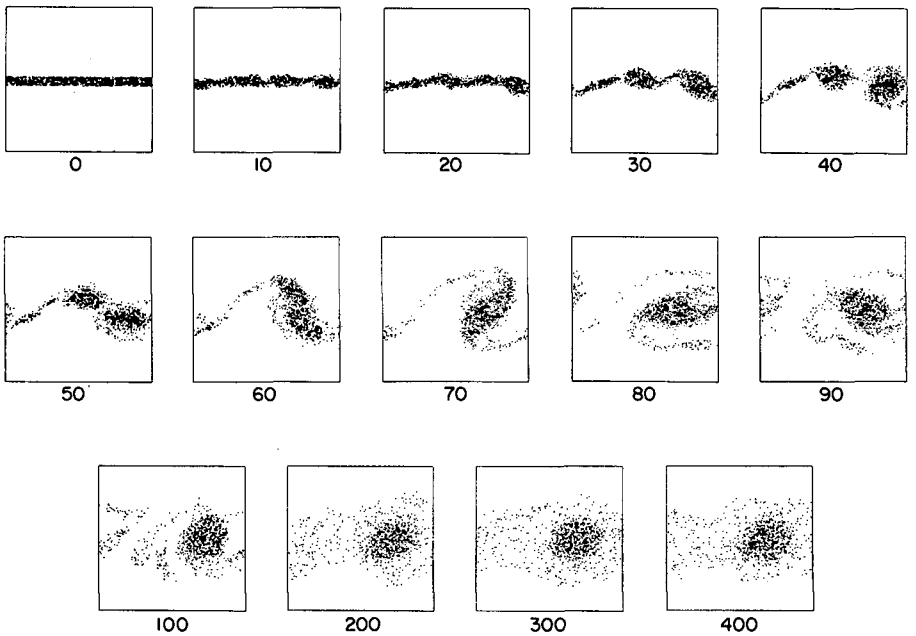


FIG. 6. The Kelvin-Helmholtz instability seen in a thin electron beam in a strong magnetic field or by analogy in the vortex layer between two fluids passing each other with equal and opposite velocities. [from LEVY, R.H., HOCKNEY, R.W., Physics Fluids 11 (1968) 766, A.I.P.].

APPLICATIONS

Anomalous diffusion

An electrostatic model with a fixed external field has been used to study the anomalously high transport of plasma across a containing magnetic field [2]. Figure 4 shows the output from such a simulation. At the top left are the positions of all the simulated particles and at the top right are flux arrows showing the flow of both ions and electrons. At the bottom is an isometric view of the potential showing the wave that gives rise to the unexpectedly high transport of particles from the plasma at the bottom to the wall at the top.

Gravity

A modified electrostatic particle model has been used to study the evolution of galaxies. Figure 5 shows still frames from a computer-made movie displaying the evolution of a bar-shaped galaxy under the influence of differential rotation, produced by the presence of a heavy central nucleus [5]. Familiar spiral shapes are observed but these are transitory and disappear after about five rotations.

Vorticity

An electrostatic particle model with guiding-centre motion has been used to study the stability of a thin beam of electrons in the interior of a containing vessel [6]. If the beam is thinner than about one fifth of the vessel, it is found to be unstable to the Kelvin-Helmholtz slipping stream instability. The nature of the instability is shown in Fig. 6 which contains still frames from a computer-generated movie. The beam is observed to wind up into vortices. The linear-growth region can be obtained by classical analysis but the computer model obtains, in addition, the non-linear saturation amplitude.

Viewed by the vortex analogy, the computer experiment of Fig. 6 can be regarded as showing the vortex layer between two fluids slipping past each other with different velocities. The vorticity is then confined to a thin layer at the junction of the fluids. In the right conditions this layer is unstable and the vorticity re-distributes itself as shown in the figure.

REFERENCES

[1] HOCKNEY, R.W., Meth. Comput. Phys. 9 (1970) 135.
[2] HOCKNEY, R.W., Physics Fluids 9 (1966) 1826.
[3] BIRDSALL, C.K., FUSS, D., J. Comput. Phys. 3 (1969) 494.
[4] HOCKNEY, R.W., Astrophys. J. 150 (1967) 797.
[5] HOCKNEY, R.W., Publ. Astronom. Soc. Pacific 80 (1968) 662.
[6] LEVY, R.H., HOCKNEY, R.W., Physics Fluids 11 (1968) 766.

PARTICLE-FIELD INTERACTIONS: NUMERICAL TECHNIQUES AND PROBLEMS

R. W. HOCKNEY
Computer Science Department,
Reading University,
Reading, United Kingdom

Abstract

PARTICLE-FIELD INTERACTIONS: NUMERICAL TECHNIQUES AND PROBLEMS.

Errors inherent in the use of a particle model in the simulation of a physical system are discussed. These arise from N, the particle number, H, the interval of the space mesh, and DT, the time-step. Measurements are presented that show that the collision rate of both the NGP and CIC models are the same and depend mainly on N. Stochastic heating which leads to poor energy conservation depends on N in the same way as the collision rate but also strongly depends on H, DT and the model used. The CIC has substantially less heating than the NGP model. Recommendations are made for the choice of H and DT.

AIM OF MODEL

In particle field simulations using the mesh method the object is usually to simulate a collisionless system. That is to say, it is desired to have the collision time of the model greater than the time of the simulated experiment. Such a simulation can be used to simulate a physical system with a very long collision time such as a 'collisionless' plasma or a galaxy of stars. Alternatively, collisional effects may be added to such a simulation in a controlled and detailed way to obtain a simulation, e.g. of a collision-dominated semi-conductor or collisional plasma.

SOURCES OF ERROR

The principal sources of error in a particle model arise from N – the number of particles which is always much less than in the physical system; H – the space mesh interval, and DT – the time-step, both of which must be finite. The effect of these errors is seen primarily in the distortion of the collision time and the introduction of stochastic heating.

COLLISION TIME

The collision time may be defined as the time, on average, for a particle to be deflected 90° from its initial direction due to collisions with other particles. In a plasma, we are not dealing with hard-billiard-ball collisions but with the accumulation of many small-angle scatterings which a test particle suffers as it bounces off the Coulomb field of other particles. Figure 1 shows some orbits of a typical particle within a simulation, and

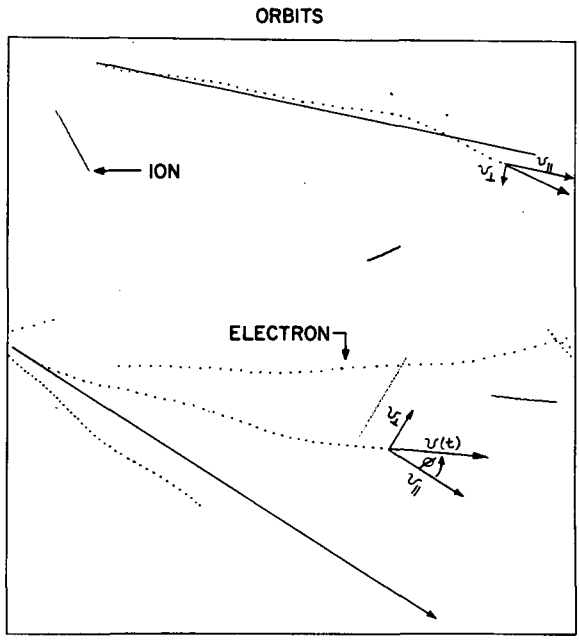


FIG. 1. Typical orbits of simulated electrons and ions in a computer model of a plasma. There is one dot per time-step in the motion. Electrons and ions can be distinguished by the difference in their velocities. [This and all other figures from J. Comput. Phys. 8 (1971) 19, Academic Press].

if $\phi_i(t)$ is the angular deflection of the i-th particle at time t from its initial direction then the average square deflection is

$$\langle \phi^2(t) \rangle = \left(\sum_{i=1}^{N} \phi_i^2(t) \right) / N$$

The collision time τ_ϕ is defined as the time for the square root of the quantity to reach 90°. Figure 2 shows the results of such a measurement made in a thermal two-dimensional computer plasma.

EFFECT OF PARTICLE NUMBER

It is sometimes difficult to see why the collision time increases as the particle number increases because one might think at first that, with more particles to collide with, this time would be decreased. But one must remember that as more particles are used to simulate the same system, less charge is associated with each particle and this rapidly decreases the collision cross-section of each particle. This effect overweighs that due to the increase in the number of collisions and the collision time is, in fact, increased.

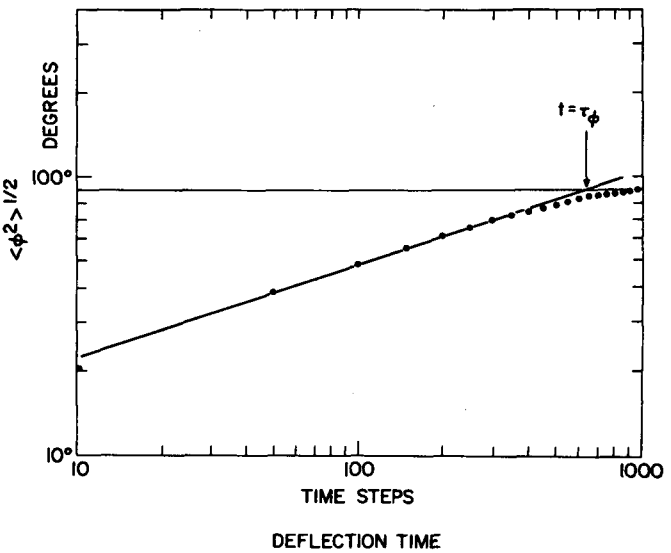


FIG.2. Typical results obtained for the measurement of deflection (or collision) time. The dots are measured values.

This may be seen from an examination of the Rutherford-Coulomb scattering cross-section:

If
- q – charge of particle,
- m – mass of particle,
- d – density of particles,
- v – average particle velocity, and
- σ – collisional cross-section

then collisions/s are given by dvσ. The Rutherford cross-section is proportional to $q^4/(m^2v^4)$. Hence the collision rate is

$$\nu \propto \frac{d\, q^4}{v^3 m^2}$$

If in the model we have 's' electrons per particle of the model then q = se, m = sm_e, d = n/s where e, m_e and n are the electronic charge, mass and physical plasma density. Then

$$\nu_{model} = \left(\frac{n}{s}\right) \frac{(s\,e)^4}{v^3(sm)^2} = s\,\frac{ne^4}{v^3m^2} = s\;\nu_{plasma} \qquad (1)$$

Hence the collision rate in the model is increased by a factor s over that of the real plasma. 's' is the ratio of number of electrons in the plasma to the number of particles in the model and is typically 10^6.

In terms of the collision time $\tau = \nu^{-1}$, and the particle number N, Eq. (1) becomes

$$\tau_{model} = \frac{\tau_{plasma}}{s} = \frac{N}{n} \tau_{plasma} \tag{2}$$

which shows the increase of collision time with particle number.

The difference between a real plasma and a computer plasma may be visualized by considering a sandy beach of 10^{15} grains of sand (corresponding to the real plasma) and a rocky beach, e.g. at Grignano, of 10^5 boulders (corresponding to the computer plasma). Both beaches contain the same mass of material but they have vastly different properties from the bather's point of view. If the bather wishes to reach the sea from his beach hut he may do so in a straight line on the sandy beach, but would be obliged to follow a circuitous route with many large deflections on the rocky beach. In the former case, one has the long collision time associated with the real plasma and in the latter case the short collision time and large deflections associated with the computer plasma.

MACROSCOPIC PROPERTIES

If a computer model exaggerates the collisional effects by many orders of magnitude, one must be concerned with the effect on other properties such as plasma frequency and Debye length.

The plasma frequency is given by

$$\omega_p^2 = 4\pi ne^2/m$$

Hence

$$(\omega_p^2)_{model} = 4\pi \left(\frac{n}{s}\right) \frac{(se)^2}{(sme)} = (\omega_{pe}^2)_{plasma} \tag{3}$$

and the plasma frequency is unaffected by the particle number used in the simulation. Since the Debye length $\lambda_D = v_{th}/\omega_{pe}$, it will be unaffected provided we make the particle velocity v the same in the model as in the real plasma. This is always done. It may similarly be shown that the amount of subdivision, s, does not affect other macroscopic quantities.

EFFECTS OF MESH SIZE

Measurements have been made of the collision time in a two-dimensional thermal plasma for both the NGP and CIC models [1]. These are found to be fitted to 20% by the relation

$$\frac{\tau_{coll}}{\tau_p} = d(\lambda_D^2 + W^2) \tag{4}$$

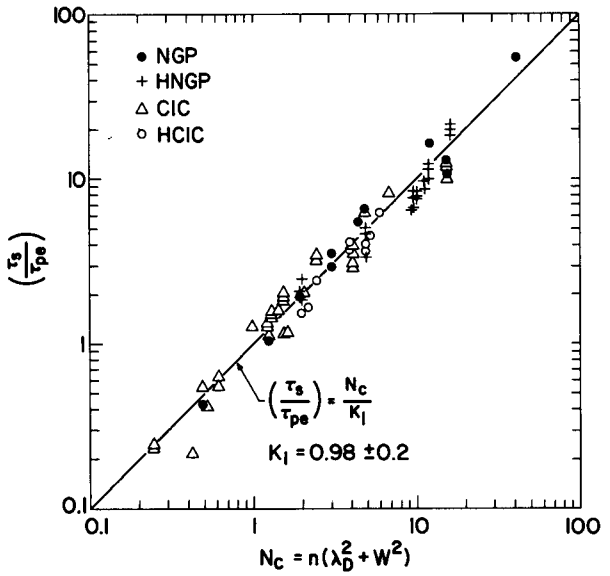


FIG.3. The dependence of collision time on particle number and width for the NGP, CIC, HNGP and HCIC models.

where d is the particle density and W the particle width. For both NGP and CIC it is found necessary to associate a particle width W = H to the mesh spacing. Figure 3 shows the results of these measurements for a wide range of conditions.

The collision time can be said to depend primarily on the number of particles used and, to a small extent — via W — on the mesh size. It is not dependent on the time-step DT and it is found to be the same, within the limit of experimental accuracy, for the NGP and CIC models. Some smoothing procedures have the effect of broadening the particle and this can affect the collision time via the particle width W.

STOCHASTIC HEATING

Errors of a random or stochastic nature occur as particles pass cell boundaries of the mesh or as field variations are ignored because of the use of a finite time-step. These errors may be regarded as being equivalent to the introduction of a stochastic error field at each time-step. This leads to a random walk in velocity space and consequent stochastic heating of the system. This in itself constitutes a loss of energy conservation.

The heating time τ_H is defined as the time for the increase in energy due to stochastic heating to equal kT/2. Since τ_H depends on the particle number in the same way as τ_{coll}, we study the ratio τ_H/τ_{coll}. This is found to be a complicated function of the time-step DT and space mesh H.

This dependence is shown in Fig. 4 for four models. These are the NGP and CIC and two others, HNGP and HCIC, obtained from them by smoothing

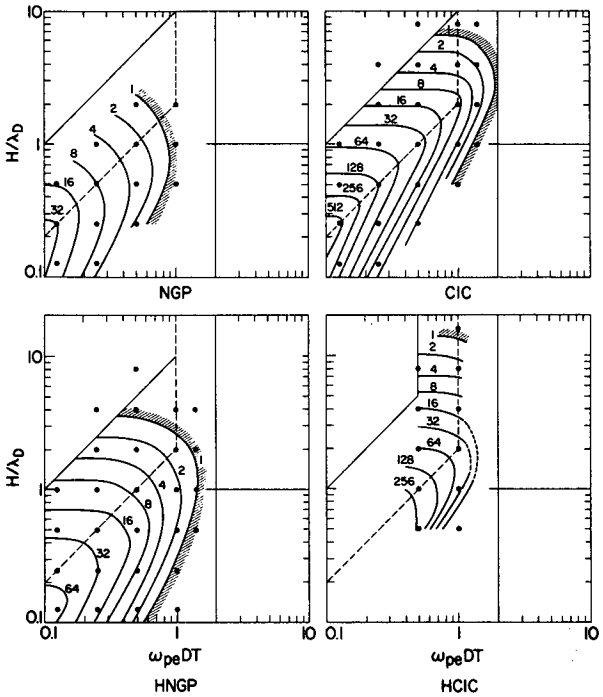


FIG. 4. Variation of heating time in the parameter plane ($\omega_{pe}DT$, H/λ_D).

the potential before use. It is desirable to keep the ratio τ_H/τ_{coll} as large as possible, and it is evident from Fig. 4 that for any value of H/λ_D there is an optimum value for $\omega_{pe}DT$. This optimum path through the parameter plane is given by

$$(\omega_{pe}\,DT)_{opt} = \min\left[\tfrac{1}{2}\frac{H}{\lambda_D},\ 1\right] \tag{5}$$

and is shown in Fig. 5 and also by the dotted line in Fig. 4.
It is evident that a time-step larger than the optimum will result in a rapid increase in stochastic heating, whilst a time-step shorter than the optimum decreases the stochastic heating very little. One has also to avoid getting too close to $\omega_{pe}DT = 2$ which represents the stability limit for the time integration scheme.

Along this optimum path the heating time is given by

$$\frac{\tau_H}{\tau_{coll}} = \frac{K_4}{(H/\lambda_D)^2} \tag{6}$$

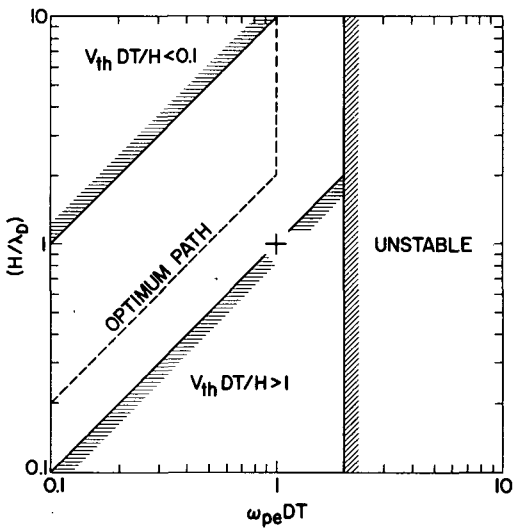


FIG. 5. Regions of the parameter plane (ω_{pe}DT, H/λ_D) and the optimum path.

where

$$K_4 = 2 \quad \text{NGP}$$
$$= 41 \quad \text{CIC}$$
$$= 6.4 \quad \text{HNGP}$$
$$= 200 \quad \text{HCIC}$$

The heating time is then very strongly dependent on the model used as well as on the size of the space- and time-steps.

CHOICE OF MODEL

The results obtained in the previous sections enable us to choose rationally between the available models for any particular application. The basis of the choice will be the cost in computer time of simulating a collisionless plasma for a given number of plasma periods using a given number of mesh points. The cost will be assessed on the basis of the number of arithmetic operations involved, and it will be assumed that a plasma may be regarded as collisionless up to a time equal to the collision time of the model.

The number of particles used must be selected on the basis of the number of collisionless plasma periods P that are required using Eq. (4),

$$P = (\tau_{coll}/\tau_{pe}) = d(\lambda_D^2 + W^2)/K_1 \qquad (7)$$

Then the number of particles necessary in a region of size L cm × L cm is

$$N = d\,L^2 = K_1\,P\,L^2/(\lambda_D^2 + W^2)$$

The time-step used will be chosen from the optimum path on the basis of the value of H/λ_D using Eq. (5)

$$(\omega_{pe}\,DT)_{opt} = \min\left[(1/2)H/\lambda_D,\ 1\right] \tag{8}$$

The number S of steps required is then given by

$$S = \frac{P\,\tau_{pe}}{DT_{opt}} = \frac{2\pi P}{(\omega_{pe}\,DT)_{opt}} \tag{9}$$

If we let K_5 (model) be the cost of computing for a given model per step per particle then the total cost is

$$\begin{aligned} C &= K_5\,S\,N \\ &= K_5\,\frac{2\pi P}{(\omega_{pe}\,DT)_{opt}}\ \ \frac{K_1 P\,L^2}{(\lambda_D^2 + W^2)} \\ &= \frac{2\pi\,K_5 K_1\,P^2\,(L/H)^2}{(\omega_{pe}\,DT)_{opt}\ \ ((\lambda_D/H)^2 + (W/H)^2)}\ . \end{aligned} \tag{10}$$

We note that the cost is proportional to the square of the number of collisionless plasma periods required and to the number of mesh cells. Hence we compare the cost per square collisionless plasma period per mesh cell which is

$$C/(PL/H)^2 = \frac{2\pi\,K_5\,K_1}{\min\left[\tfrac{1}{2}H/\lambda_D,\ 1\right]\ ((\lambda_D/H)^2 + (W/H)^2)} \tag{11}$$

This function is plotted in relative units for the different models as a function of (H/λ_D) in Fig. 6. On the basis of the number of computer operations we have taken K_5(CIC or HCIC) = 5 K_5 (NGP or HNGP). Because of this factor, the NGP and HNGP models are always cheaper if they can be used. However, the noise in these models prevents their being used for large values of (H/λ_D). If we assume that a model can be used provided the heating to collision time ratio (τ_H/τ_{coll}) is greater than 10 (and therefore the total energy conservation better than 2.5%) then the models may only be used on the solid parts of the curves. If in addition we only use smoothing when the unsmoothed model is too noisy, in order to keep as much spatial

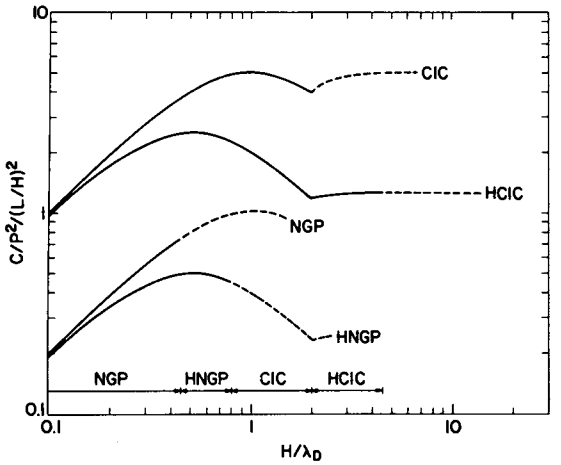


FIG. 6. The cost per square collisionless plasma period per mesh cell in relative units for the NGP, CIC, HNGP and HCIC models.

resolution as possible, then we find that there is a favoured zone of (H/λ_D) where each model is cheapest. This is given by the tableau

$$0 \leftarrow \text{NGP} \rightarrow 0.45 \leftarrow \text{HNGP} \rightarrow 0.8 \leftarrow \text{CIC} \rightarrow 2.0 \leftarrow \text{HCIC} \rightarrow 4.5$$

where the name of the model is written between the values of H/λ_D for which it is best suited.

If one is less conservative and allows computing up to a $(\tau_H/\tau_{coll}) = 1$, equivalent to a total energy conservation of 25%, then the dotted parts of the curves of Fig. 6 may be used and the favoured ranges of (H/λ_D) for the different models are

$$0 \leftarrow \text{NGP} \rightarrow 1.5 \leftarrow \text{HNGP} \rightarrow 2.5 \leftarrow \text{CIC} \rightarrow 6.5 \leftarrow \text{HCIC} \rightarrow 14$$

Under these circumstances, none of the models considered here can be used for $H/\lambda_D > 14$.

REFERENCE

[1] HOCKNEY, R.W., J. Comput. Phys. 8 (1971) 19.

THE SOLUTION OF POISSON'S EQUATION

R.W. HOCKNEY
Computer Science Department,
Reading University,
Reading, United Kingdom

Abstract

THE SOLUTION OF POISSON'S EQUATION.

Iterative methods for the solution of Poisson's equation are considered first and the convergence rates of the SOR, Gauss-Seidel and Chebyshev methods are compared. Direct methods, based on Fourier analysis and cyclic reduction are then discussed, and finally measurements are given of the execution times for Poisson solvers using a variety of computers and compilers.

In many areas of computational physics, particularly in time-dependent simulations, a central problem is the rapid solution of the field equations. In the past few years a variety of special direct methods have been developed which can be used in simple geometries, and, in this case, are very much superior to the more commonly known iterative methods. In this paper, we shall confine our attention to the solution of one equation, namely that of Poisson:

$$\nabla^2 \phi = -4\pi\rho(x, y) \tag{1}$$

ITERATIVE METHODS

Before proceeding to a discussion of the special direct methods, we start by quoting some convergence results for the most commonly used iterative methods, in order to demonstrate how bad the convergence can be.

We consider the solution of Poisson's equation in the square with zero values for the potential on the boundary. The five-point difference approximation is used

$$\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1} - 4\phi_{i,j} = -4\pi\rho_{i,j}H^2 = q_{i,j} \tag{2}$$

We consider three iterative methods:

(a) Gauss-Seidel method

Equation (2) is solved for $\phi_{i,j}$ and this formula is used to update mesh values in some ordered sequence, e.g. line by line. Freshly computed values immediately overwrite old values on the mesh and the hence latest values are always used on the right-hand side

$$\phi^{new}_{i,j} = \phi'_{i,j} = \frac{1}{4}[\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1} - q_{i,j}] \qquad \text{latest values} \tag{3}$$

(b) Successive over-relaxation (SOR)

In this method, the new value is calculated as an average of the ϕ' used by Gauss-Seidel and the old value. A constant ω, the relaxation factor, determines the weighting:

$$\phi_{ij}^{new} = \omega\phi'_{ij} + (1 - \omega)\phi_{ij}^{old} \tag{4}$$

$$1 \leq \omega \leq 2$$

For the square a formula exists for ω_b, the best value of the factor ω.

(c) Chebyshev method

A variant of SOR in which ω varies every half iteration has better convergence properties than SOR. This is the Chebyshev method in which the iteration is divided into two parts. The first half is the adjustment of all odd points (i.e. those for which i + j is odd) on the mesh by Eq. (4), the second half iteration is the similar adjustment over the remaining even points.

The relaxation factor varies at each half iteration according to

$$\omega^{(0)} = 1$$

$$\omega^{(\frac{1}{2})} = 1\Big/\left(1 - \frac{1}{2}\mu^2\right)$$

$$\omega^{(t+\frac{1}{2})} = 1\Big/\left(1 - \frac{1}{4}\mu^2\omega^{(t)}\right) \qquad t = \frac{1}{2},\ 1,\ \frac{3}{2}\ldots$$

where $\mu = \cos \pi/n$ for an $n \times n$ mesh. It can be shown that $\omega^{(\infty)} = \omega_b$, so that the Chebyshev method starts with a half iteration of Gauss-Seidel (when $\omega = 1$) and then smoothly varies ω until one is performing SOR with $\omega = \omega_b$.

CONVERGENCE RATES

If $\phi^*_{i,j}$ is the exact solution to the Poisson problem then we define the error vector at the t-th iteration to be

$$\epsilon^{(t)}_{i,j} = \phi_{i,j} - \phi^*_{i,j} \tag{5}$$

and the norm of the error vector to be

$$||\epsilon^{(t)}|| = \left(\frac{\sum \epsilon^2_{i,j}}{n^2}\right)^{\frac{1}{2}} \tag{6}$$

All the above iterative processes are linear and the t-th iterate of the error can be related by a matrix $M^{(t)}$ to the initial error $\epsilon^{(0)}$,

$$\epsilon^{(t)} = M^{(t)}\epsilon^{(0)} \tag{7}$$

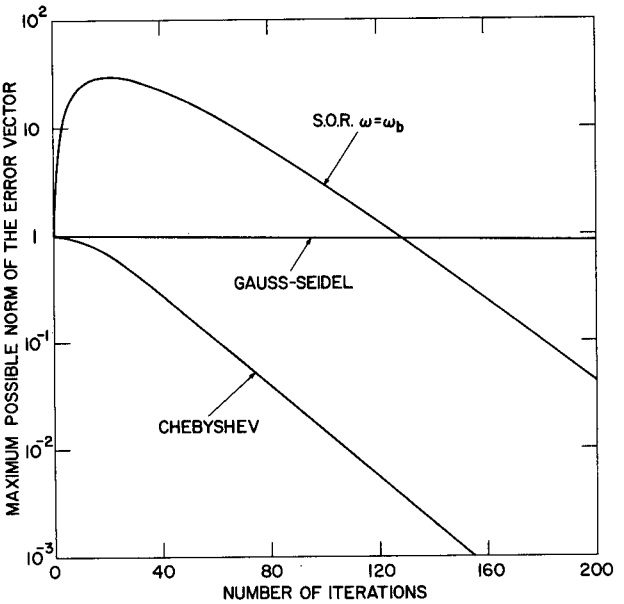


FIG.1. Theoretical error bounds for the variation of the maximum possible norm of the error vector (relative to its initial value) with the number of iterations, for SOR, Gauss-Seidel and Chebyshev methods on a 128 x 128 mesh using Odd/Even ordering. [from Meth. Comput. Phys. 9 (1970) 166, Academic Press].

Taking norms and dividing we have

$$\frac{||\epsilon^{(t)}||}{||\epsilon^{(0)}||} \leq ||M^{(t)}|| \qquad (8)$$

Hence $||M^{(t)}||$ is an upper bound on the factor by which the initial error is reduced in t iterations. Fortunately, for the Poisson problem in the square, $||M^{(t)}||$ can be calculated by analysis for all the above iterative methods, and the results are plotted in Fig. 1 for a 128 x 128 mesh.

From this figure it can be seen that the convergence of the Gauss-Seidel method can be so slow that the method is useless. The SOR method, whilst it gives a reasonable convergence rate for large t, can give unexpectedly bad results for small numbers of iterations. In fact, it is possible for the error to increase in size by 30 times in the first ten iterations. The superiority of the Chebyshev method is seen in that it overcomes this problem with SOR and, as may be proved analytically, $||M^{(t)}||$ is a monotonically decreasing function of t. Even with the Chebyshev method it can be seen that about n iterations are required to guarantee an error reduction by a factor of 10^{-2} to 10^{-3}.

We should like it to be clear what the above theoretical results mean. They do not mean that for any particular case the error decay will follow the curve of Fig. 1 but only that no error decay curve met in practice can rise above the lines in the figure. The curves are therefore a worst-case

result. The error bound is sharp, however, because of the equality in Eq. (8). This means that there exists an initial error vector which would give an error decay curve that will touch any chosen point on the curves.

We hope this brief discussion of the most commonly used iterative methods and convergence results will motivate the following discussion of direct methods. These are methods that solve the difference equations (2) in a known finite number of arithmetic operations that are equivalent in work to about 5 to 10 SOR iterations. It is clear that within this time one can guarantee no significant reduction of the error by any of the iterative methods.

DIRECT METHODS

A direct method of solution is one in which a single solution is obtained after a fixed finite number of arithmetic operations which is accurate to within, say, 10 times the rounding error. This contrasts with an iterative method in which many approximate solutions are found which, if all goes well, converge gradually towards the solution to the problem.

If we let $\vec{\phi}_j$ be the vector of unknown values on the j-th line, then the Poisson problem can be expressed in matrix form as:

$$\vec{\phi}_{j-1} + A\vec{\phi}_j + \vec{\phi}_{j+1} = \vec{q}_j \qquad j = 1, 2, \ldots, n-1$$

with $\vec{\phi}_0 = \vec{\phi}_n = 0$ and

$$A = \begin{bmatrix} -4 & 1 & 0 & \cdots & 0 \\ 1 & & & & 0 \\ & & & & \\ 0 & & & & 1 \\ 0 & \cdots & 0 & 1 & -4 \end{bmatrix} \tag{9}$$

The special form of these equations enables the variables on every other line of the mesh to be eliminated. This process which we call odd/even (or cyclic) reduction is fundamental to many direct methods. Let us take three neighbouring equations for the j - 1, j and j + 1 lines:

$$\begin{aligned} \vec{\phi}_{j-2} + A\vec{\phi}_{j-1} + \vec{\phi}_j &= \vec{q}_{j-1} \\ \vec{\phi}_{j-1} + A\vec{\phi}_j + \vec{\phi}_{j+1} &= \vec{q}_j \\ \vec{\phi}_j + A\vec{\phi}_{j+1} + \vec{\phi}_{j+2} &= \vec{q}_{j+1} \end{aligned} \tag{10}$$

All reference to the odd lines j - 1 and j + 1 (considering j to be even) is eliminated if we multiply the central equation by the matrix -A and add. We then obtain

$$\vec{\phi}_{j-2} + (2I - A^2)\vec{\phi}_j + \vec{\phi}_{j+2} = \vec{q}_{j-1} + \vec{q}_{j+1} - A\vec{q}_j \tag{11}$$

If now we define new values for $A^{(1)}$ and $q^{(1)}$ appropriate to the first level of reduction as

$$A^{(1)} = 2I - A^2, \quad \vec{q}_j^{(1)} = \vec{q}_{j-1} + \vec{q}_{j+1} - A\vec{q}_j \tag{12}$$

then the reduced equations are

$$\vec{\phi}_{j-2} + A^{(1)}\vec{\varphi}_j + \vec{\phi}_{j+2} = \vec{q}_j^{(1)} \tag{13}$$

$$j = 2, 4, 6, \ldots, n-2$$

The reduced equations on the even lines are n/2 in number. They are also more complex since $A^{(1)}$, being a product of two tridiagonal matrices, is five-diagonal in form. However, as regards the dependence on j, they are in the same form as the original Eqs (10). Hence the process of reduction can be repeated to obtain equations on every fourth line with A and q defined by

$$A^{(2)} = 2I - (A^{(1)})^2 \text{ is now nine-diagonal}$$

$$\vec{q}^{(2)} = \vec{q}_{j-1}^{(1)} + \vec{q}_{j+1}^{(1)} - A^{(1)}\vec{q}_j \tag{14}$$

A whole family of direct methods can now be envisaged depending on how often the reduction is repeated and how the reduced equations are finally solved.

Solution of reduced equations

The reduced equations are eminently suited to solution by Fourier analysis using the techniques of Cooley, Tukey [1] and others which are now generally referred to as the FFT (Fast Fourier Transform). To classify as such a transform, the number of operations needed to compute all the n harmonic amplitudes from n data values must be proportional to $n \log_2 n$. This is to be compared with the n^2 operations required if the harmonics are computed by the defining series. When n is large (say > 100) the saving in computer time is dramatic. It is probably fair to say that any direct algorithm that uses an n^2 Fourier analysis calculation is throwing away, quite unnecessarily, most of the potential advantage that the method has.

Solution by Fourier analysis of the reduced equations like (13) is worthwhile only because sines and cosines are the eigenvectors of the rather special operator A, and also of the derived operators $A^{(1)}$, $A^{(2)}$, etc. It is because of this that the resulting equations for the harmonic amplitudes are uncoupled, and the equations for each harmonic amplitude may be solved independently.

On Fourier analysis of the reduced equations (like 13) one obtains equations for the harmonic amplitudes $\bar{\phi}^k$ and $\bar{q}^k$:

$$\bar{\phi}_{j-2}^k + \lambda\bar{\phi}_j^k + \bar{\phi}_{j+2}^k = \bar{q}_j^k \tag{15}$$

where we can see that the complicated matrix $A^{(1)}$ has been reduced by Fourier analysis to a scalar quantity λ. The indexing in Eq. (15) has been given for one level of reduction, but the above remarks apply, with appropriate indices, to an analysis performed at any level.

The harmonic equations (15) are a tridiagonal system and can be solved by any convenient means. We recommend the method of cyclic reduction [2] for periodic conditions in y and Gauss elimination [3] for other conditions (and necessarily if λ in a more difficult problem depends on j).

Solution of the tridiagonal system (15) gives the harmonics of the potential on the specified lines. A Fourier synthesis (of course using an FFT algorithm) gives the potential values on these lines.

The remaining lines

Having obtained the solution ϕ on some lines, the remaining lines may be solved for using the appropriate reduced equation. Supposing we had taken two levels of reduction then we would know the potential on every fourth line. The solution on every second line can then be obtained by using the reduced equations of the first level (Eq. (13)), on every even line, in the form

$$A^{(1)} \vec{\phi}_j = \vec{q}_j^{(1)} - \vec{\phi}_{j-2} - \vec{\phi}_{j+2} \qquad j \text{ even} \qquad (16)$$

Since j is even, φ_{j-2} and φ_{j+2} are known values from every fourth line. Equations (16) are solved by any suitable method, e. g. Gauss-elimination or cyclic reduction, and it is important to note that all the matrices $A^{(\ell)}$ can be expressed as products of tridiagonal matrices. For example,

$$A^{(1)} = 2I - A^2 = (\sqrt{2I} - A)(\sqrt{2I} + A) \qquad (17)$$

Hence the solution of equations like Eq. (16) can be performed by successive application of a routine for solving tridiagonal systems. This is clearly better than multiplying out the A^2 and solving the five-diagonal system $A^{(1)}$ directly. Similar considerations apply to any level of reduction ℓ.

To return to our example, the solution of Eq. (16) yields the solution on all even lines. The solution on the odd lines is then obtained similarly from the original equations:

$$A\vec{\phi}_j = \vec{q}_j - \vec{q}_{j-1} - \vec{q}_{j+1} \qquad (18)$$

The FACR(ℓ) method

The method just described may be summarized:

(a) reduce equations to level 'ℓ'
(b) solve equations of level 'ℓ' by Fourier analysis getting solutions on every (2^ℓ)-th line.
(c) expand getting solution of intermediate lines.

This method has been referred to variously as the FACR(ℓ) [4] method and CORF [5] algorithm. If FFT is used, the number of operations to solve the Poisson equation on an n × n mesh is, for a particular implementation,

$$n^2[2 + 4.5\ell + (5 \log_2 n - 4)/2^\ell] \qquad (19)$$

This function is plotted in Fig. 2 as a function of ℓ for a 128 × 128 mesh.

A clear minimum in the number of operations is seen at two levels of reduction and there is little gain in two levels as compared to one. The

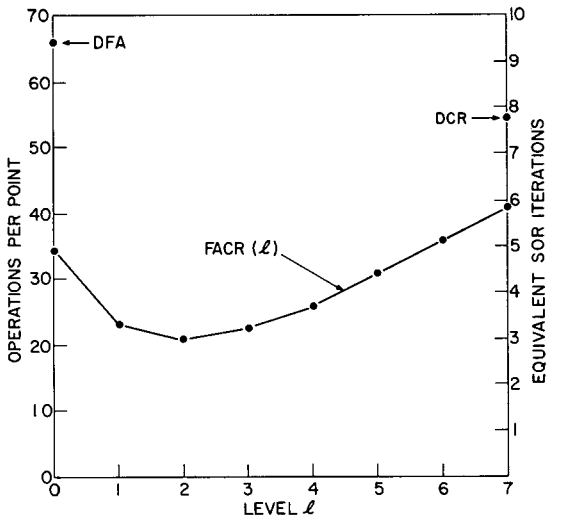


FIG.2. The number of operations per mesh point for the FACR(ℓ), DFA and DCR algorithms on a 128 x 128 mesh. [from Meth. Comput. Phys. 9 (1970) 161, Academic Press].

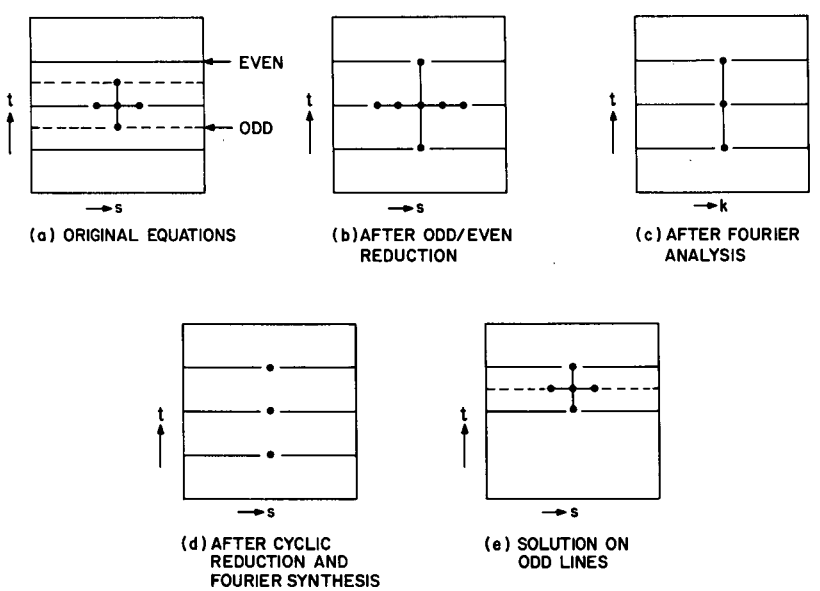


FIG.3. The different stages of the FACR(ℓ) algorithm. The circles show the positions of the variables related at each stage; even-lines are shown as solid lines and odd-lines are shown as dashed lines. [from Meth. Comput. Phys. 9 (1970) 149].

TABLE I. MEASURED EXECUTION TIMES (s) FOR 3 POISSON-SOLVING PROGRAMS ON A VARIETY OF COMPUTERS. THE STAR INDICATES THE FASTEST PROGRAM IN EACH CASE. 128 × 128 MESH IS USED EXCEPT WHERE SPECIFIED

ALGORITHM	FACR(ℓ)	Modified FACR ($\log_2 n - 1$)		
COMPUTER	POT1 R. Hockney	XYPOIS O. Buneman	ODDEVN A. George	COMPILER
IBM 360/91	0.50	0.99	0.35*	FORTRAN H[b] opt = 2
CDC 6600	0.75 1.93 2.78*	- 1.64* 3.24	- 2.79 5.79	ASSEMBLY FTN[b] RUN[c]
IBM 360/75	2.56* 3.74*	3.47 5.66	3.18 5.84	FORTH opt = 2[b] FORTRAN G[c]
360/67	3.53*	6.15	-	FORTH opt = 2
370/155	4.90 6.50	- -	- -	H opt = 2[b] G[c]
IBM 7090	0.37[a]	-	-	32 × 32 ASSEMBLY
ICL 4130	7.0	5.26	3.48*	32 × 32 FORTRAN[c]

[a] Scaled from measured value of 0.832 on 48 × 48 mesh.
[b] These compilers claim to optimize object code.
[c] Compilers with no claim to optimization.

equivalent number of SOR iterations is given on the right-hand side based on there being seven operations per point in the SOR calculation. All the direct methods considered get the solution comprise less than 10 SOR iterations.

Buzbee et al. [5] have shown that the FACR(ℓ) algorithm rapidly loses accuracy as ℓ is increased. Since the algorithm becomes slower if $\ell > 2$ and there is not much speed advantage in $\ell = 2$ over $\ell = 1$, the accuracy consideration leads one to favour an $\ell = 1$ algorithm. Such an algorithm has been in successful use for many years [2] and a recent version POT1 is well documented [4, 6]. The relation between quantities during the FACR(ℓ) algorithm is shown in Fig. 3.

BUNEMAN ALGORITHM

An attractive feature of the reduction process is that if $\ell = \log_2 n - 1$ then the problem is reduced to the solution of a single equation for the central line of the mesh. This may be solved by repeated application of a

tridiagonal equation solver without the need for Fourier analysis. Unfortunately, as already mentioned, the method is numerically inaccurate and cannot be used. Buneman, however, has rephrased the algorithm slightly and, at the expense of introducing an extra arithmetic operation, made the method accurate. This method is called DCR in Fig. 2 and two implementations have been compared by the author. They are XYPOIS [7] written by Buneman and ODDEVN written by Alan George, both at Stanford University.

COMPARISON OF PROGRAMS

The programs POT 1, XYPOIS, ODDEVN have been run and accurately timed (using a CPU timer) on a variety of computers using a variety of compilers. The results are interesting and are shown in Table I.

We see that the computer time depends strongly on the particular computer and compiler used, sometimes more than on the algorithm itself. We find, for example, that ODDEVN is three times faster than XYPOIS on the 360/91 but that XYPOIS is almost twice as fast as ODDEVN on the 6600. Also we find that the algorithm with the minimum number of arithmetic operations (POT1) is not always the quickest in execution (see 360/91, 4130 & 6600 under FTN). The loss of efficiency when using FORTRAN as opposed to Assembly code even with an optimizing compiler is quite striking.

OTHER GEOMETRIES

The methods described above depend heavily for their speed on the problem having simple enough geometry for sines and cosines to be the eigenfunctions of the operator A. A curved rather than rectangular boundary or the presence of electrodes in the interior destroy this fact. However, extensions to the methods have been developed that allow the inclusion of a number of interior and surface electrodes and cover some cases of mixed boundary conditions [8].

REFERENCES

[1] COOLEY, J.W., TUKEY, J.W., Math. Comput. 19 (1965) 297.
[2] HOCKNEY, R.W., J. Ass. comput. Mach. 12 (1965) 95.
[3] VARGA, R.S., Matrix Iterative Analysis, Prentice Hall, Englewood Cliffs, New Jersey (1962).
[4] HOCKNEY, R.W., Meth. comput. Phys. 9 (1970) 135.
[5] BUZBEE, B.L., GOLUB, G.H., NIELSON, C.W., Method of Odd/Even reduction and factorisation with application to Poissons equation. Technical reports CS128 (1969) and STAN-CS-70-155 (1970), Stanford University, Stanford, California.
[6] CHRISTIANSEN, J., HOCKNEY, R.W., Comput. Phys. Communs 2 (1971) 139.
[7] BUNEMAN, O., Technical Report SUIPR 294 (1969), Institute for Plasma Research, Stanford University, Stanford, California.
[8] HOCKNEY, R.W., P0T4 – a fast Direct Poisson Solver for the rectangle allowing some mixed boundary conditions and internal electrodes, IBM Research Report RC2870 (May 1970), IBM Research Laboratory, Yorktown Heights, New York.

DIFFERENCE METHODS IN FLUID DYNAMICS, WITH APPLICATIONS

J. KILLEEN
University of California,
Lawrence Livermore Laboratory,
Livermore, Calif.
and
Department of Applied Science,
Davis, Calif.,
United States of America

Abstract

DIFFERENCE METHODS IN FLUID DYNAMICS, WITH APPLICATIONS.

Finite-difference methods for solving the differential equations of fluid dynamics are described. Time-dependent, initial-value problems in one and two spatial dimensions are considered. Also discussed are the applications of these methods to problems in meteorology and magnetohydrodynamics.

1. INTRODUCTION

In this paper, computational methods for solving the differential equations of fluid dynamics are discussed. Time-dependent, initial-value problems in one and two spatial dimensions are considered.

A general introduction to this subject is given in Chapters 12-13 of Ref. [1]. Applications of these methods to important practical problems of hydrodynamics are found in Refs [2, 3], and applications to problems in plasma physics are found in Ref. [4]. These last three references are volumes in the series of books published annually on Methods in Computational Physics. In addition, recent work in the field of computational fluid dynamics can be found in the Journal of Computational Physics published by Academic Press every two months.

Section 2 presents the Eulerian and Lagrangian forms of the differential equations to be solved. In Section 3, the main topic of this paper is considered – the solution of the equations by finite-difference methods. The stability of these difference schemes is considered, but the methods for determining stability are not given in detail as they are the subject of papers SMR-9/14a, SMR-9/14b and SMR-9/14c in these Proceedings.

Section 4 deals with the addition of diffusion terms to the differential equations and discusses difference methods appropriate for such problems. In this last section, some applications of the methods to more complicated systems of equations which arise in meteorology and magnetohydrodynamics are also studied.

2. EQUATIONS OF FLUID DYNAMICS

In this section, we shall give the differential equations of fluid dynamics neglecting dissipative effects such as viscosity and thermal conduction.

There are two descriptions used in fluid dynamics – the Eulerian, in which a fixed co-ordinate system is used for the spatial variables, and the Lagrangian, in which a co-ordinate system is employed which moves with the fluid.

2.1. Eulerian form

Let $\vec{r}$ denote the position vector. The fluid is characterized by the density, $\rho(\vec{r},t)$, the pressure, $p(\vec{r},t)$, the specific internal energy, $\epsilon(\vec{r},t)$, and the fluid velocity, $\vec{u}(\vec{r},t)$. The conservation of matter is expressed by the following equation

$$\left(\frac{\partial}{\partial t} + \vec{u}\cdot\nabla\right)\rho = -\rho\nabla\cdot\vec{u} \tag{1}$$

The equation of motion is

$$\rho\left(\frac{\partial}{\partial t} + \vec{u}\cdot\nabla\right)\vec{u} = -\nabla p \tag{2}$$

The energy equation is

$$\rho\left(\frac{\partial}{\partial t} + \vec{u}\cdot\nabla\right)\epsilon = -p\nabla\cdot\vec{u} \tag{3}$$

In one dimension the above equations are

$$\left(\frac{\partial}{\partial t} + u\frac{\partial}{\partial r}\right)\rho = -\rho\,\frac{1}{r^{\alpha-1}}\,\frac{\partial}{\partial r}\,(r^{\alpha-1}u) \tag{4}$$

$$\rho\left(\frac{\partial}{\partial t} + u\frac{\partial}{\partial r}\right)u = -\frac{\partial p}{\partial r} \tag{5}$$

$$\rho\left(\frac{\partial}{\partial t} + u\frac{\partial}{\partial r}\right)\epsilon = -p\,\frac{1}{r^{\alpha-1}}\,\frac{\partial}{\partial r}(r^{\alpha-1}u) \tag{6}$$

where for slab symmetry, $r = x$, $\alpha = 1$, for cylindrical symmetry, $r = (x^2+y^2)^{1/2}$, $\alpha = 2$, and for spherical symmetry,

$$r = (x^2+y^2+z^2)^{\frac{1}{2}}, \qquad \alpha = 3$$

In two-dimensional Cartesian co-ordinates we have

$$\rho_t + u\rho_x + v\rho_y = -\rho(u_x + v_y) \tag{7}$$

$$\rho(u_t + uu_x + vu_y) = -p_x \tag{8}$$

$$\rho(v_t + uv_x + vv_y) = -p_y \tag{9}$$

$$\rho(\epsilon_t + u\epsilon_x + v\epsilon_y) = -p(u_x + v_y) \tag{10}$$

where u and v are x and y components of the velocity and we have used the subscript notation for derivatives. The above form of the equations is called the advective form. We could also write the above equations in conservation form, e.g. we can write Eq. (7) as

$$\rho_t + (\rho u)_x + (\rho v)_y = 0 \tag{11}$$

Although Eqs (7) and (11) are equivalent, when we consider their difference approximations later we shall see that it is sometimes advantageous to use the conservation form.

We can also consider the energy equation in a different form. From Eq. (1) we have

$$\nabla \cdot \vec{u} = -\frac{1}{\rho}\frac{d\rho}{dt}$$

where $\frac{d}{dt} = \left(\frac{\partial}{\partial t} + \vec{u}\cdot\nabla\right)$. We can then write Eq. (3) as

$$\frac{d\epsilon}{dt} = \frac{p}{\rho^2}\frac{d\rho}{dt} \tag{12}$$

and if we define the specific volume $V = 1/\rho$, then we have

$$\frac{d\epsilon}{dt} = -p\frac{dV}{dt} \tag{13}$$

If we compare this with the relation

$$\frac{d\epsilon}{dt} = -p\frac{dV}{dt} + T\frac{ds}{dt} \tag{14}$$

we have $ds/dt = 0$, hence if we are considering fluid flow satisfying Eq. (3) or (13) then the entropy, s, is constant in time.

2.2. Equation of state, sound speed

To solve a fluid dynamic problem, we need a fourth equation which is a relation between the pressure, energy, and density or specific volume. We shall assume that the thermodynamic properties of the fluid are described by an equation of the form

$$p = P(\epsilon, V) \tag{15}$$

which is called the equation of state.

When performing a calculation we need to know the sound speed in the fluid as a function of space and time, i.e.

$$c = \left(\frac{\partial p}{\partial \rho}\right)_s^{\frac{1}{2}} = \left[-V^2\left(\frac{\partial p}{\partial V}\right)_s\right]^{\frac{1}{2}} \tag{16}$$

We have

$$dp = \left(\frac{\partial p}{\partial \epsilon}\right)_V d\epsilon + \left(\frac{\partial p}{\partial V}\right)_\epsilon dV$$

where

$$d\epsilon = \left(\frac{\partial \epsilon}{\partial s}\right)_V ds + \left(\frac{\partial \epsilon}{\partial V}\right)_s dV = Tds - pdV$$

so

$$dp = \left(\frac{\partial p}{\partial \epsilon}\right)_V Tds + \left[\left(\frac{\partial p}{\partial V}\right)_\epsilon - p\left(\frac{\partial p}{\partial \epsilon}\right)_V\right] dV$$

But since entropy is constant, i.e. $ds = 0$, we have

$$c = V\left[p\left(\frac{\partial p}{\partial \epsilon}\right)_V - \left(\frac{\partial p}{\partial V}\right)_\epsilon\right]^{\frac{1}{2}} \tag{17}$$

As an illustration, consider the ideal gas law

$$p = \frac{(\gamma - 1)\epsilon}{V}$$

then

$$c = (\gamma pV)^{\frac{1}{2}} = (\gamma p/\rho)^{\frac{1}{2}} \tag{18}$$

2.3. Lagrangian form

We shall derive the Lagrangian form of the equations in two dimensions. Consider a particle at (a, b) at $t = 0$; at a later time, its co-ordinates will be

$$x = x(a, b, t)$$

$$y = y(a, b, t)$$

The components of the velocity of a particle are

$$u(a, b, t) = (x_t)a, b \text{ constant}$$

$$v(a, b, t) = (y_t)a, b \text{ constant}$$

We define the Jacobian of the variables x and y as functions of the real variables a, b

$$J = \begin{vmatrix} x_a & y_a \\ x_b & y_b \end{vmatrix}$$

By the rules of implicit differentiation

$$a_x = J^{-1} y_b \qquad b_x = -J^{-1} y_a$$

$$a_y = -J^{-1} x_b \qquad b_y = J^{-1} x_a$$

In the x-direction the equation of motion is

$$x_{tt} = -\frac{1}{\rho} p_x$$

$$= -\frac{1}{\rho} (p_a a_x + p_b b_x)$$

$$= -\frac{1}{\rho J} (p_a y_b - p_b y_a)$$

and in the y-direction we have

$$y_{tt} = -\frac{1}{\rho} (p_a a_y + p_b b_y)$$

$$= -\frac{1}{\rho J} (-p_a x_b + p_b x_a)$$

In the continuity equation we have

$$u_x + v_y = u_a a_x + u_b b_x + v_a a_y + v_b b_y$$

$$= \frac{1}{J} (u_a y_b - u_b y_a - v_a x_b + v_b x_a)$$

Now,

$$J = x_a y_b - y_a x_b$$

and

$$J_t = u_a y_b + x_a v_b - v_a x_b - y_a u_b$$

Therefore,

$$\operatorname{div} \vec{u} = \frac{1}{J} J_t$$

The continuity equation is then

$$\rho_t + \frac{\rho}{J} J_t = 0$$

or

$$\frac{\rho_t}{\rho} + \frac{J_t}{J} = 0$$

Integrating we get $\log \rho J = \text{constant}$, or

$$\rho J = \rho_0(a, b)$$

The energy equation is

$$\epsilon_t + pV_t = 0$$

We summarize the Lagrangian equations in two dimensions

$$x_t = u \qquad y_t = v \tag{19}$$

$$u_t = -\frac{1}{\rho_0}[p_a y_b - p_b y_a] \tag{20}$$

$$v_t = -\frac{1}{\rho_0}[p_b x_a - p_a x_b] \tag{21}$$

$$\rho J = \rho_0 \tag{22}$$

$$\epsilon_t + pV_t = 0 \tag{23}$$

$$p = P(\epsilon, V) \tag{24}$$

These equations take on a particularly simple and useful form in one dimension. As with the Eulerian equations, we combine the formulas by writing $\alpha = 1, 2, 3$ for slab, cylindrical, or spherical symmetry. With r as the Lagrangian variable we have

$$\rho_0 \frac{\partial u}{\partial t} = -\left(\frac{R}{r}\right)^{\alpha-1} \frac{\partial p}{\partial r} \tag{25}$$

$$\frac{\partial R}{\partial t} = u \tag{26}$$

$$V = \frac{1}{\rho_0}\left(\frac{R}{r}\right)^{\alpha-1} \frac{\partial R}{\partial r} \tag{27}$$

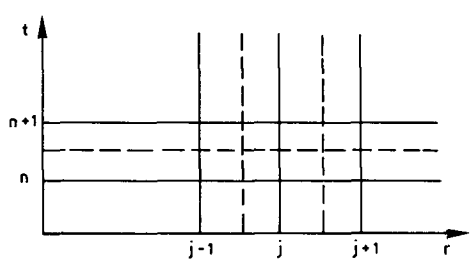


FIG. 1. Difference scheme for the solution of Eqs (25) - (29).

$$\frac{\partial \epsilon}{\partial t} = - p \frac{\partial V}{\partial t} \tag{28}$$

$$p = P(\epsilon, V) \tag{29}$$

where $R(r,t)$ is the position of the fluid element at time t, which was at $r = 0$ at $t = 0$.

3. DIFFERENCE METHODS

3.1. Lagrangian difference equations

A difference scheme for the solution of Eqs (25) - (29) is given in Fig. 1.

We subdivide the r,t domain as shown above and denote $R(r_j, t_n) = R_j^n$, etc.

$$u_j^{n+\frac{1}{2}} - u_j^{n-\frac{1}{2}} = - \frac{V_0 \Delta t}{\Delta r} \left(\frac{R_j^n}{r_j} \right)^{\alpha-1} (p_{j+\frac{1}{2}}^n - p_{j-\frac{1}{2}}^n) \tag{30}$$

$$R_j^{n+1} - R_j^n = \Delta t \, u_j^{n+\frac{1}{2}} \tag{31}$$

$$V_{j+\frac{1}{2}}^{n+1} = V_0 \frac{(R_{j+1}^{n+1})^\alpha - (R_j^{n+1})^\alpha}{(r_{j+1})^\alpha - (r_j)^\alpha} \tag{32}$$

$$\epsilon_{j+\frac{1}{2}}^{n+1} - \epsilon_{j+\frac{1}{2}}^n = - \frac{1}{2} (p_{j+\frac{1}{2}}^n + p_{j+\frac{1}{2}}^{n+1})(V_{j+\frac{1}{2}}^{n+1} - V_{j+\frac{1}{2}}^n) \tag{33}$$

$$p_{j+\frac{1}{2}}^{n+1} = p(\epsilon_{j+\frac{1}{2}}^{n+1}, V_{j+\frac{1}{2}}^{n+1}) \tag{34}$$

where $V_0 = 1/\rho_0$.

The above system is an explicit scheme as long as the calculations are performed in the given order. In Eq. (33), the quantity $p_{j+\frac{1}{2}}^{n+1}$ must be estimated at first and then the last two equations are iterated at least once on each time step. This centring gives a scheme that is accurate to second order in Δt and Δr.

The stability condition for the system of equations (30) - (34) is given by

$$\frac{c \Delta t}{R_{j+1}^n - R_j^n} \leq 1 \qquad \text{for all } n, j \tag{35}$$

In one-dimensional problems, the Lagrangian scheme is very simple and perhaps the best to use. Another advantage of the Lagrangian method is in problems where there are different materials present with different equations of state. The Lagrangian variable identifies fluid elements so the correct equation of state is automatically used.

In two dimensions the Lagrangian method has the serious disadvantage that the mesh becomes badly distorted in time and re-zoning is required.

Nevertheless, for problems with several materials it is widely used [2]. In one-fluid problems such as in meteorology, ocean circulation, and magnetohydrodynamics, the Eulerian form of the equations is preferable.

When shocks are present in the flow, the Eqs (30) and (33) are modified to include the von Neumann - Richtmyer [1] artificial viscosity.

3.2. Eulerian difference equations — hyperbolic systems

The differential equations (4) - (6) or (7) - (10) form what is called a hyperbolic system. Consider the system of equations

$$\frac{\partial u_i}{\partial x} + \sum_{j=1}^{n} a_{ij} \frac{\partial u_j}{\partial y} + b_i = 0 \qquad i = 1, 2, \ldots, n \tag{36}$$

Here the a_{ij} and b_i are functions of x, y, u_1, $u_2, \ldots, u_n$. This system of equations is non-linear, but the coefficients do not depend on $\partial u_i/\partial x$ or $\partial u_i/\partial y$. A system such as this is called a quasi-linear system. Let us transform system (36) as follows:

$$\sum_{i=1}^{n} v_{ki} \frac{\partial u_i}{\partial x} + \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ki} a_{ij} \frac{\partial u_j}{\partial y} + \sum_{i=1}^{n} v_{ki} b_i = 0 \tag{37}$$

Now we choose the v_{ki} such that

$$\sum_{i=1}^{n} v_{ki} a_{ij} = \lambda_k v_{kj} \qquad k, j = 1, 2, \ldots, n$$

Then we can write Eq. (37) as

$$\sum_{j=1}^{n} v_{kj} \left(\frac{\partial u_j}{\partial x} + \lambda_k \frac{\partial u_j}{\partial y} \right) + \sum_{j=1}^{n} v_{kj} b_j = 0$$

Note that for this system we have n characteristic directions given by

$$\frac{dy}{dx} = \lambda_k \qquad k = 1, 2, \ldots, n$$

The equation for the eigenvalues is

$$\det(A - \lambda I) = \begin{vmatrix} a_{11} - \lambda & a_{12} & a_{13} \ldots & a_{1n} \\ a_{21} & a_{22} - \lambda & a_{23} \ldots & a_{2n} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ a_{n1} & \cdot \; \cdot \; \cdot & & a_{nn} - \lambda \end{vmatrix} = 0$$

If A ($A = (a_{ij})$) has n real eigenvalues, not necessarily distinct, and has a full set of eigenvectors $V_k = (v_{k1}, v_{k2}, \ldots, v_{kn})$, $k = 1, 2, \ldots, n$, then Eq. (36) is called a hyperbolic system. If A is symmetric, then Eq. (36) is said to be a symmetric hyperbolic system. This is of note because much of the theory for hyperbolic systems has been developed for symmetric hyperbolic systems. We can write Eq. (36) in vector form as

$$u_t + Au_x + B = 0$$

where we have made the notational changes of replacing x by t and y by x.

Now consider an example

$$u_t + uu_x + \frac{1}{\rho} P_x = 0$$

$$\rho_t + u\rho_x + \rho u_x = 0$$

Assume, as an equation of state, $P = k\rho^{\gamma}$. Then

$$c^2 = \left(\frac{\partial P}{\partial \rho}\right)_s = \gamma k \rho^{\gamma - 1}$$

so

$$\frac{1}{\rho} P_x = \frac{c^2}{\rho} \rho_x$$

Hence we can write

$$u_t + uu_x + \frac{c^2}{\rho} \rho_x = 0$$

$$\rho_t + u\rho_x + \rho u_x = 0$$

or

$$\begin{bmatrix} u_t \\ \rho_t \end{bmatrix} + \begin{bmatrix} u & c^2/\rho \\ \rho & u \end{bmatrix} \begin{bmatrix} u_x \\ \rho_x \end{bmatrix} = 0$$

To find the characteristics, we set

$$\det(A - \lambda I) = 0$$

or

$$\begin{vmatrix} u - \lambda & c^2/\rho \\ \rho & u - \lambda \end{vmatrix} = (u - \lambda)^2 - c^2 = 0$$

We find

$$\lambda = u \pm c$$

Hence the characteristics are

$$\frac{dx}{dt} = u \pm c$$

We can write Eqs (4) - (6) in the form

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial x} + B = 0 \tag{38}$$

where U and B are three-dimensional column vectors and A is a 3×3 matrix. Since we are mainly interested in the advective terms, we shall consider the simpler system

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial x} = 0 \tag{39}$$

We consider a finite-difference grid with $x_j = j\Delta x$, $t_n = n\Delta t$, where j and n are integers and $U_j^n = U(x_j, t_n)$. The simplest difference approximation to Eq. (39) is

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{2\Delta x} A_j^n (U_{j+1}^n - U_{j-1}^n)$$

which is unstable. (The stability criteria given in this discussion are derived assuming that the coefficients are constant, so for our non-linear equations they must be regarded as local conditions which must be tested numerically.) A better scheme is the so-called "upstream-downstream" difference equation. For scalar u we have

$$\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0$$

The difference equations are

$$u_j^{n+1} = u_j^n - \frac{a_j^n \Delta t}{\Delta x} \begin{cases} u_{j+1}^n - u_j^n & \text{if } a_j^n < 0 \\ u_j^n - u_{j-1}^n & \text{if } a_j^n > 0 \end{cases} \tag{40}$$

The stability condition is $|a\Delta t/\Delta x| < 1$. A scheme with higher-order accuracy is the "leap-frog" equation

$$U_j^{n+1} = U_j^{n-1} - \frac{\Delta t}{\Delta x} A_j^n (U_{j+1}^n - U_{j-1}^n) \tag{41}$$

which has the same stability condition $|a\Delta t/\Delta x| < 1$ for all eigenvalues a of A, but has the disadvantage of being a three-level equation. Another equation which has second-order accuracy but uses only two time levels is based on the expansion

$$U_j^{n+1} = U_j^n + \Delta t \left(\frac{\partial U}{\partial t}\right)_j^n + \frac{(\Delta t)^2}{2} \left(\frac{\partial^2 U}{\partial t^2}\right)_j^n$$

If A is assumed constant, then we have the difference equation

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{2\Delta x} A(U_{j+1}^n - U_{j-1}^n) + \frac{1}{2}\left(\frac{A\Delta t}{\Delta x}\right)^2 (U_{j+1}^n - 2U_j^n + U_{j-1}^n) \quad (42)$$

When A is not constant, the above equation can become much more complicated. The condition is $|a\Delta t/\Delta x| < 1$. Again with A assumed constant we can go to fourth-order accuracy by using five points:

$$\begin{aligned} U_j^{n+1} = U_j^n &- \frac{1}{12}\left(\frac{A\Delta t}{\Delta x}\right)[8(U_{j+1}^n - U_{j-1}^n) - (U_{j+2}^n - U_{j-2}^n)] \\ &- \frac{1}{24}\left(\frac{A\Delta t}{\Delta x}\right)^2[30\,U_j^n - 16(U_{j+1}^n + U_{j-1}^n) + (U_{j+2}^n - U_{j-2}^n)] \\ &- \frac{1}{12}\left(\frac{A\Delta t}{\Delta x}\right)^3[-2(U_{j+1}^n - U_{j-1}^n) + (U_{j+2}^n - U_{j-2}^n)] \\ &+ \frac{1}{24}\left(\frac{A\Delta t}{\Delta x}\right)^4[6\,U_j^n - 4(U_{j+1}^n + U_{j-1}^n) + (U_{j+2}^n + U_{j-2}^n)] \end{aligned} \quad (43)$$

Equations (40) - (43) are all examples of explicit difference schemes and they all have the same stability criteria.

It is sometimes possible to write the system in conservation form

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0 \quad (44)$$

where F is an m-dimensional column vector. The system of equations (4) to (6) can be put into this form. We define $m = \rho u$ and $e = \rho(\epsilon + \frac{1}{2}u^2)$, where ρ, m, e are the mass, momentum and energy per unit volume. The equations take the form (44) where U and F(U) are vectors, defined as

$$U = \begin{bmatrix} \rho \\ m \\ e \end{bmatrix} \qquad F(U) = \begin{bmatrix} m \\ (m^2/\rho) + p \\ (e+p)\,m/\rho \end{bmatrix} \quad (45)$$

The pressure is given by the equation $p = P(\epsilon, V)$, where $P(\epsilon, V)$ is the equation of state of the fluid. For the Lagrangian formulation in slab symmetry, we can eliminate R to give:

$$\frac{\partial V}{\partial t} = V_0 \frac{\partial u}{\partial x} \quad (46)$$

and define E, the total energy per unit mass, $E = \epsilon + u^2/2$. Then with

$$U = \begin{bmatrix} V \\ u \\ E \end{bmatrix} \qquad F(U) = V_0 \begin{bmatrix} -u \\ p \\ pu \end{bmatrix} \tag{47}$$

where $p = P(E - u^2/2, V)$, the equations are in the conservative law form. The Lax-Wendroff system of difference equations is given by

$$U_j^{n+1} = U_j^n - \frac{1}{2}\frac{\Delta t}{\Delta x}(F_{j+1}^n - F_{j-1}^n) + \frac{1}{2}\left(\frac{\Delta t}{\Delta x}\right)^2 [A_{j+\frac{1}{2}}^n (F_{j+1}^n - F_j^n) - A_{j-\frac{1}{2}}^n (F_j^n - F_{j-1}^n)] \tag{48}$$

where the matrix A(U) is the Jacobian of F(U) with respect to U, that is, the matrix component A_{ij} is

$$A_{ij} = \partial F_i / \partial U_j$$

and the matrix $A_{j+\frac{1}{2}}^n$ denotes $A(\frac{1}{2}[U_{j+1}^n + U_j^n])$.

Where A is a constant matrix, F(U) = AU, and the system reduces to Eq. (42). An equivalent two-step Lax-Wendroff procedure with second-order accuracy was proposed by Richtmyer:

$$U_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \tfrac{1}{2}(U_{j+1}^n + U_j^n) - \frac{\Delta t}{2\Delta x}(F_{j+1}^n - F_j^n) \tag{49}$$

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{\Delta x}(F_{j+\frac{1}{2}}^{n+\frac{1}{2}} - F_{j-\frac{1}{2}}^{n+\frac{1}{2}}) \tag{50}$$

The stability condition in the Eulerian formulation is found by examining the Eulerian equations in the advection form

$$\frac{\partial U}{\partial t} + A\frac{\partial U}{\partial x} = 0$$

where

$$A = \begin{bmatrix} u & \rho & 0 \\ 0 & u & 1/\rho \\ 0 & \rho c^2 & u \end{bmatrix} \qquad U = \begin{bmatrix} \rho \\ u \\ p \end{bmatrix}$$

The stability condition for the Lax-Wendroff difference equations is then

$$(|u| + c)\frac{\Delta t}{\Delta x} < 1 \tag{51}$$

where the matrix A was assumed to be constant.

Similarly, for the advection form of the Lagrangian equations, where

$$A = \begin{bmatrix} 0 & -V_0 & 0 \\ 0 & 0 & V_0 \\ 0 & V_0 (c/V)^2 & 0 \end{bmatrix} \qquad U = \begin{bmatrix} V \\ u \\ p \end{bmatrix}$$

its eigenvalues are $0, \pm cV_0/V$, so that the stability condition is

$$\frac{V_0}{V} \frac{c\Delta t}{\Delta x} < 1 \tag{52}$$

Artificial viscosity can appear implicitly in the difference equations and results in an attenuation of short-wavelength components of the solution. The Lax-Wendroff scheme for the advection form

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial x} = 0$$

where U is a vector and A is a constant matrix, may be written as

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} + A \frac{U_{j+1}^n - U_{j-1}^n}{2\Delta x} = \frac{\Delta t}{2} A^2 \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{(\Delta x)^2}$$

A solution for the difference equation, $U(x,t)$, will not be an exact solution of the differential equation, but will be a solution of the modified differential equation

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial x} = QU \tag{53}$$

We can determine Q to various orders of Δt by demanding that U satisfy the difference equation to the specified order in Δt, for example, that the differential equation solution U satisfy

$$U_j^{n+1} - U_j^n + \frac{1}{2} \frac{A\Delta t}{\Delta x} (U_{j+1}^n - U_{j-1}^n) - \frac{1}{2} \left(\frac{A\Delta t}{\Delta x} \right)^2 (U_{j+1}^n - 2U_j^n + U_{j-1}^n) = 0[(\Delta t)^4]$$

which, on Taylor series expanding, yields for Q

$$QU = -\frac{1}{6} A[(\Delta x)^2] \frac{\partial^3 U}{\partial x^3} \tag{54}$$

This Q is dispersive but not dissipative. Solving to the next order, the next correction is

$$Q'U = -\frac{1}{8} A^2 \Delta t [(\Delta x)^2 - (A\Delta t)^2] \frac{\partial^4 U}{\partial x^4} \tag{55}$$

where the correction is now $Q+Q'$, which is dissipative. Empirical results show that even with this dissipation, oscillations and non-linear instabilities

can occur near sonic points or stagnation points. There an eigenvalue of A(u, U±c) is zero and the corresponding eigenvalue of the amplification matrix goes to one. Additional dissipation is then needed to prevent nonlinear instabilities. (Recall that all of our stability analysis has been done assuming A is constant. If the solution variables, hence A, vary fast near a sonic point or stagnation point, our stability analysis may be invalidated.)

3.3. Two-dimensional Eulerian difference equations

The Eulerian form of the differential equations in two dimensions, Eqs (7)-(10), can be written as

$$\frac{\partial U}{\partial t} + A\frac{\partial U}{\partial x} + B\frac{\partial U}{\partial y} + C = 0 \tag{56}$$

where U is a four-dimensional column vector and A and B are 4×4 matrices. The difference schemes discussed in the previous section can be generalized to two dimensions. In the case of the upstream-downstream and leap-frog methods the extension to two dimensions is obvious. With the second-order scheme, Eq. (42), we must be more careful [5]. Consider the simpler equation

$$\frac{\partial U}{\partial t} + A\frac{\partial U}{\partial x} + B\frac{\partial U}{\partial y} = 0 \tag{57}$$

We can represent Eq. (42) by the equation $U^{n+1} = (I + A')U^n$. A natural extension of the one-dimensional scheme for a regular mesh x_j, y_k with intervals Δx, Δy would be to take

$$U^{n+1}_{j,k} = U^n_{j,k} - \frac{\alpha}{2}(U^n_{j+1,k} - U^n_{j-1,k}) + \frac{\alpha^2}{2}(U^n_{j+1,k} - 2U^n_{j,k} + U^n_{j-1,k})$$

$$- \frac{\beta}{2}(U^n_{j,k+1} - U^n_{j,k-1}) + \frac{\beta^2}{2}(U^n_{j,k+1} - 2U^n_{j,k} + U^n_{j,k-1}) \tag{58}$$

where

$$\alpha = \frac{A\Delta t}{\Delta x}$$

and

$$\beta = \frac{B\Delta t}{\Delta y}$$

are the dimensionless interpolation coefficients in the two directions. This expression involves the five values of U at the mesh point (x_j, y_k) and the four surrounding points. We represent Eq. (58) by the equation

$$U^{n+1} = (I + A' + B')U^n$$

$$= RU^n$$

From the point of view of a two-dimensional Taylor series expansion of U about the point (x_j, y_k) the cross-term $AB(\Delta t)^2(\partial^2 U/\partial x\,\partial y)$ is missing, and so the suggested scheme is not truly accurate to second order.

Worse than this is the fact that this scheme is unstable. For an eigenvector of the form

$$e^{ikx_j}\ e^{i\ell y_k}$$

the associated eigenvalue of the recursion matrix R is

$$r_{k,\ell} = 1 - i\alpha \sin k\Delta x - \alpha^2(1 - \cos k\Delta x) - i\beta \sin \ell\Delta y - \beta^2(1 - \cos \ell\Delta y) \qquad (59)$$

An example of an eigenvalue outside the unit circle is given for $\cos kx = \cos \ell y = 1/2$, thus $\sin kx = \sin \ell y = 3^{\frac{1}{2}}/2$. In case $\alpha = \beta$ then

$$\begin{aligned} r_{k,\ell} &= 1 - \alpha^2 - i3^{\frac{1}{2}}\alpha \\ |r_{k,\ell}|^2 &= 1 + \alpha^2 + \alpha^4 \geq 1 \end{aligned} \qquad (60)$$

This is an example of an instability arising out of the interaction of two difference schemes, each of which is stable. Using the matrix notation, we have an example of the fact that, although the recursion matrix I+A' describing advection in the x-direction has eigenvalues on or in the unit circle and thus characterizes a stable difference scheme, as is also the case for the recursion matrix I+B' describing advection in the y-direction, it cannot be concluded that the combined recursion matrix I+A'+B' has its eigenvalues on or in the unit circle.

However, by a different composition of the two one-dimensional operators a stable scheme can be generated. Following the method of fractional time steps the recursion matrix is taken to be the product of the one-dimensional matrices. Thus

$$R = (I+A')(I+B') \qquad (61)$$

The stability of the individual factors is maintained in the product. In the simple example considered here the two matrices commute and so may be taken in either order. This recursion matrix R differs from the earlier one by the term $AB = BA$, which provides the full second-order accuracy missing from the earlier scheme, as well as stability.

In applying this scheme, the difference equation can be written as a sequence of two equations

$$\begin{aligned} U^{n+\frac{1}{2}}_{j,k} &= U^n_{j,k} - \frac{\beta}{2}(U^n_{j,k+1} - U^n_{j,k-1}) + \frac{\beta^2}{2}(U^n_{j,k+1} - 2U^n_{j,k} + U^n_{j,k-1}) \\ U^{n+1}_{j,k} &= U^{n+\frac{1}{2}}_{j,k} - \frac{\alpha}{2}(U^{n+\frac{1}{2}}_{j+1,k} - U^{n+\frac{1}{2}}_{j-1,k}) + \frac{\alpha^2}{2}(U^{n+\frac{1}{2}}_{j+1,k} - 2U^{n+\frac{1}{2}}_{j,k} + U^{n+\frac{1}{2}}_{j-1,k}) \end{aligned} \qquad (62)$$

The computational process involved in a single time-step is divided into two cycles. In the first of these, advection in the y-direction only is evaluated,

whereas in the second, advection in the x-direction only is evaluated starting, of course, with the results of the first cycle. The fractional time-step notation is a convenience, and the results of the first cycle would seem to have no particular physical significance.

The conservation form of Eqs (7) - (10) takes the form

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{63}$$

where

$$U = \begin{bmatrix} \rho \\ m \\ n \\ e \end{bmatrix} \qquad F = \begin{bmatrix} m \\ (m^2/\rho)+p \\ mn/\rho \\ (e+p)m/\rho \end{bmatrix} \qquad G = \begin{bmatrix} n \\ mn/\rho \\ (n^2/\rho)+p \\ (e+p)n/\rho \end{bmatrix}$$

and $m = \rho u$, $n = \rho v$, $e = \rho[\epsilon + \frac{1}{2}(u^2+v^2)]$.

A direct two-dimensional generalization of schemes (49) and (50) is used:

$$U_{j\ell}^{n+1} = \frac{1}{4}(U_{j+1\,\ell}^{n} + U_{j-1\,\ell}^{n} + U_{j\,\ell+1}^{n} + U_{j\,\ell-1}^{n}) - \frac{\Delta t}{2\Delta x}(F_{j+1\,\ell}^{n} - F_{j-1\,\ell}^{n}) - \frac{\Delta t}{2\Delta y}(G_{j\,\ell+1}^{n} - G_{j\,\ell-1}^{n}) \tag{64}$$

$$U_{j\ell}^{n+2} = U_{j\ell}^{n} - \frac{\Delta t}{\Delta x}(F_{j+1\,\ell}^{n+1} - F_{j-1\,\ell}^{n+1}) - \frac{\Delta t}{\Delta y}(G_{j\,\ell+1}^{n+1} - G_{j\,\ell-1}^{n+1}) \tag{65}$$

This scheme has been used by Burstein [6] for hypersonic flow. For the Lax-Wendroff scheme, a stability analysis which results from assuming A and B constant yields the condition

$$\frac{\Delta t}{\Delta} \leqq \frac{1}{\sigma^* \sqrt{8}}$$

Where $\Delta x = \Delta y = \Delta$, and σ^* is the maximum eigenvalue of A or B. Empirically, this can be exceeded by a factor of 2 in the steady state, if non-linear instabilities at shocks and stagnation points are not encountered. There a value of Δt, one-tenth of that required by the stability condition, can be unstable. Negative densities were found empirically in this case. The problem of unit values in the amplification matrix, with no effective damping, was responsible. To stabilize the difference equations, an artificial viscosity term is introduced of the form

$$\frac{\Delta t}{2\Delta x}[\delta_x(Q_x \delta_x U) + \delta_y(Q_y \delta_y U)] \tag{66}$$

where the δ's are centred difference operators and Q_x and Q_y are quadratic polynomials in A and B respectively,

$$Q_x = \sum_n a_n A^n \qquad Q_y = \sum_n b_n B^n \qquad n = 0, 1, 2$$

By proper choice of the coefficients (a_n), (b_n), the eigenvalues of Q_x and Q_y will be proportional to the difference of the eigenvalues of A and B across a mesh interval. Thus, where U changes slowly, a_n and b_n will be small; also the artificial viscosity will be small. It will be significant where U changes rapidly.

To analyse the stability with the artificial viscosity present, A and B are assumed locally constant, and for the case

$$Q_x = \frac{1}{2}\alpha A^2 \qquad Q_y = \frac{1}{2}\beta B^2$$

the resulting stability condition is

$$r \equiv \frac{\Delta t}{\Delta x}\left|\sigma^*\right| \leq \frac{1}{\sqrt{6}}$$

Thus the artificial viscosity improves stability. However, empirical results must be consulted because we have again linearized the problem by assuming A and B constant. For the above condition the criterion is $r \leq 0.408$. Empirical results are:

r	Condition of solution
0.65	Unstable at 90 cycles (negative densities)
0.55	Negative velocities at 350 cycles near stagnation point
0.45	Negative velocities at 640 cycles near stagnation point
0.35	No negative velocities or densities at 2500 cycles

4. APPLICATIONS

In this section, we shall consider a few applications of the methods discussed in the previous sections.

4.1. The motion of the earth's atmosphere

This section is based on the work of Leith [5]. In this model the hydrodynamic and thermodynamic evolution of a moist atmosphere on the whole globe is computed, taking into account such external influences as solar heating, evaporation and surface friction. The equations are those describing the thermodynamic relations and conservation of mass, momentum, energy, and water vapour.

It is assumed that the atmosphere is always in hydrostatic equilibrium. This means that the pressure at a given point is determined by the weight per unit area of the air above that point. A differential expression of this assumption is the hydrostatic relation

$$dp = -g\rho\, dz \qquad (67)$$

giving the increment in pressure dp in terms of an increment in height dz for given density ρ. Here g is the (assumed constant) acceleration of gravity that transforms the mass element $\rho\, dz$ into a weight element $g\rho\, dz$.

The hydrostatic assumption rules out the possibility of dynamic pressure differences leading to vertical accelerations. For horizontal scales of motion large compared to the thickness of the atmosphere this assumption is valid.

The hydrostatic assumption permits the replacement of z by p as an independent variable. There must be a replacement of p by z as dependent variable. The dependent variable that is used is the geopotential $\varphi = gz$, giving the potential energy per unit mass and serving as a measure of the height of a given pressure surface.

The earth is considered to be a sphere of radius $a = 2/\pi \times 10\,000 = 6366$ km. Horizontal position co-ordinates are latitude θ and east longitude λ.

It is important to distinguish two kinds of time derivative. The Eulerian time derivative $\partial/\partial t$ is based on the rate of change at a fixed point in the space co-ordinates (θ, λ, p). The Lagrangian time derivative D/Dt is based on the rate of change at a point imbedded in and moving with the fluid.

The horizontal velocity components are

$$u = a \cos\theta \frac{D\lambda}{Dt} \tag{68}$$

$$v = a \frac{D\theta}{Dt} \tag{69}$$

positive towards the east and pole respectively.

In the pressure co-ordinate system the vertical velocity component is

$$\omega = \frac{Dp}{Dt} \tag{70}$$

positive downward. In terms of these velocity components the relation between the two kinds of time derivative is

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u \frac{1}{a \cos\theta} \frac{\partial}{\partial \lambda} + v \frac{1}{a} \frac{\partial}{\partial \theta} + \omega \frac{\partial}{\partial p} \tag{71}$$

The kinematic boundary condition at the earth's surface $p = p_s$ gives

$$\omega_s = \left(\frac{Dp}{Dt}\right)_s = \frac{\partial p_s}{\partial t} + u_s \frac{1}{a \cos\theta} \left(\frac{\partial p_s}{\partial \lambda}\right)_s + v_s \frac{1}{a} \left(\frac{\partial p_s}{\partial \theta}\right)_s \tag{72}$$

where u_s, v_s are surface wind components, and the partial derivatives $(\partial p_s/\partial \lambda)_s$, $(\partial p_s/\partial \theta)_s$ are taken in the earth's surface.

At the top of the atmosphere $p = 0$ we have

$$\omega = 0 \tag{73}$$

Dry air is assumed to be a perfect gas satisfying the gas law

$$\frac{p}{\rho} = p\alpha = RT \tag{74}$$

where $R = 2.87\times10^6$ cm^2/s^2 deg is the gas constant, T is the absolute temperature in degrees Kelvin, and $\alpha = 1/\rho$ is the specific volume.

Water vapour is also assumed to be a perfect gas, but for it the gas constant R' is greater with $R/R' = \sigma = 0.622$.

In the pressure co-ordinate system an element of volume is an element of weight

$$dp\,dx\,dy = -\rho g\,dz\,dx\,dy = -g\,dm \tag{75}$$

where dm is a mass element. Thus conservation of mass requires non-divergent three-dimensional flow, i.e.

$$D + \frac{\partial\omega}{\partial p} = 0 \tag{76}$$

where D is the divergence of the two-dimensional wind field calculated in a pressure surface from horizontal components u, v, i.e.

$$D = \frac{1}{a\cos\theta}\left[\frac{\partial u}{\partial\lambda} + \frac{\partial v\cos\theta}{\partial\theta}\right] \tag{77}$$

Using the boundary condition Eq. (73) at $p = 0$, Eq. (76) can be integrated to give

$$\omega(p) = -\int_0^p D(p')\,dp' \tag{78}$$

The acceleration equations in the pressure co-ordinate system are

$$\frac{Du}{Dt} - \left\{2\Omega\sin\theta + \frac{u\tan\theta}{a}\right\}v = -\frac{1}{a\cos\theta}\frac{\partial\varphi}{\partial\lambda} + F_\lambda \tag{79}$$

$$\frac{Dv}{Dt} + \left\{2\Omega\sin\theta + \frac{u\tan\theta}{a}\right\}u = -\frac{1}{a}\frac{\partial\varphi}{\partial\theta} + F_\theta \tag{80}$$

The bracket terms take into account Coriolis and centrifugal-force terms; Ω is the angular velocity of the earth's rotation. F_λ and F_θ are frictional force components due largely to eddy viscosity. The acceleration due to pressure differences in this system is given by the negative gradient of geopotential.

With no heat sources or sinks the motion of a parcel of air in the atmosphere is adiabatic; that is, if s is entropy per unit mass,

$$\frac{Ds}{Dt} = 0 \tag{81}$$

A more convenient measure of entropy is the potential temperature θ defined as the temperature that a parcel of air would have if adiabatically compressed to a standard pressure p_0. For air, with $\gamma = C_p/C_v = 1.4$,

$$\theta = \left(\frac{p_0}{p}\right)^{\kappa} T \tag{82}$$

where

$$\kappa = 1 - \frac{1}{\gamma} = \frac{0.4}{1.4} = \frac{2}{7} = \frac{R}{C_p}$$

When heating occurs, the potential temperature equation becomes

$$\frac{D\theta}{Dt} = \frac{1}{C_p}\left(\frac{p_0}{p}\right)^{\kappa} \dot{q} \tag{83}$$

where $\dot{q}$ is the heating rate.

In the pressure co-ordinate system this equation is

$$\frac{\partial T}{\partial t} + u\frac{1}{a\cos\theta}\frac{\partial T}{\partial \lambda} + v\frac{1}{a}\frac{\partial T}{\partial \theta} + \omega\frac{T}{\theta}\frac{\partial \theta}{\partial p} = \frac{1}{C_p}\dot{q} \tag{84}$$

This is a formulation of the law of conservation of energy which implicitly takes into account changes in potential energy associated with changes in internal energy in a hydrostatic atmosphere.

The heating rate $\dot{q}$ includes the heating due to incoming solar radiation, the release of latent heat associated with water vapour condensation, and the eddy diffusion or convection of heat. It also includes the cooling due to outgoing terrestrial radiation.

The mixing ratio μ of grams of water vapour to total grams in a parcel of air is changed during its motion only by precipitation, evaporation, and eddy diffusion. If we denote by $\dot{r}$ these sources and sinks, we have then

$$\frac{D\mu}{Dt} = \frac{\partial \mu}{\partial t} + u\frac{1}{a\cos\theta}\frac{\partial \mu}{\partial \lambda} + v\frac{1}{a}\frac{\partial \mu}{\partial \theta} + \omega\frac{\partial \mu}{\partial p} = \dot{r} \tag{85}$$

The finite difference equations are based on two separate, fixed Eulerian space-time meshes which are distinguished as even and odd. Some dependent variables such as temperature and water vapour concentration are defined on the even mesh; others such as horizontal velocity components are defined on the odd mesh.

Pressure, p, is used as a vertical co-ordinate; even mesh points are at pressure levels of 100, 200, 400, 600, 850, and 1000 mb. Latitude, θ, and longitude, λ, are used as horizontal co-ordinates. Even mesh points are at even multiples of 5° in latitude and in longitude between the equator and 60° latitude. Poleward of 60° a coarsening of the mesh in the longitudinal direction is introduced. For 65°, 70°, and 75° latitude the even mesh points are at even multiples of 10° in longitude, for 80°, at even multiples of 20°,

and for 85°, at even multiples of 40°. The pole is an even mesh point. The time variable t is Greenwich Civil Time. The even mesh times are even multiples of 10 minutes starting from midnight.

Equations (79), (80), (84) and (85) form a system of advection equations of the type described in the last section. The scheme that is used to solve them is the fractional time-step or "splitting" method given by Eq. (62), generalized for variable coefficients.

4.2. Advection-diffusion equations, magnetohydrodynamics

In many time-dependent problems of practical interest we have, in addition to the hydrodynamic equations of hyperbolic type, coupled diffusion equations, i.e. equations of parabolic type. It is not unusual to have both of these types of equations in the same system.

The finite-difference methods for diffusion equations are different from those that we have been considering in that it is usually advisable to use implicit difference schemes. We shall illustrate these techniques with the equations of magnetohydrodynamics. The problems that we have chosen to discuss have all come from plasma physics – the physics of fully ionized gases. We have taken these problems from controlled thermonuclear research because this program has provided the impetus for much of the theoretical development of the subject and because only numerical methods could give the answers to specific questions in the design and analysis of experiments. Similar problems occur in geophysics.

There are many mathematical models that are used to describe a plasma in a magnetic field and these vary from the description of single-particle orbits to the kinetic theory of ionized gases. The fluid model, or magnetohydrodynamics (MHD), is basic, and any experiment is first analysed to determine if MHD equilibria and stability exist. Within the fluid theory various degrees of complexity are considered. The so-called ideal MHD is an infinite conductivity approximation. In some models the pressure is a scalar function, but in some problems it is necessary to consider it as a tensor with different values along and perpendicular to the magnetic field. The more realistic models include the transport coefficients, e.g. thermal conductivity and electrical resistivity, and these can also be scalars or tensors. In this section, we have made no attempt at completeness in describing MHD models, but have picked problems for their computational interest. The numerical methods described have all been used in the modelling of actual experiments.

The application of computers to plasma physics has advanced rapidly in the last few years. Volume 9 of Methods in Computational Physics (1970) [4] is devoted to this field. Most of it is devoted to articles on the solution of the Vlasov or collisionless Boltzmann equation and these include the many-particle simulation techniques. There are two chapters on collisional plasma models – one on the numerical solution of the Fokker-Planck equation for a plasma and one on magnetohydrodynamic calculations. The article by Roberts and Potter [7] gives a good review of the role of MHD computations and discusses methods for the solution of time-dependent problems.

The use of numerical calculations in the design and analysis of pinch experiments has been of central importance. In most of these computations the equations of magnetohydrodynamics are used. The earliest problems used the infinite conductivity theory in analysing the linear pinch. At

Livermore, in order to analyse some experiments, we coupled the pinch dynamical equations with the external circuit equations. The resulting set of differential equations was solved numerically.

As the stability theory of the pinch advanced and more complex experiments evolved, such as the diffuse "stabilized" pinch, hardcore pinch and theta pinch, more elaborate numerical computations have been made. Fortunately, at this same time, the speed and capacity of computing machines have been increasing at a rapid rate.

Most of the above experiments have been analysed with computations on a time-dependent, fully ionized hydromagnetic model. Electrical resistivity and thermal conductivity of the plasma are included and separate temperatures are assigned to the electrons and ions. The equations that we shall consider are

$$\frac{\partial \rho}{\partial t} + (\vec{v}\cdot\nabla)\rho = -\rho \operatorname{div}\vec{v} \tag{86}$$

$$\rho\left[\frac{\partial \vec{v}}{\partial t} + (\vec{v}\cdot\nabla)\vec{v}\right] = \vec{j}\times\vec{B} - \nabla p \tag{87}$$

$$\frac{\partial \theta_i}{\partial t} + (\vec{v}\cdot\nabla)\theta_i = -(\gamma-1)\theta_i \operatorname{div}\vec{v} + \frac{1}{\rho}\operatorname{div}(k_i \nabla\theta_i) + \frac{\theta_e - \theta_i}{\tau_{eq}} \tag{88}$$

$$\frac{\partial \theta_e}{\partial t} + (\vec{v}\cdot\nabla)\theta_e = -(\gamma-1)\theta_e \operatorname{div}\vec{v} + \frac{1}{\rho}\operatorname{div}(k_e \nabla\theta_e) - \frac{\theta_e - \theta_i}{\tau_{eq}} + \frac{\gamma-1}{\rho}\eta j_\theta^2 \tag{89}$$

$$\frac{\partial \vec{B}}{\partial t} + (\vec{v}\cdot\nabla)\vec{B} = -\vec{B}\operatorname{div}\vec{v} + (\vec{B}\cdot\nabla)\vec{v} - \operatorname{curl}\eta\vec{j} \tag{90}$$

where p is the plasma pressure, $p = \rho(\theta_i + \theta_e)$, θ_i and θ_e are the ion and electron temperatures, $\vec{B}$ is the magnetic field, and $\vec{j} = \operatorname{curl}\vec{B}$. Equations (88) - (90) are examples of advection-diffusion equations. Let us consider Eq. (90) in one-dimension using the cylindrical co-ordinate r as the spatial dimension. Let $B_z = B(r,t)$, $v_r = v(r,t)$, and $\eta(r,t)$ is the electrical resistivity, then Eq. (90) becomes

$$\frac{\partial B}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}(rvB) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\eta\frac{\partial B}{\partial r}\right) \tag{91}$$

It is important [1] to use an implicit difference approximation for the diffusion term in Eq. (91), otherwise Δt must satisfy $\eta\Delta t/(\Delta r)^2 \leq 1/2$, which is a very severe condition. The equation can be treated by the "splitting" technique [1] which is also used in multi-dimensional problems. The calculation of each time-step is split into two cycles. On the first cycle the advection equation

$$\frac{\partial B}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}(rvB) = 0 \tag{92}$$

is solved by one of the explicit difference equations described previously.

On the second cycle the diffusion equation

$$\frac{\partial B}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\eta\frac{\partial B}{\partial r}\right) \tag{93}$$

is solved by an implicit difference scheme (which is stable) using the results of the first cycle.

We can write an implicit difference approximation to Eq. (93). Let $r_j = j\Delta r$, $t_n = n\Delta t$, $B_j^n = B(r_j, t_n)$, etc. Consider the following equation

$$B_j^{n+1} - B_j^n = \frac{\theta\Delta t}{(\Delta r)^2 r_j}\left[r_{j+\frac{1}{2}}\eta_{j+\frac{1}{2}}^{n+1}(B_{j+1}^{n+1} - B_j^{n+1}) - r_{j-\frac{1}{2}}\eta_{j-\frac{1}{2}}^{n+1}(B_j^{n+1} - B_{j-1}^{n+1})\right]$$

$$+\frac{(1-\theta)\Delta t}{(\Delta r)^2 r_j}\left[r_{j+\frac{1}{2}}\eta_{j+\frac{1}{2}}^{n}(B_{j+1}^{n} - B_j^{n}) - r_{j-\frac{1}{2}}\eta_{j-\frac{1}{2}}^{n}(B_j^{n} - B_{j-1}^{n})\right] \tag{94}$$

where θ is a weighting constant; for stability we must have $1/2 \leq \theta \leq 1$. In the above it is assumed that the η_j^{n+1} have already been computed. We must solve an algebraic system of the form

$$-\alpha_j^n B_{j+1}^{n+1} + \beta_j^n B_j^{n+1} - \gamma_j^n B_{j-1}^{n+1} = \delta_j^n \tag{95}$$

for $j = 1, \ldots, J-1$. In order to solve the linear system given by Eq. (95) we use the algorithm given in Ref. [1]. Let

$$B_j^{n+1} = E_j^n B_{j+1}^{n+1} + F_j^n \tag{96}$$

where E_j^n and F_j^n are determined by the recurrence relations

$$E_j^n = (\beta_j^n - \gamma_j^n E_{j-1}^n)^{-1}\alpha_j^n \tag{97}$$

$$F_j^n = (\beta_j^n - \gamma_j^n E_{j-1}^n)^{-1}(\delta_j^n + \gamma_j^n F_{j-1}^n) \tag{98}$$

for $j = 1, \ldots, J$. To determine E_0^n, F_0^n we use the boundary conditions at $r = 0$, e.g. if $\partial B/\partial r = 0$ then $E_0^n = 1$, $F_0^n = 0$ or if $B(0,t) = B_0$ then $E_0^n = 0$, $F_0^n = B_0$. The computation consists of two sweeps of the r mesh. On the first sweep the E_j^n and F_j^n are computed using Eqs (97) and (98), and on the return sweep the B_j^{n+1} are computed from Eq. (96).

To give a better description of many of the experiments in CTR programs a two-dimensional model is needed. We consider a cylindrical system with azimuthal symmetry. The variables are then functions of r, z, and t. Because of the success of the earlier calculations in describing certain experiments, we shall again consider the hydromagnetic model. For purposes of discussion we shall consider a two-fluid model (unequal electron and ion temperatures) with thermal conduction. The velocity has components v_r and v_z and the magnetic field has components B_r and B_z. This model is

suitable for analysing theta pinches, conical plasma guns, and other cylindrical magnetic compression experiments. The use of computers for two-dimensional magnetohydrodynamic calculations is well established. The current density is $\vec{j} = (0, j_\theta, 0)$ with

$$j_\theta = \frac{\partial B_r}{\partial z} - \frac{\partial B_z}{\partial r} \tag{99}$$

Equations (86) and (87) can be written in terms of cylindrical coordinates r and z as the following hyperbolic system

$$\frac{\partial U}{\partial t} + A\frac{\partial U}{\partial r} + B\frac{\partial U}{\partial z} + F = 0 \tag{100}$$

where

$$U = \begin{bmatrix} \rho \\ v_r \\ v_z \end{bmatrix} \qquad A = \begin{bmatrix} v_r & \rho & 0 \\ 0 & v_r & 0 \\ 0 & 0 & v_r \end{bmatrix}$$

$$B = \begin{bmatrix} v_z & 0 & \rho \\ 0 & v_z & 0 \\ 0 & 0 & v_z \end{bmatrix} \qquad F = \begin{bmatrix} \rho(v_r/r) \\ -(B_z/\rho)j_\theta + (1/\rho)\partial p/\partial r \\ (B_r/\rho)j_\theta + (1/\rho)\partial p/\partial z \end{bmatrix}$$

The difference schemes discussed in Section 3 for one-dimensional hyperbolic systems can be generalized with some modifications to Eq. (100). The simplest explicit scheme would be to rearrange the equations so that A and B are diagonal with v_r and v_z as the diagonal elements and then use the "upstream-downstream" differencing scheme. Let $z_i = i\Delta z$, $r_j = j\Delta r$, and $t_n = n\Delta t$, then

$$U^{n+1}_{i,j} = U^n_{i,j} - \frac{A'^n_{i,j}\Delta t}{\Delta r}\begin{cases} U^n_{i,j+1} - U^n_{i,j} & \text{if } (v_r)^n_{i,j} < 0 \\ U^n_{i,j} - U^n_{i,j-1} & \text{if } (v_r)^n_{i,j} > 0 \end{cases}$$

$$- \Delta t F'^n_{i,j} - \frac{B'^n_{i,j}\Delta t}{\Delta z}\begin{cases} U^n_{i+1,j} - U^n_{i,j} & \text{if } (v_z)^n_{i,j} < 0 \\ U^n_{i,j} - U^n_{i-1,j} & \text{if } (v_z)^n_{i,j} > 0 \end{cases}$$

where A' and B' are diagonal and F' is suitably modified from F. This scheme is effective for short time calculations such as the implosive phase of a theta pinch or the expansion of a plasma across a magnetic field. For longer time integrations a higher-order scheme is advisable, such as the angled-derivative and staggered mesh schemes discussed by Roberts and Weiss [8]. Another scheme is the fractional time-step method given by Eq. (62) or the two-step Lax-Wendroff method.

In their two-dimensional MHD codes Freeman and Lane [9] and Roberts and Potter [7] use a modified form of the Lax-Wendroff method for the advective equations.

The equations for the field components B_r and B_z in cylindrical coordinates r and z are given by

$$\frac{\partial B_r}{\partial t} + v_r \frac{\partial B_r}{\partial r} + v_z \frac{\partial B_r}{\partial z} = - B_r \left(\frac{v_r}{r} + \frac{\partial v_z}{\partial z} \right) + B_z \frac{\partial v_r}{\partial z} + \frac{\partial \eta}{\partial z} \left(\frac{\partial B_r}{\partial z} - \frac{\partial B_z}{\partial r} \right)$$

$$+ \eta \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial B_r}{\partial r} \right) + \frac{\partial^2 B_r}{\partial z^2} - \frac{B_r}{r^2} \right]$$

$$\frac{\partial B_z}{\partial t} + v_r \frac{\partial B_z}{\partial r} + v_z \frac{\partial B_z}{\partial z} = - B_z \left(\frac{\partial v_r}{\partial r} + \frac{v_r}{r} \right) + B_r \frac{\partial v_z}{\partial r} - \frac{\partial \eta}{\partial r} \left(\frac{\partial B_r}{\partial z} - \frac{\partial B_z}{\partial r} \right)$$

$$+ \eta \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial B_z}{\partial r} \right) + \frac{\partial^2 B_z}{\partial z^2} \right]$$

We see that these equations are of parabolic type. To avoid a severe time-step restriction the most suitable method of solution is the alternating-direction implicit difference scheme [1]. For this purpose it is convenient to write the above system in the form

$$\frac{\partial B}{\partial t} = a \frac{\partial^2 B}{\partial r^2} + c \frac{\partial^2 B}{\partial z^2} + d \frac{\partial B}{\partial r} + e \frac{\partial B}{\partial z} + fB \qquad (101)$$

where

$$B = \begin{bmatrix} B_r \\ B_z \end{bmatrix} \qquad a = c = \begin{bmatrix} \eta & 0 \\ 0 & \eta \end{bmatrix}$$

$$d = \begin{bmatrix} \left(\frac{\eta}{r} - v_r \right) & - \frac{\partial \eta}{\partial z} \\ 0 & \left(\frac{\eta}{r} + \frac{\partial \eta}{\partial r} - v_r \right) \end{bmatrix} \qquad e = \begin{bmatrix} \left(\frac{\partial \eta}{\partial z} - v_z \right) & 0 \\ - \frac{\partial \eta}{\partial r} & - v_z \end{bmatrix}$$

$$f = \begin{bmatrix} - \left(\frac{v_r}{r} + \frac{\partial v_z}{\partial z} + \frac{\eta}{r^2} \right) & \frac{\partial v_r}{\partial z} \\ \frac{\partial v_z}{\partial r} & - \left(\frac{\partial v_r}{\partial r} + \frac{v_r}{r} \right) \end{bmatrix}$$

In the computational sequence we calculate ρ, v_r, v_z at the new time-step using Eq. (100). Hence we know v_r, v_z and their derivatives at time t_{n+1}.

In the ADI method we treat r derivatives implicitly on one time-step and the z derivatives implicitly on the next step. The difference equation for n even is

$$\frac{B_{i,j}^{n+1} - B_{i,j}^{n}}{\Delta t} = a_{i,j}^{n+1} \frac{B_{i,j+1}^{n+1} - 2B_{i,j}^{n+1} + B_{i,j-1}^{n+1}}{(\Delta r)^2}$$

$$+ c_{i,j}^{n} \frac{B_{i+1,j}^{n} - 2B_{i,j}^{n} + B_{i-1,j}^{n}}{(\Delta z)^2} + d_{i,j}^{n+1} \frac{B_{i,j+1}^{n+1} - B_{i,j-1}^{n+1}}{2\Delta r}$$

$$+ e_{i,j}^{n} \frac{B_{i+1,j}^{n} - B_{i-1,j}^{n}}{2\Delta z} + \frac{1}{2}[f_{i,j}^{n+1} B_{i,j}^{n+1} + f_{i,j}^{n} B_{i,j}^{n}]$$

The difference equation for the next time-step (n odd) is

$$\frac{B_{i,j}^{n+1} - B_{i,j}^{n}}{\Delta t} = a_{i,j}^{n} \frac{B_{i,j+1}^{n} - 2B_{i,j}^{n} + B_{i,j-1}^{n}}{(\Delta r)^2}$$

$$+ c_{i,j}^{n+1} \frac{B_{i+1,j}^{n+1} - 2B_{i,j}^{n+1} + B_{i-1,j}^{n+1}}{(\Delta z)^2} + d_{i,j}^{n} \frac{B_{i,j+1}^{n} - B_{i,j-1}^{n}}{2\Delta r}$$

$$+ e_{i,j}^{n+1} \frac{B_{i+1,j}^{n+1} - B_{i-1,j}^{n+1}}{2\Delta z} + \frac{1}{2}[f_{i,j}^{n+1} B_{i,j}^{n+1} + f_{i,j}^{n} B_{i,j}^{n}]$$

The above difference approximations lead to the following algebraic systems to be solved, for n even

$$-(\alpha_e)_{i,j}^{n+1} B_{i,j+1}^{n+1} + (\beta_e)_{i,j}^{n+1} B_{i,j}^{n+1} - (\gamma_e)_{i,j}^{n+1} B_{i,j-1}^{n+1} = (\delta_e)_{i,j}^{n}$$

and for n odd

$$-(\alpha_0)_{i,j}^{n+1} B_{i+1,j}^{n+1} + (\beta_0)_{i,j}^{n+1} B_{i,j}^{n+1} - (\gamma_0)_{i,j}^{n+1} B_{i-1,j}^{n+1} = (\delta_0)_{i,j}^{n}$$

In the first equation we have a tri-diagonal matrix equation to solve for each value of i, and in the second equation we have a similar problem for j. Hence we can use the algorithm mentioned earlier for one-dimensional diffusion problems, but in this case the unknowns are vectors and the coefficients are 2×2 matrices.

The pair of equations (88) and (89) can be solved in a similar manner. In a version of their code, Freeman and Lane [9] solve the advective equations explicitly, but solve the diffusion equations by the ADI method as above. In the code of Roberts and Potter [7] both the advection and diffusion are done explicitly.

We have developed a general method for solving numerically two-dimensional, two-fluid magnetohydrodynamic equations [10]. The method uses second-order accurate, alternating direction implicit finite difference equations. The accuracy and implicitness of the codes which have been

developed using this method are features not available in previously reported codes. The equations which are solved by the codes are given by Eqs (86) to (90). The equations are solved in cylindrical co-ordinates with axial symmetry, so that no variation in the azimuthal direction occurs.

Denoting U as a nine-component vector whose components are dependent variables and κ as a four-component vector whose components are the transport coefficients, each a function of U, i.e.

$$U = (\rho,\ v_r,\ v_\varphi,\ v_z,\ \theta_i,\ \theta_e,\ B_r,\ B_\varphi,\ B_z)$$

and

$$\kappa = \kappa(U) = (\kappa_i,\ \kappa_e,\ 1/\tau_{eq},\ \eta)$$

Eqs (86) - (90) can be written in cylindrical co-ordinates with axial symmetry as

$$\frac{\partial U}{\partial t} + T\left(U,\ \frac{\partial U}{\partial r},\ \frac{\partial U}{\partial z},\ \frac{\partial^2 U}{\partial r^2},\ \frac{\partial^2 U}{\partial z^2},\ \kappa\,\frac{\partial \kappa}{\partial r},\ \frac{\partial \kappa}{\partial z}\right) = 0 \qquad (102)$$

where T is a nine-component, non-linear vector function of the arguments indicated. The alternating direction implicit finite difference approximations to Eq.(102) have one time-step of the form

$$(A)^n_{i,j}\,U^{n+1}_{i,j} + (B)^n_{i,j}\,U^{n+1}_{i+1,j} - (C)^n_{i,j}\,U^{n+1}_{i-1,j} = V^n_{i,j} \qquad (103)$$

and the following time-step

$$(A')^{n+1}_{i,j}\,U^{n+2}_{i,j} + (B')^n_{i,j}\,U^{n+2}_{i,j+1} - (C')^{n+1}_{i,j}\,U^{n+2}_{i,j-1} = V'^{n+1}_{i,j} \qquad (104)$$

In the above equations, A, A', B, B', C, and C' are 9×9 matrices, V and V' are nine-component vectors, and the superscript n and subscripts i and j refer to the space-time point (t^n, r_i, z_j). The difference equation is tridiagonal in the implicit quantities and is solved by a generalization to vector equations of the method given by Richtmyer and Morton [1] for scalar equations.

Tokamak and Levitron geometries involve all three components of the fluid velocity and magnetic field. The computer codes written to calculate all nine dependent variables are also set up to calculate several subsets of the nine variables. The subsets are

$$U = (\rho,\ v_r,\ \theta_i,\ \theta_e,\ B_z)$$

applicable to one-dimensional theta pinches,

$$U = (\rho,\ v_r,\ \theta_i,\ \theta_e,\ B_\varphi,\ B_z)$$

applicable to one-dimensional stabilized z-pinches, and

$$U = (\rho,\ v_r,\ v_z,\ \theta_i,\ \theta_e,\ B_r,\ B_z)$$

appropriate for two-dimensional theta-pinch geometries and laser-produced plasma expansion studies. For one-dimensional calculation, of course, B' and C' are zero and the calculations are performed for only one value of j.

It is difficult to assess at this time the extent of the advantages of the alternating direction implicit MHD calculations over the several explicit schemes reported with respect to enhanced numerical stability. Our calculations performed to date indicate numerical stability for time-steps several times larger than would be allowed with an explicit method. It is expected that the implicitness and second-order accuracy of the method presented above will allow calculations covering a longer real time than is presently feasible with explicit methods. With the advent of the new generation of computers it is certain that alternating direction implicit calculations in magnetohydrodynamics and other areas of computational physics will be performed more and more frequently.

REFERENCES

[1] RICHTMYER, R.D., MORTON, K.W., Difference Methods for Initial-Value Problems, J. Wiley, New York (1967).

[2] ALDER, B., FERNBACH, S., ROTENBERG, M., Methods in Computational Physics 3, Fundamental Methods in Hydrodynamics, Academic Press, New York (1964).

[3] ALDER, B., FERNBACH, S., ROTENBERG, M., Methods in Computational Physics 4, Applications in Hydrodynamics, Academic Press, New York (1965).

[4] ALDER, B., FERNBACH, S., ROTENBERG, M., Methods in Computational Physics 9, Plasma Physics, Academic Press, New York (1970).

[5] LEITH, C.E., in Methods in Computational Physics 4, Academic Press, New York (1965) 1.

[6] BURSTEIN, S.Z., J. Comput. Phys. 1 (1966) 198.

[7] ROBERTS, K.V., POTTER, D.E., in Methods in Computational Physics 9, Academic Press, New York (1970) 340.

[8] ROBERTS, K.V., WEISS, N.O., Math. Comput. 20 (1966) 272.

[9] FREEMAN, J.R., LANE, F.O., in Proc. A.P.S. Topical Conf. Numerical Simulation of Plasma, Los Alamos, N. Mex. (1968), LA-3990, paper C7.

[10] LINDEMUTH, I., KILLEEN, J., in Proc. 4th Conf. Numerical Simulation of Plasma, Naval Research Laboratory, Washington, D.C. (1970).

APPLICATION OF COMPUTERS TO PROBLEMS OF CONTROLLED THERMONUCLEAR REACTORS

M.N. ROSENBLUTH
Institute for Advanced Study,
Princeton, N.J.,
United States of America

Abstract

APPLICATION OF COMPUTERS TO PROBLEMS OF CONTROLLED THERMONUCLEAR REACTORS.
A discussion of the successful applications of computing, and its failures, to the study of the problem of stably confining a plasma in a so-called magnetic bottle.

In this paper, I shall discuss primarily the successful applications of computing, and its failures, to the study of the controlled release of thermonuclear energy, i.e. the problem of stably confining a plasma in a so-called magnetic bottle. For those totally unfamiliar with this problem let me briefly outline the idea. To produce a fusion energy release it is necessary to heat the plasma (consisting of light elements such as deuterium or a deuterium-tritium mixture) to temperatures sufficient to overcome the Coulomb repulsion between nuclei, that is, to temperatures of the order of 10^8 °K, at which point the thermonuclear reaction rates may become appreciable. To keep pressures at a reasonably low level where they can be held by conventional material structures, this implies densities of the order of $10^{16}/\text{cm}^3$, and the reaction rates then indicate the necessity of confinement times of the order of 0.1 s or more.

Note that at temperatures of 10^8 °K in deuterium sound speeds are of the order of 10^8 cm/s and hence natural hydrodynamic disassembly times are very short in reasonable-sized structures — much shorter than reaction times.

To digress briefly from the main thread of our argument, it should be noted that one way around this dilemma would be to relax the requirement of reasonable pressures and consequent low densities. An extreme of this is the hydrogen bomb, which is of course not a very attractive type of power station. Recently, considerable interest has been aroused in the possibility of "mini-bombs" in which high pressures are attained very briefly by shining intense laser light on solid deuterium-tritium pellets, with consequent yields small enough to be contained and utilized for power. Some magnetic shock schemes such as imploding a magnetized plasma with a metallic liner or the so-called "plasma focus" are other conceivable methods of obtaining densities high enough to achieve thermonuclear burn on a timescale compatible with hydrodynamic disassembly. In all these impulsive concepts most of the physics is well described by the standard hydrodynamic or magnetohydrodynamic equations, and two- or three-dimensional standard computing techniques are both suitable and essential for their investigation.

However, in this paper, I wish to confine my comments to what is now generally considered to be the most promising line of approach — that involving low-pressure long-time confinement by static magnetic fields — in particular the Tokamak concept. Let me mention the basic idea. The trajectory of a charged particle in a homogeneous magnetic field is of course a spiral along the field lines with gyroradius $a = MVc/eB$. With easily realizable fields these radii transverse to the field are of the order of a few millimetres, hence representing satisfactory transverse confinement. To prevent loss along the field lines it is natural to consider wrapping them into a torus as shown in Fig. 1. Here we depict a configuration symmetric around the major axis, i.e. $\partial/\partial\phi = 0$, with a primary magnetic field B_φ. Now, however, a problem arises in that the field is no longer uniform since from Ampère's Law we see $B_\varphi \sim 1/R$ with R the major radius. In such an inhomogeneous field there exist magnetic drifts upwards or downwards caused by the fact that the circles of gyration are larger towards the outside of the torus. This leads to rapid particle loss.

A simple expedient for curing this is to provide also a poloidal magnetic field B_θ as shown in cross-section in Fig. 2. Now the field is a spiral consisting of the main toroidal field B_φ plus the poloidal fields B_θ. As the

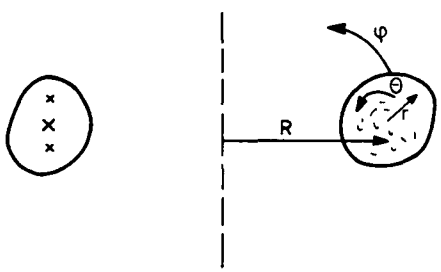


FIG. 1. Toroidal co-ordinate system.

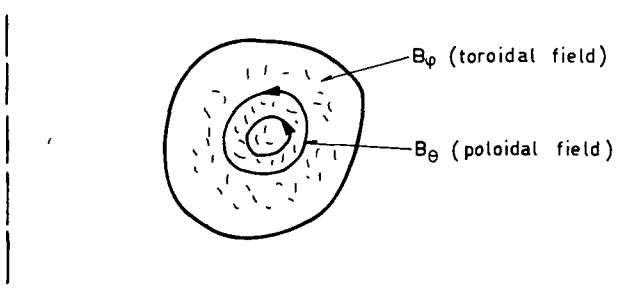


FIG. 2. Poloidal and toroidal field components.

particle orbits along the spiral field its magnetic drifts cancel so that it remains centred on a toroidal magnetic surface depicted in Fig. 2. An easy way to see this is to note that two constants of the motion exist, the total energy $\epsilon = \frac{1}{2} mV^2$ and the canonical angular momentum $p\varphi = R(mv_\varphi + eA_\varphi)$ where A_φ is the vector potential from which B_θ is derived. RA_φ is constant along a field line. From these constants we see immediately that motions away from the magnetic surface are limited in amplitude to the order of a gyroradius in the poloidal field.

How can we provide such a poloidal magnetic field? The simplest method is to apply a voltage around the torus, hence drawing a plasma current and creating the desired fields. This is the idea of the Tokamak. Other possible schemes involve the use of levitated internal rings as in the spherator, or external helical windings (inducing more complex fields but with the same basic principle) as in the stellarator.

Thus with the Tokamak a simple, satisfactory magnetic container for single particles may be achieved. This, however, is only the beginning of the problem. There remains first the problem of creating and heating the plasma and drawing the appropriate plasma currents. Much more difficult, there remains the problem of plasma collective effects and stability. That is, when a plasma is introduced into the system it is capable of producing charges and currents which may serve as the sources of fields which destroy the confinement properties of the bottle. Moreover, these collective effects may be unstable, small perturbations growing rapidly to destroy confinement. While analytical linear theory may go a long way in discussing such questions, only experiment or possibly computer simulation seems suitable for deciding the crucial nonlinear evolution of such instabilities.

The equations governing the system are the Boltzmann equation for the electron and ion distributions:

$$\frac{\partial f}{\partial t} + V \cdot \nabla f + \frac{e}{m} (\vec{E} + \vec{v} \times \vec{B}) \cdot \nabla_v f = \left(\frac{\partial f}{\partial t}\right)_{coll} \qquad (1)$$

with consequent charge and current densities

$$\rho = e \int f d^3\vec{v} \qquad \vec{j} = e \int f \vec{v} d^3\vec{v} \qquad (2)$$

plus Maxwell's equations. We observe that Eq. (1) is a six-dimensional time-dependent equation. At first sight it might appear that with particle simulation techniques it might be barely tractable on a computer. However, this is not so, due to the wide range of phenomena with different timescales which may occur. Thus electrostatic plasma oscillations have frequencies of $\sim 10^{12}$/s, as do electron gyrations; Alfvén waves and ion sound waves may have frequencies of 10^9; hydrodynamic motions and instabilities, 10^6; drift waves and trapped particle modes in which we are interested, 10^4 (resistive phenomena, 10^3); while we must study confinement times of the order of 10^{-1} s.

Clearly, a brute force approach is hopeless and only a very careful delineation of particular questions of interest with a consequent tailoring of the equations to eliminate rapid phenomena of no interest will allow

computers to be of help. Let me now indicate some of these questions and how the computer has been utilized.

The simplest such question is the magnetic field structure itself. For axisymmetric currents it is easy to show that the fields indeed have the desired structure of nested toroids. However, coils are not perfect and the field structure is quite sensitive to small errors. The computer may be of great help here. First, the vector potential is calculated from the applied currents, $\vec{B}$ is determined and then a field line is followed from the defining equation:

$$\frac{dr}{ds} = \frac{B_r}{B} \qquad \frac{r d\theta}{ds} = \frac{B_\theta}{B} \qquad \frac{R d\varphi}{ds} = \frac{B_\varphi}{B} \tag{3}$$

In this application, computers have been invaluable since the size and precision of field coils are only achieved with great cost, and while analytical mapping theory is capable of understanding the situation qualitatively, the necessary numerical factors are only attainable numerically. An interesting example is the stellarator where later computer runs indicated that many of the earlier experiments were ruined because the field lines were unconfined. A typical stellarator field structure is revealed in Fig. 3. Here we see nice nested toroids near the magnetic axis, surrounded by an "island structure", outside which the field lines are effectively ergodic and wander to the walls, being useless for confinement.

Nowadays, careful field structure computations are an essential feature of experimental design. A similar situation exists with respect to single particle orbits in slightly imperfect magnetic fields (for perfect symmetry they can be calculated analytically). For example, it is known, and this is the basis for magnetic mirror confinement, that the magnetic moment $\mu = mV_\perp^2/2eB$ of a particle is an adiabatic invariant of the motion. That is, for fields in which $(a_i/B)(dB/dx) = \epsilon \ll 1$, i.e. the field varies only a little in a gyroradius, the magnetic moment changes only like $e^{-\alpha/\epsilon}$, with α a constant of order unity which is difficult to determine analytically. To determine it accurately numerically for various shaped fields by computer usage may mean a factor of ten savings in equipment cost by indicating necessary field properties. A similar situation exists with respect to stellarator orbits.

Returning now to the more complex collective effects, we may divide the discussion into two parts — equilibrium and stability. For equilibrium considerations it is sufficient to consider an axisymmetric situation, and we are interested in the evolution of plasma density, temperature, and currents under the influence of the applied electric fields. Here the governing equations are those of the so-called neoclassical theory, which determine resistivity, particle and thermal diffusion, and various thermoelectric coefficients, together with Maxwell's equations. Numerical integration of these equations by various authors, particularly Duchs, Furth, and Rutherford, have succeeded in casting considerable light on the experimental Tokamak results. To date, such calculations have been one-dimensional (radial), adequate for small aspect ratio r/R, but could and will be extended without too much difficulty to the two-dimensional (r, θ) case. In fact, preliminary two-dimensional investigations using a hydrodynamic model and non-selfconsistent fields have already been made.

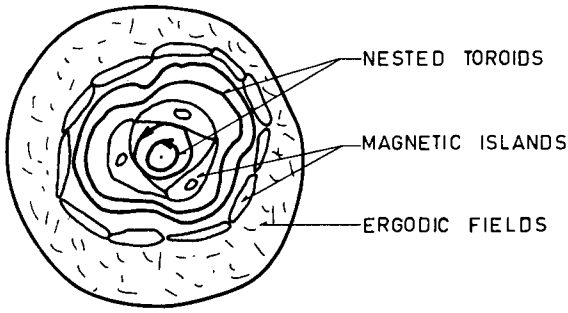


FIG. 3. Magnetic field structure patterns.

Here a degree of subtlety in excluding uninteresting fast Alfvén and sound waves along the field lines is required.

The principal result of comparison with experiments has been that while there is considerable agreement with neoclassical predictions of plasma behaviour, nonetheless there certainly exists an anomalously rapid skin effect – penetration of current into the plasma – pointing to the existence of some sort of instability leading to anomalous dissipation. With somewhat less certainty, because the impurity level is unknown, there are strong indications in the steady state of an anomalously high electron thermal transport.

This brings us to the really crucial question – plasma stability. At least to date the computer has not helped much here. Let us examine the complexities of the situation. I believe the basic problem is that the Tokamak is in fact very stable. In the familiar linear pinch, for example, hydromagnetic instabilities are very strong, comparable in rate to the fastest magnetohydrodynamic waves which can exist in the system. Such fast modes are easily amenable to computer study, but in the Tokamak the kink modes are driven extremely feebly and hence are hard to see by simulation – and likely to be masked by numerical noise.

For Tokamak plasmas which are large in scale and only subject to low-frequency oscillations, it is possible to average out the fast gyration in Eq. (1) and reduce the governing equations to a one-dimensional Vlasov-Boltzmann equation along the field lines, which is complicated by the fact that the field geometry is time-varying, plus the usual MHD-type fluid equations across the lines. Only some very preliminary numerical work has been attempted on this system. However, some phenomena, particularly the kink mode (see Fig. 4) or its resistive variant, the tearing mode, are probably adequately described by the pure fluid MHD equations.

The linear stability of an MHD equilibrium against arbitrary perturbations is described by an energy principle, the system being self-adjoint, in which it is studied whether an arbitrary plasma displacement $\vec{\xi}(\vec{r})$ can lower the plasma energy. In the Tokamak case, this may involve careful trial functions, a great deal of integration, and algebraic complexity even though the theory is in principle well known. An interesting application of computer algebraic manipulation to the problem of evaluating the energy integrals has been made by Biskamp and the Garching group.

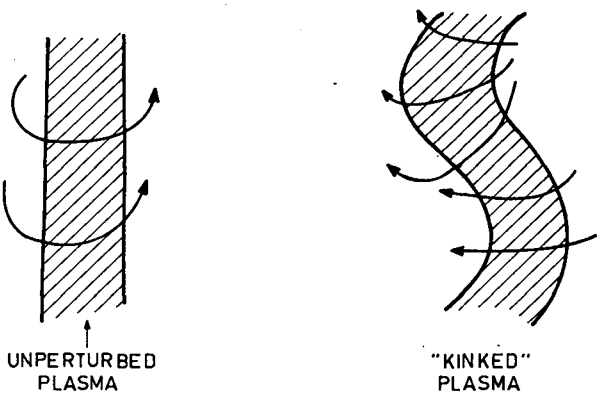


FIG. 4. Kink instability.

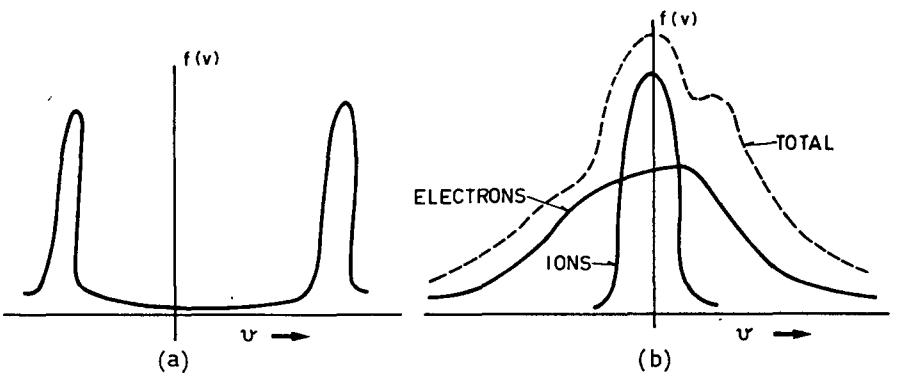


FIG. 5. Distribution functions for strong and weak instabilities.
(a) Two-stream instability; (b) Tokamak case.

A more interesting question is the nonlinear evolution of these kink modes which after all determines their effect on the plasma. To study this in full would be a three-dimensional time-dependent problem complicated by the weakness of the mode already referred to. Probably considerable information could be obtained by studying a restricted number of harmonics of the basic helical perturbation, i.e. a $2\frac{1}{2}$D calculation. As yet, these studies have not been attempted.

Finally, there remains the question of the microinstabilities, those for which the MHD equations are not adequate but which involve the use of at least the Vlasov equation along field lines as mentioned earlier.

Here, a great deal of linear analytical theory has been done on so-called drift waves and trapped-particle modes but the correct treatment of radial eigenfunctions remains to be done. Needless to say, the real interest is again in the nonlinear problem.

TABLE I. POSSIBLE COMPUTER APPLICATIONS TO ASPECTS OF THE TOKAMAK PROBLEM

Calculations	Present status	Relative importance	Possibility
Full treatment-time-dependent Fokker-Planck			No
Accurate determination of vacuum fields	√	1	
Single-particle orbits	√	1	
Evolution and 1 D guiding centre equilibrium	√	1	
Evolution and 2 D MHD equilibrium	~	Important for large r/R (1/4)	Yes, not too hard
Evolution and 2 D guiding centre equilibrium		Important for large r/R (1/2)	Yes, difficult
MHD instabilities – linear algebraic	~	1	Yes
Resistive mode – linear		1	Yes
MHD instabilities – nonlinear		2	$2\frac{1}{2}$ D or weak instability 3 D time dependent
Tearing modes – nonlinear		2	Different timescales
Anomalous transport coefficients – nonlinear drift waves, trapped particle modes		5	3 D or $2\frac{1}{2}$ D weak modes + guiding centres

To see the complexities for the computer, let us briefly compare the situation for the current-driven drift wave with that of the much computed two-stream instability. In both cases the instability arises because the distribution function of particle velocity parallel to B is not single-valued (see Fig. 5) because of the currents being drawn. In this case an interaction, such as ion sound waves, which conserves total momentum may lower the plasma energy by "interchanging" high and low velocity particles. In the two-stream instability (Fig. 5a), where the distribution function is violently non-monotonic, the sheet model simulations of K. Roberts and others have shed great light on the nonlinear evolution. As the distribution approaches more that of Fig. 5b the instabilities become more and more feeble and the numerical simulation of the effects of interest is progressively obscured by numerical noise.

Further, for actual Tokamak situations the distribution is completely linearly stable against all such one-dimensional modes, and only the so-called drift wave with $\omega \approx k_{\perp} (a_i/n) (dn/dx) V_i$ and $\omega \approx k_{\parallel} V_d$ remains unstable. As mentioned before, its radial dependence is also crucial. Hence, instead of studying a very strong one-dimensional effect, we are faced with simulating a very weak three-dimensional one. The techniques of simulation clearly will need to be greatly extended to study this case and probably some compromises with the full problem made. Nonetheless, it seems clear that the behaviour of these drift modes, and the similarly complex trapped particle modes, holds the key to the Tokamak future.

In Table I, possible computer applications are listed to aspects of the Tokamak problem with some assessment of their present status, probable importance, and comments. In the column marked "present status" a check mark indicates presently functioning codes; a ~ mark indicates the beginning of useful operation.

MONTE CARLO TECHNIQUES IN STATISTICAL MECHANICS

M.N. ROSENBLUTH
Institute for Advanced Study,
Princeton, N.J.,
United States of America

Abstract

MONTE CARLO TECHNIQUES IN STATISTICAL MECHANICS.

A study of the behaviour of systems with several hundred interacting particles, giving information on the properties of large assemblages.

One of the earliest applications of computers to physics was in the area of statistical mechanics. It was obvious that since it was now feasible to follow systems of several hundred interacting particles, one might be able to learn something about the properties of large assemblages – equations of state, transport coefficients, and perhaps the nature of phase transitions. In particular, the nature of the liquid-solid transition has never really been understood – whether, for example, its existence might depend on many-body attractive potentials or whether it should be exhibited by molecular systems interacting through only repulsive two-body potentials. Does it exist in two dimensions?

The most obvious computer approach to the problem is simply a time integration of the dynamical trajectories of N interacting particles placed in a box. Since $N^2/2$ pair potentials are involved, this can be very time consuming especially if long-range forces, e.g. Van der Waals' attractions, are important, and for this reason most applications of this so-called "molecular dynamics" approach have been restricted to the hard sphere problem where only nearby pairs need be considered. A detailed discussion of this topic is given by Killeen in paper SMR-9/17 in these Proceedings.

One question which immediately arises is how big is N – how large a system is needed? It helps, and is conventional among all workers in the field, to use periodic boundary conditions, so that particles near the right-hand boundary are also considered as neighbours of those near the left-hand boundary. Thus, the material is considered to be composed of a periodic array of identical samples of N particles.

One expects, of course, that a finite assemblage of particles will exhibit large fluctuations and will not have perfect phase transitions but smeared-out ones. In practice, one simply tries different values of N, hoping that nearly asymptotic results are obtained for computationally feasible values. In fact, samples of several hundred seem to give good results.

Let me now turn to the Monte Carlo calculations – the alternative to the straightforward dynamical method. We are discussing a classical problem and the method is based on Gibbs' canonical ensemble which says that the relative probability of any point in phase space is simply exp (-H/kT)

with the Hamiltonian H = K + V. Thus, we can calculate any desired ensemble-averaged physical quantity by:

$$\bar{A} = \frac{\int A\, e^{-H/kT}\, d^{3N}\vec{x}\, d^{3N}\vec{v}}{\int e^{-H/kT} d^{3N}\vec{x}\, d^{3N}\vec{v}}$$

As is well known, the velocity integrals may always be done explicitly, if we assume velocity-independent potentials due to the simple dependence of the kinetic energy on velocity. Hence, the average reduces to a 3N-dimensional integral over configuration space:

$$\bar{A} = \frac{\int A\, e^{-V/kT}\, d^{3N}\vec{x}}{\int e^{-V/kT}\, d^{3N}\vec{x}} \tag{1}$$

For example, to find the equation of state p(Y, T), the volume Y could be varied by changing the box size, the temperature varied in Eq. (1), and the pressure determined from the virial theorem:

$$pY = NkT - \tfrac{1}{2} \sum_{ij} r_{ij} \frac{\partial V(r_{ij})}{\partial r_{ij}} \tag{2}$$

where we have assumed a two-body potential dependent only on the distance r_{ij} between the i-th and j-th molecule. Our task would then be to use Eq. (1) to make a canonical average of the last term in Eq. (2) or equivalently to find the canonical average of the two-particle distribution function.

It is clear that a straightforward integration technique to evaluate the integrals in Eq. (1) is hardly feasible since even if we chose a very crude mesh of 10 points per dimension we would have 10^{3N} mesh points. On the other hand, it is obvious that if we have to perform a multidimensional integral of a smoothly varying function we can obtain accuracy of order $1/\sqrt{M}$ by simply summing over M randomly chosen points. This is the idea of Monte Carlo integration, first suggested, I believe, by S. Ulam.

For our problem, this most naive Monte Carlo approach would then consist of assigning random positions to the N particles, calculating the values of A and V, repeating this M times, and deducing

$$\bar{A} = \frac{\sum_{M} A_M e^{-V_M/kT}}{\sum_{M} e^{-V_M/kT}}$$

While this would be a correct procedure, it is not a practical one for the reason that there is usually a strong repulsive core to intermolecular potentials. Hence, when we choose a random point in phase space, it is

highly probable that two molecules will be close together, V will be very large, the exponential very small, and we will have chosen a very unimportant point in phase space.

Clearly, what we need is a weighted sampling procedure, one which selects points in phase space not with equal probability, but with weighted probability exp (-V/kT). To accomplish this, we play the following game beginning from an arbitrary initial configuration:

(1) Make a random move. For example, choose a particle at random, choose a direction at random. Move the particle a distance RZ, where Z is an appropriately chosen maximum displacement and R is a random number between zero and one.

(2) Calculate the change in potential energy ΔV caused by the move. This involves calculation of the change of N two-body potentials. If $\Delta V < 0$, allow the move. If $\Delta V > 0$, take a random number R' between 0 and 1 and allow the move if $R' < \exp(-\Delta V/kT)$.

(3) After modifying the configuration or not, according to rule (2), begin again with rule (1) for the next move and iterate the procedure as many times as desired.

If only rule (1) were used, then clearly after many moves we would have sampled all phase space with equal probability. By applying also rule (2), we sample phase space with the proper Gibbs weighting factor. To see that our prescription indeed leads to proper phase space averages, we note that the probability $F(\vec{X})$ that the configuration is at point X in phase space obeys the steady-state equation

$$F(X) = P(X' \rightarrow X)F(X')$$

where $P(X' \rightarrow X)$ is the probability of moving from $X' \rightarrow X$. Since the ratio of $P(X' \rightarrow X)$ and $P(X \rightarrow X')$ is just the Gibbs weighting factor $\exp[V(X') - V(X)/kT]$ it is clear that a solution for F(X) is $\exp[-V(X)/kT]$. By a method analogous to that used in demonstrating Boltzmann's H-theorem, we may indeed show that the canonical distribution is always approached by F.

After a transient determined by the arbitrary initial configuration has died away, we may assume therefore that the sampled configurations are in the canonical ratio and calculate the desired statistical averages. However, one word of caution is in order. Since moves always go to neighbouring configurations it is clear that many moves are needed to cover all phase space, i.e. to be effectively ergodic. In practice, it has been found that it is about optimal to choose the maximum displacement Z so that about half the moves are allowed. With this choice, a cycle of a few N moves (i.e. each particle moves a few times) seems adequate for sampling phase space.

The main virtue of the Monte Carlo technique vis-à-vis the molecular dynamics technique is that a much more rapid motion through phase space is allowed since to move each particle once in the Monte Carlo procedure takes about as much computation as one dynamical time-step. On the other hand, the latter is limited to be quite small to get accurate trajectories, while Z may be chosen relatively large.

The main disadvantage of the Monte Carlo technique is that it is limited to computation of equilibrium properties while, by following the dynamics, one may study rates of approach to equilibrium, transport coefficients, etc.

Perhaps the most interesting result obtained with these methods is the unequivocal demonstration from the pressure versus volume curves that in both two and three dimensions a solid-liquid phase transition (with about 5% expansion) occurs even for a fluid of hard spheres. In the Monte Carlo runs, one sees quite clearly a sudden change from a situation (liquid) where moves carry particles freely throughout the box to a situation (solid) where only small oscillations around fixed positions occur with reasonable frequency. Needless to say, just at the melting point there is a rather slow relaxation from the lattice to the fluid structure and 5 or 10% fluctuations in pressure are seen corresponding to surface energies between coexisting solid and liquid phases in our very finite box.

For future work it would seem to me that with present-day computers one could study by the Monte Carlo method more complex materials, for example, polyatomic molecules where both relative position and orientation of neighbours is relevant.

Finally, I would like to close by discussing a possible computer Monte Carlo study of another fundamental statistical mechanics problem – the zero temperature equation of state of a many-body system of Bose-Einstein particles such as helium-4. This is of course equivalent to finding the ground-state of such a system, governed by the Schrödinger equation

$$E\psi = -\frac{h^2}{2m}\nabla^2\psi + V\psi \tag{3}$$

where again the equation is to be interpreted as being in the 3N-dimensional configuration space and therefore too complex for straightforward integration techniques.

Again we may consider a Monte Carlo technique. Here we note the analogy between Eq. (3) and a diffusion equation. For example, to describe neutronics in a reactor structure, we may use the diffusion approximation

$$\frac{\partial n}{\partial t} = D\nabla^2 n + \sigma_f(\vec{r})n \tag{4}$$

where D is the diffusion coefficient and σ the local fission cross-section. For complicated reactor structures an alternative approach to the solution of Eq. (4) is simply to follow the trajectories of individual neutrons through the structure, allowing them to scatter or fission. This is the Monte Carlo neutronics method widely used by reactor designers. We now propose to solve the Schrödinger equation in the same way – representing the wave function ψ by a few randomly chosen systems of N particles, just as the neutron density n is represented by a few typical neutrons. (Note that the Monte Carlo neutronics method actually is a solution of the integral diffusion equation rather than Eq. (4). However, by allowing the scattering mean free path to become appropriately small, the integral problem always becomes equivalent to the diffusion approximation.)

The term $E\psi$ in Eq. (3) is replaced by $-\partial\psi/\partial t$ (note this is not the usual time-dependent Schrödinger equation) and E is then determined by the multiplication rate of the system. Naturally, in such a sampling calculation, only the fastest growing eigenmode will be seen. This corresponds to the lowest energy eigenvalue, hence we determine the ground-state energy. Moreover, as we are not enforcing any particular symmetry on the wave function, the lowest wave function will automatically be symmetric and the method applies to a Bose-Einstein fluid.

To go back to the neutronics analogy, we see that since the sample neutrons are in fact N-body systems the calculation is indeed rather ambitious. Nonetheless, 18 years ago, using the MANIAC computer, my wife and I succeeded in getting a reasonably good fit to the helium-4 pressure versus volume curve. Curiously enough, the lower mass of helium-3 seemed also to account pretty well for its higher ground-state energy even though the statistics were of course incorrect. However, with the relatively crude computer we used, the fluctuations were very large and the work remains unpublished. Probably the calculation could be done well on present computers.

A FLUID TRANSPORT ALGORITHM THAT WORKS

J.P. BORIS
Naval Research Laboratory,
Washington, D.C., United States of America

Abstract

A FLUID TRANSPORT ALGORITHM THAT WORKS.

This paper describes a class of explicit, Eulerian finite-difference algorithms for solving the continuity equation which are built around a technique called "flux correction". These Flux-Corrected Transport Algorithms are of indeterminate order but yield realistic, accurate results. In addition to the mass-conserving property of most conventional algorithms, the FCT algorithms strictly maintain the positivity of actual mass densities, so steep gradients and inviscid shocks are handled particularly well. This paper concentrates on a one-dimensional version called SHASTA.

INTRODUCTION

This paper proposes a new approach for numerically solving the continuity equation which yields physically reasonable results even in circumstances where standard algorithms fail. This new approach, called Flux-Corrected Transport (FCT), leads to a class of algorithms which strictly enforce the non-negative property of realistic mass and energy densities. As a result, steep gradients and shocks can be handled particularly well. A one-dimensional explicit Eulerian FCT algorithm is described here, and several computational examples and comparisons with more conventional methods are given. Revised versions of this paper are in preparation including complete treatments of the fluid equations on shock and piston problems.

In most conventional approaches using an Eulerian finite-difference grid [1, 2], the solution of

$$\frac{\partial \rho}{\partial t} = - \nabla \cdot (\rho \vec{v}) \tag{1}$$

is approximated by expanding locally up through a given order in the two parameters

$$\delta \equiv \left| \frac{\partial \rho / \partial x}{\rho / \delta x} \right|$$
$$\epsilon \equiv \frac{v \delta t}{\delta x} \tag{2}$$

In pursuing this approach, the standard finite-difference expansions can be endowed with certain desirable qualitative features such as stability

and exact conservation. For finite-difference methods of a given order, typically the first or second, the distinguishing qualitative features are determined by the error terms. The crucial importance of the form of the error comes glaringly to light in regions where ϵ and/or δ is of order unity, i.e. in most problems of physical interest.

In regions where the mass density $\rho(x)$ and the flow velocity $\vec{v}(x)$ in Eq. (1) are smooth, most second-order schemes, such as Lax-Wendroff [3, 4] or leapfrog [1, 4-6], treat the continuity equation quite adequately. In shocks and steep gradients, however, δ is of order unity. In regions of large velocity ϵ is of order unity because one always wants to take as large a timestep as possible, δ is also of order unity near sources of fluid near sinks, and at interfaces. In all of these cases, the truncation errors, formally of asymptotic order ϵ^3 or $\epsilon\delta^2$, are, in fact, as large as the solution, so numerical garbage is certain to arise. Using higher-order numerical methods in these regions does not help because the terms at all orders are roughly the same size.

The FCT finite-difference technique described here [7] circumvents these steep-gradient problems by requiring another physical property of the continuity equation, positivity, instead of vigorously adhering to an asymptotic ordering. The technique is also stable and mass-conservative and is essentially second-order in regions where the concept of order is meaningful. The FCT technique is of indeterminate order near sharp gradients, the physical behaviour of the continuity equation being folded into the technique directly. The FCT technique described here is explicit and Eulerian and therefore generalizes easily to two-and three-dimensional problems.

The FCT class of algorithms consist of two major stages, a transport or convective stage (stage I) followed by an antidiffusive or corrective stage (stage II). Both stages are conservative and maintain positivity. Their interaction enables FCT algorithms to treat strong gradients and shocks without the usual dispersively-generated ripples.

We illustrate the FCT principle by considering in detail a one-dimensional explicit algorithm called SHASTA. Section I of this paper discusses the transport stage in SHASTA. It has a simple geometric interpretation which generalizes for two-and three-dimensional applications. Section 1 also contains a detailed analysis of the transport stage. Section 2 is devoted to the antidiffusion stage of the algorithm (stage II) which corrects numerical errors introduced in stage I (Transport). An analysis of error in the overall algorithm is presented.

Section 3 treats the square-wave problem in one dimension using several conventional difference schemes as well as the FCT technique introduced in this paper and implemented in SHASTA. The concepts of dispersion and diffusion used in sections I and II are also applied to these conventional schemes. This analysis allows a more complete understanding of the new technique in light of the problems encountered in these earlier finite-difference schemes [1, 4, 8].

The SHASTA subroutine can be generalized to include equations of the momentum and pressure type as well as perfect continuity equations for the mass and energy density.

The results of several shock and piston calculations will be compared to other numerical methods of solving these problems in a future paper. We wish to emphasize here that the new algorithm employs no adjustable

artificial viscosity of the von Neumann type [1, 9] and displays none of the undershoot-overshoot phenomena which plague other finite-difference methods [8].

In section 4 the workings of the algorithm are described qualitatively and applications for the FCT algorithms are discussed.

1. THE TRANSPORT STAGE (STAGE I)

The transport stage of the FCT algorithm in one-dimension has a geometric interpretation with a simple extension to multi-dimensional calculations. Figure 1a shows the uniformly-spaced Eulerian grid at the beginning of the cycle (t = 0). The densities $\{\rho_j^0\}$ are known and the velocities $\{V_j^{\frac{1}{2}}\}$ are known at $t = \delta t/2$, half a time-step ahead. We seek the densities $\{\rho_j^1\}$ at the end of the time-step when $t = \delta t$. Stage I of the algorithm proceeds by considering individually each of the fluid-element trapezoids formed by connecting adjacent density values with straight line segments. This piece-wise linear density profile is fully consistent with the definition of the total mass,

$$M_0 = \sum_j \rho_j \, \delta x \tag{3}$$

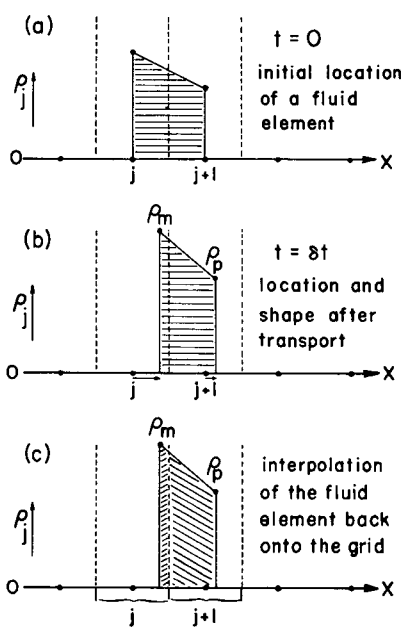


FIG. 1. The transport stage of the 1 D FCT algorithm SHASTA (stage I). Linear interpolations are used throughout so that mass is conserved and the density is non-negative as long as $|v\delta t/\delta x| < \frac{1}{2}$.

The short arrows at the grid points j and j + 1 indicate representative motions the fluid at these grid points might undergo. In the examples and analysis of this paper I consider only $|v\delta t/\delta x| < \frac{1}{2}$ so that no grid point can convect past the cell boundaries, indicated by vertical dashed lines in Fig. 1a, in one cycle. This restriction is probably not strictly necessary but ensures that no two grid points cross in a single cycle, a necessary condition for this algorithm. It also simplifies the programming immensely.

Following the motion of this fluid element for one time-step in a Lagrangian sense, the two boundaries move by amounts $v_j^{\frac{1}{2}}\,\delta t$ and $v_{j+1}^{\frac{1}{2}}\,\delta t$. At the end of the cycle, the fluid element has been convected and deformed as shown in Fig. 1b. The height of each side of the trapezoid is varied in inverse proportion to the contraction or dilation in x of the base of the trapezoid. Thus

$$\rho_j^{(+)} = \frac{\rho_{j+1}^0\,\delta x}{\delta x + \delta t\,(v_{j+1}^{\frac{1}{2}} - v_j^{\frac{1}{2}})} \qquad \text{and}$$
$$\rho_j^{(-)} = \frac{\rho_j^0\,\delta x}{\delta x + \delta t\,(v_{j+1}^{\frac{1}{2}} - v_j^{\frac{1}{2}})} \tag{4}$$

It is clear from these formulae that the area of the fluid element, and hence the mass, is fully conserved. It is also clear that the valves $\rho_j^{(+)}$ and $\rho_j^{(-)}$ are always non-negative if all the $\{\rho_j^0\}$ are non-negative initially and if

$$\left|\frac{v\delta t}{\delta x}\right| < \tfrac{1}{2}. \tag{5}$$

This ensures that the mass density, interpolated back onto the original grid, is also non-negative.

The velocities used in Eqs (4) are evaluated at the centred time but not the centred position with respect to the motion of the fluid at the grid points. Thus, the transport is not fully second-order. This minor defect is easily rectified by using modified velocities found by linearly interpolating on the grid. That is, replace $v_j^{\frac{1}{2}}$ in Eqs (4) with

$$\bar{v}_j^{\frac{1}{2}} = \left[1 - \frac{\epsilon_j}{2}\right] v_j^{\frac{1}{2}} + \frac{\epsilon_j}{2}\, v_{j+1}^{\frac{1}{2}} \quad (v_j^{\frac{1}{2}} > 0)$$
$$\bar{v}_j^{\frac{1}{2}} = \left[1 - \frac{\epsilon_j}{2}\right] v_j^{\frac{1}{2}} + \frac{\epsilon_j}{2}\, v_{j-1}^{\frac{1}{2}} \quad (v_j^{\frac{1}{2}} < 0) \tag{6}$$

where $\epsilon_j \equiv |v_j^{\frac{1}{2}}\,\delta t/\delta x|$.

Interpolation of the displaced fluid element back onto the original Eulerian grid is accomplished simply, as shown in Fig. 1c. The area of the trapezoid lying to the right of the cell boundary (midway between the j-th and the j+1-st grid points) can be calculated by a simple linear interpolation. This amount of fluid is assigned to grid point j+1. The remainder of the fluid in the trapezoid (residing to the left of the cell boundary) is assigned to grid point

j, since it lies in the j-th cell. All of the fluid elements are treated in the same way and independently. Thus, a portion of the fluid-element trapezoid which extended from j-1 to j initially also gets assigned to grid point j. In a similar way the cell j+1 also gets some of the fluid originating between grid points j+1 and j+2.

The transport prescription can be expressed algebraically as

$$\rho_j^{n+1} = \tfrac{1}{2} Q_-^2 \rho_{j-1}^n + \left[Q_-(1 - \tfrac{1}{2}Q_-) + Q_+ (1 - \tfrac{1}{2}Q_+) \right] \rho_j^n + \tfrac{1}{2} Q_+^2 \rho_{j+1}^n \qquad (7)$$

$$\text{where } Q_\pm = (\tfrac{1}{2} \mp v_j^{\frac{1}{2}} \delta t/\delta x) \Big/ \left[1 \pm (v_{j+1}^{\frac{1}{2}} - v_j^{\frac{1}{2}}) \delta t/\delta x \right]$$

For a uniform velocity field, this reduces to

$$\rho_j^{n+1} = \rho_j^n - \tfrac{1}{2} \epsilon (\rho_{j+1}^n - \rho_{j-1}^n) + (\tfrac{1}{8} + \frac{\epsilon^2}{2}) (\rho_{j+1} - 2\rho_j + \rho_{j-1}) \qquad (8)$$

This is a simple two-sided differencing of the dilation term plus a strong diffusion. Without the velocity-independent diffusion it is identical to the two-step Lax-Wendroff algorithm.

The treatment at the boundaries depends entirely on the physical problem being simulated and will not be considered in detail here since the geometric and numerical interpretation of any particular boundary condition is a straightforward generalization of the above.

This transport stage is conservative and non-negative as promised but has a very large, zeroth-order diffusion associated with it as well as the usual second-order dispersion and velocity-dependent diffusion. This zeroth-order diffusion arises principally as shown in Fig. 2 for the special case of v = 0. Consider ρ = 0 at all grid points but one as in Fig. 2a. The fluid in the two shaded regions is withdrawn from cell j and added to cells j - 1 and j + 1 during stage I. This strongly diffusive effect amounts to operating on the initial density profile $\{ \rho_j^0 \}$ in the following way:

$$\rho_j^1 = \rho_j^0 + \text{"}\tfrac{1}{8}\text{"} \left(\rho_{j+1}^0 - 2 \rho_j^0 + \rho_{j-1}^0 \right) \qquad (9)$$

In this simple zero-velocity example the diffusion coefficient "$\frac{1}{8}$" is strictly $\frac{1}{8}$ = 0.125. In the case of non-zero velocities "$\frac{1}{8}$" is roughly 0.125 plus a small velocity-dependent term.

Consider the slightly more general problem where $\epsilon \equiv v\delta t/\delta x$ is constant, greater than zero, but less than 0.5. Let

$$\rho_j^0 \equiv e^{ikj\delta x} \qquad (10)$$

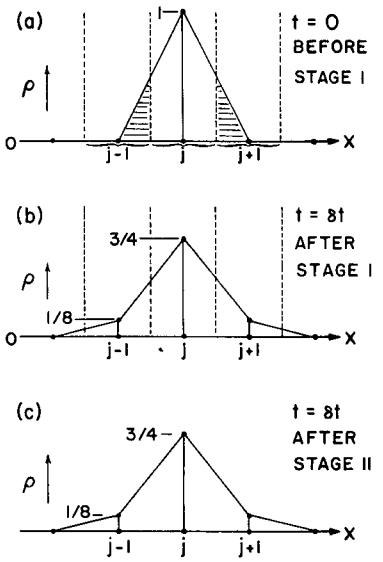


FIG.2. The strong diffusion of stage I. With the velocity everywhere zero, the central density value is strongly diffused, the shaded portions of (a) being transferred to the adjacent cells as in (b). The flux correction of stage II prevents re-enhancement of the central density.

where the cell index j is to be distinguished from $i \equiv \sqrt{-1}$, k is the wave number, and the superscript is the time-step number. This initial condition corresponds to the physical solution

$$\rho\ (x,t) \ = \ e^{ikx}\ e^{-ikv\ (t-t_0)} \tag{11}$$

an infinite wave propagating to the right. The transport prescription Eq. (8) gives rise to an amplification coefficient

$$\frac{\rho_j^{\,1}}{\rho_j^{\,0}} \ = \left\{1 - \left(\tfrac{1}{4} + \epsilon^2\right)\ \left(1-\cos k\delta x\right)\ - i\epsilon\ \sin k\delta x\right\} \tag{12}$$

The two velocity-dependent terms, proportional to ϵ^2 and ϵ, describe the phase propagation of the wave and include small velocity- and wavenumber-dependent errors in both phase and amplitude of the wave. Without the sin $k\delta x$ term, Eq. (12) looks like a pure three-point diffusion equation. Some of the $\epsilon^2/2$ term is involved in the wave phase change so the actual diffusion in Eq. (12) has a smaller coefficient than $(1/8 + \epsilon^2/2)$. Furthermore, the velocity-dependent part is also wave-number-dependent.

Using Eq. (12) we find the amplification factor:

$$\left|\frac{\rho_j^{\,1}}{\rho_j^{\,0}}\right|^2 \ = \left[\ 1 - \tfrac{1}{4}\ (1\ -\ \cos k\delta x)\right]^2 \ - \frac{\epsilon^2}{2}\ \ (1-2\epsilon^2)\ (1-\cos k\delta x)^2 \tag{13}$$

which is always less than unity for $|\epsilon| < \frac{1}{2}$ (the numerical wave always decays in time and hence the transport scheme is stable). With a velocity-dependent diffusion of the form $(1/8 + \alpha(k)/2)$, we would have

$$\left|\rho_j / \rho_j^0\right|^2 = \left[1 - (\tfrac{1}{4} + \alpha)(1 - \cos k\delta x)\right]^2$$

The effective velocity-dependent diffusion coefficient α then follows from this equation and from Eq. (13).

$$\alpha(k) = \left[\frac{1}{(1 - \cos k\delta x)} - \tfrac{1}{4}\right] - \sqrt{\left[\frac{1}{(1 - \cos k\delta x)} - \tfrac{1}{4}\right]^2 - \frac{\epsilon^2}{2}(1-2\epsilon^2)} \quad (14)$$

The quantity $\alpha(k)$ is always positive, i.e. damping. Thus the effect of the velocity is always to increase the diffusion. When the wavelengths are sufficiently long or ϵ sufficiently small that

$$\left[(1 - \cos k\delta x)^{-1} - \tfrac{1}{4}\right] \text{ is much larger than } \tfrac{1}{2}\epsilon^2 (1-2\epsilon^2),$$

$$\alpha(k) \simeq \frac{\epsilon^2}{4} \frac{(1-2\epsilon^2)}{\left[\frac{1}{(1 - \cos k\delta x)} - \frac{1}{4}\right]} \quad (15)$$

The maximum diffusion coefficient ($k\delta x = \pi$) is

$$\left(1/8 + \frac{\alpha \max}{2}\right) \cong 1/8 + \epsilon^2/2 \quad (16)$$

The analysis of the transport stage of SHASTA so far has considered only the diffusion errors. The phase error in stage I is also a function of wavenumber $k\delta x$ (here identical with the parameter δ defined by Eq. (2)) and velocity ϵ. Define $x_\phi(\epsilon, k)$ to be the position where the imaginary part of ρ_j^1 goes to zero. x_ϕ is zero at $t = 0$ for the exact solution (11) and the numerical solution (10). Thus at $t = \delta t$ the deviation of $x_\phi(\epsilon, k)$ from the analytically exact value $\epsilon\delta x$ measures the phase error as a function of ϵ and k. Setting Im $(\rho_j^1) = 0$ in Eq. (12) with $x_\phi \equiv j\delta x$ gives

$$\tan kx_\phi = \frac{\epsilon \sin k\delta x}{1 - (\frac{1}{4} + \epsilon^2)(1 - \cos k\delta x)} \quad (17)$$

For long wavelengths this gives the relative phase error

$$\frac{x_\phi - \epsilon\delta x}{\epsilon\delta x} \cong k^2 \delta x^2 \left[\frac{\epsilon^2}{6} - \frac{1}{24}\right] + O(k^4 \delta x^4) \quad (18)$$

Equation (18) shows that the algorithm is second-order accurate and that the phase errors reduce for larger velocities, becoming fourth-order accurate when $|\epsilon| = \frac{1}{2}$, the maximum allowable velocity. Equation (17) also shows that the numerical phase velocity is smaller than the actual transport velocity for all wavelengths. The very shortest wavelength, $k\delta x = \pi$, does not propagate at all, according to Eq. (17). Section 3 compares these phase errors with those of other widely-used methods for solving the continuity equation [4].

2. THE ANTIDIFFUSION STAGE (STAGE II)

As seen in the previous section, stage I of the FCT algorithm SHASTA has quite small phase errors for long and intermediate wavelengths but has a large diffusion which is only weakly velocity dependent. In particular, for zero velocity stage I gives effectively the diffusion equation

$$\rho_j^1 = \rho_j + 1/8\,(\rho_{j+1}^0 - 2\rho_j^0 + \rho_{j-1}^0) \tag{19}$$

Removal of this erroneous diffusion by applying an equal and opposite antidiffusion immediately suggests itself. Equation (19) can be inverted in one dimension since it is tridiagonal. Thus we could take the results of stage I $\{\rho_j^1\}$ and find a corrected density $\{\bar{\rho}_j^1\}$ by solving the implicit equation [10].

$$\rho_j^1 = \bar{\rho}_j^1 + 1/8\left[\bar{\rho}_{j+1}^1 - 2\bar{\rho}_j^1 + \bar{\rho}_{j-1}^1\right] \tag{20}$$

This would effectively remove the factor of $\frac{1}{4}$ from Eq. (12) in the $\epsilon = 0$ limit and gives in general

$$\frac{\bar{\rho}_j^1}{\rho_j^0} = \frac{\left[1 - (\frac{1}{4} + \epsilon^2)\,(1-\cos k\delta x) - i\epsilon \sin k\delta x\right]}{\left[1 - \frac{1}{4}(\cos k\delta x\right]} \tag{21}$$

The squared amplitude is

$$\left|\frac{\bar{\rho}_j^1}{\rho_j^0}\right| = 1 - \frac{\frac{\epsilon^2}{2}\,(1-2\epsilon^2)\,(1-\cos k\delta x)^2}{[1 - \frac{1}{4}(1-\cos k\delta x)]^2} \tag{22}$$

The residual diffusive amplitude factor, in this case, has a minimum value $[1 - 8\epsilon^2\,(1-2\epsilon^2)]$.

This implicit approach has several minor drawbacks:

1) It is implicit and hence difficult to generalize to multi-dimensions.

2) It is not positive intrinsically and it would be more expensive to apply the flux-limiting procedure (to be defined shortly) that does make the antidiffusion positive.
3) The result is only slightly better than the following explicit approach which is faster and simpler,

so we concentrate, in this paper, on the following explicit equation:

$$\bar{\rho}_j^{\,1} = \rho_j^{\,1} - 1/8\,(\rho_{j+1}^{1} - 2\rho_j^{\,1} + \rho_{j-1}^{1}) \tag{23}$$

Notice that this antidiffusion step adds only a real multiplicative factor to Eq. (12) and hence does not disturb the phases:

$$\bar{\rho}_j^{\,1} = \left[1 + \tfrac{1}{4}\,(1 - \cos k\delta x)\right]\rho_j^{\,1} \tag{24}$$

The amplitude of the modified $\bar{\rho}_j^{\,1}$ is

$$\left|\frac{\bar{\rho}_j^{\,1}}{\rho_j^{\,0}}\right|^2 = \left[1 - 1/16\,(1-\cos k\delta x)^2\right]^2 - \frac{\epsilon^2}{2}\,(1-2\epsilon^2)\,(1-\cos k\delta x)^2$$

$$\times\left[1 - \tfrac{1}{4}\,(1 - \cos k\delta x)\right]^2 \tag{25}$$

The velocity-dependent term in Eq. (25) is smaller than in Eq. (22) but a small velocity-independent term has been added.

We see that the antidiffusion as given in Eq. (23), is certainly not positive. The simple example of Fig. 3 shows why this is so. The antidiffusion of stage II, which is only intended to remove numerical errors introduced in stage I, in fact introduces additional numerical errors at grid points j and j+1 in the figure. New maxima and minima are formed where they are physically unreasonable. The new minimum is in fact negative.

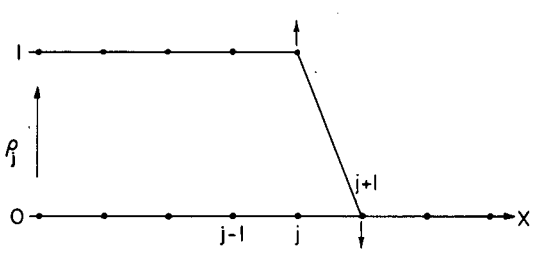


FIG.3. Showing the non-positive tendencies of antidiffusion. At gridpoint j a new maximum will be created by antidiffusion and the new minimum at j+1 will be negative.

The following qualitative limitation on the antidiffusive mass fluxes suggests itself.

The antidiffusion stage should generate no new maxima or minima in the solution, nor should it accentuate already existing extrema.

This qualitative condition obviously limits stage II to just those antidiffusive corrections which are positive. What is not so obvious is how to make this limiting procedure both quantitative and coservative in a simple way. This is done by limiting the size of antidiffusive mass fluxes without allowing their sign to change, thus the name Flux-Corrected Transport.

In more quantitative terms, the antidiffusion stage can be written the following way:

$$\bar{\rho}_j^{\,1} = \rho_j^{\,1} - (f_{j+\frac{1}{2}} - f_{j-\frac{1}{2}}) \tag{26}$$

where the $\{f_{j+\frac{1}{2}}\}$ are the limited antidiffusion fluxes. These are defined to be

$$f_{j+\frac{1}{2}} = \operatorname{sgn}(\Delta_{j+\frac{1}{2}}) \max\Big[0,\ \min(\Delta_{j-\frac{1}{2}} \cdot \operatorname{sgn}(\Delta_{j+\frac{1}{2}}),\ \text{"1/8"}\ |\Delta_{j+\frac{1}{2}}|,$$

$$\Delta_{j+\frac{3}{2}} \cdot \operatorname{sgn}(\Delta_{j+\frac{1}{2}})\Big] \tag{27}$$

where $\Delta_{j+\frac{1}{2}} \equiv \rho_{j+1}^{\,1} - \rho_j^{\,1}$. It is clear immediately from the form of Eq. (26) that this second stage is also fully conservative. The prescription given by Eq. (27) also ensures that the density remains positive and develops no new extreme as a few trials will convince the reader.

The factor 1/8 of Eq. (23) has been replaced by "1/8". The quotation marks indicate that more exact cancellation of errors can be achieved, if one expends a small amount of computational effort by including at least rough approximations to the velocity-and wavenumber-dependent corrections $\alpha(k)$ (Eq. (14)). In the next section, we compare this FCT algorithm with other more-or-less standard schemes. These comparisons, for the most part are made on the simplifying basis that "1/8" = 0.125.

3. THE SQUARE-WAVE TEST PROBLEM

This section compares the one-dimensional FCT algorithm given in the preceding section with three contemporary, explicit, Eulerian, finite-difference techniques. They are the "one-sided" first-order scheme [11], the second-order Lax-Wendroff two-step scheme, and the second-order "leapfrog" scheme. Tables I, II and III show the amplification factors, phase behaviour and long wavelength relative phase error of each algorithm applied to the Fourier harmonic with wavenumber $k \equiv 2\pi/\lambda$. The correct theoretical result is also shown for comparison.

In this section, we assume, as earlier, that v is spatially and temporally constant and define $\epsilon \equiv v\delta t/\delta x$. Then the four algorithms and the exact analytic formula are linear operations except for the flux-correction part of stage II. Each harmonic can be treated independently and should be a

TABLE I. COMPARING THE AMPLIFICATION FACTOR SQUARED FOR FOUR DIFFERENCE SCHEMES AND THEORY. ONLY THE LEAPFROG METHOD, OF THOSE SHOWN, HAS NO DAMPING

Algorithm	Order	Square of the amplification factor*	
Theory	∞	1	
One-sided	1	$1 - 2\lvert\epsilon\rvert(1-\lvert\epsilon\rvert)(1-\cos k\delta x)$	strong damping
Lax-Wendroff	2	$1 - (\epsilon^2-\epsilon^4)(1-\cos k\delta x)^2$	weak damping
Leapfrog	2	1 (exact result)	no damping
SHASTA (FCT)	"2"	$[1 - 1/16\,(1-\cos k\delta x)^2]^2$	
		$-\frac{\epsilon}{2}(1-2\epsilon^2)(1-\cos k\delta x)^2\,[1+\frac{1}{4}(1-\cos k\delta x)]^2$	

* $\{\text{here } \epsilon \equiv \frac{v\delta t}{\delta x}\}$

TABLE II. THE SINGLE-CYCLE PHASE SHIFT FOR FOUR DIFFERENCE SCHEMES AND THEORY. THE EXPRESSIONS ARE VALID FOR ALL WAVELENGTHS WITH FIXED VELOCITY

Algorithm	Order	Phase shift x_ϕ in one cycle*
Theory	∞	$kx_\phi = k\epsilon\,\delta x\,[x_\phi = v\delta t]$
One-sided	1	$\tan kx_\phi = \dfrac{\epsilon \sin k\delta x}{[1 - \lvert\epsilon\rvert(1-\cos k\delta x)]}$
Lax-Wendroff	2	$\tan kx_\phi = \dfrac{\epsilon \sin k\delta x}{[1 - \epsilon^2(1-\cos k\delta x)]}$
Leapfrog	2	$\tan kx_\phi = \dfrac{\sqrt{1 - (1-\frac{\epsilon^2}{2}\sin^2 k\delta x)^2}}{(1-\frac{\epsilon^2}{2}\sin^2 k\delta x)}$
SHASTA (FCT)	"2"	$\tan kx_\phi = \dfrac{\epsilon \sin k\delta x}{[1 - (\frac{1}{4}+\epsilon^2)(1-\cos k\delta x)]}$

* $\{\text{here } \epsilon \equiv \frac{v\delta t}{\delta x}\}$

travelling wave of constant amplitude moving with velocity v (independent of k). Finite-difference algorithms, in general, cause each harmonic to travel at the wrong velocity and cause the amplitude to vary slowly in time.

Consider first Table I for the harmonic amplitudes. The amplification factor is the relative change in the amplitude of the Fourier harmonic during one computational cycle of the given algorithm. Theoretically the amplitude should remain constant for all time. The second order leapfrog method alone achieves this result – not a great surprise since the leapfrog method is reversible and also gives undamped solutions to wave equations. The

TABLE III. COMPARISON OF LONG- AND INTERMEDIATE-WAVELENGTH RELATIVE PHASE ERRORS. THE FCT-ALGORITHM HAS A FOUR TIMES SMALLER ERROR, FOR SMALL VELOCITIES, THAN THE OTHER METHODS

Algorithm	Order	Relative phase error*
Theory	∞	$\frac{x_\phi - v\delta t}{v\delta t} = 0$ (no error)
One-sided	1	" $= -(1/6 - \frac{\lvert\epsilon\rvert}{2} + \frac{\epsilon^2}{3})k^2\delta x^2 + O(k^4\delta x^4)$
Lax-Wendroff	2	" $= -(1/6 - \frac{\epsilon^2}{6})k^2\delta x^2 + O(k^4\delta x^4)$
Leapfrog	2	" $= -(1/6 - \frac{\epsilon^2}{24})k^2\delta x^2 + O(k^4\delta x^4)$
SHASTA (FCT)	"2"	$\frac{x_\phi - v\delta t}{v\delta t} = -(1/24 - \frac{\epsilon^2}{6})k^2\delta x^2 + O(k^4\delta x^4)$

* {here $\epsilon \equiv \frac{v\delta t}{\delta x}$}

other three algorithms give damped solutions with the shorter wavelengths decaying quite rapidly. The one-sided method has particularly strong diffusion. The damping is linear in $|v|$ and only second-order in $k\delta x$. Thus with $\epsilon = 0.2$, a mode of four cells wavelength loses 18% of its amplitude in one cycle. Half of the modes in the system (all those with shorter wavelength) are even more strongly damped.

The Lax-Wendroff two-step method has much lower intrinsic damping, of order ϵ^2 and $k^4\delta x^4$. Thus for the same mode with $\lambda = 4\ \delta x$ the amplification factor is ~ 0.94. Clearly leapfrog is best by this criterion. In practical usage, however, both the Lax-Wendroff and leapfrog algorithms require an additional diffusive smoothing term to keep dispersively generated ripples from pulling the density negative near even moderate gradients. A typical value, per cycle, is ≥ 0.05. Thus both Lax-Wendroff and leapfrog are more diffusive in practice than the new algorithm, which nevertheless guarantees non-negative values because of the flux-correcting procedure. When velocity- and wavenumber- dependent corrections to Eq. (23) are included, or if the implicit antidiffusion is used, the numerical diffusion of the new algorithm can be made almost non-existent.

Table II shows phase behaviour as a function of ϵ and k for the exact solution and for the four algorithms. x_ϕ is the position of the $\mathrm{Im}(\rho) = 0$ point at the end of one cycle, assuming $\mathrm{Im}(\rho) = 0$ at $x = 0$ initially. The theoretical result is $x_\phi = v\delta t$, independent of k. For long wavelength, all expressions have the correct limit, as is to be expected. When $k\delta x$ equals π (2 cells per wavelength), the phase velocity vanishes for all four algorithms.

As an example consider short wavelengths for which $k\delta x = \pi/2$ (four-celled modes). Also take $\epsilon = \frac{1}{2}$. Then

$$\begin{aligned} \tan kx_\phi &= 1, && \text{(theory)} \\ \tan kx_\phi &= 1, && \text{(one-sided)} \\ \tan kx_\phi &= \tfrac{2}{3}, && \text{(Lax-Wendroff)} \\ \tan kx_\phi &= 15/7, && \text{(leapfrog)} \\ \tan kx_\phi &= 1. && \text{(FCT algorithm)} \end{aligned} \tag{28}$$

The new algorithm and the one-sided algorithm give the correct result in this special case. For shorter wavelengths all algorithms give basically nonsensical results.

Table III shows the relative phase error for one cycle for the four algorithms. The relative phase error is defined to be $[x_\phi - v\delta t]/v\delta t$. All four algorithms have phase errors quadratic in $k\delta x$. For most problems of interest the ϵ^2 terms can be neglected since $\epsilon^2 \ll 1$. At long wavelengths the one-sided scheme has a term proportional to $|\epsilon|$ which competes with the 1/6 term where ϵ approaches $\frac{1}{2}$. This competition effectively reduces the relative phase error.

Lax-Wendroff and leapfrog have comparable relative phase errors at long and intermediate wavelength. The last line of Table III shows that the relative phase errors of the FCT algorithm are typically four times smaller than those of the other methods for small velocity at long and intermediate wavelengths. The error becomes fourth order as the velocity gets large ($\epsilon \to \frac{1}{2}$).

Since the one-sided, first-order algorithm has fairly good phase properties even though strongly diffusive, one can apply a velocity-dependent anti-diffusion stage to correct the diffusion problem. This has been done in tests, with good results. The conservative one-sided scheme, however, provides no intrinsic guarantee that positive quantities will remain positive. Dispersively generated ripples and regions of negative density may still get through the anti-diffusion stage to spoil the solution.

In summary, these tables show that the FCT algorithm is also superior to the other methods in regions of x where the solutions are smooth and the flux-corrections unnecessary.

Diffusion behaviour in the simple form using 0.125 for "1/8" is not noticeably better than pure Lax-Wendroff but slightly superior to Lax-Wendroff and leapfrog in their practical form where filtering has been applied to keep the solutions positive. The phase behaviour of the FCT algorithm is considerably better than for the other methods, however, and the flux-limiting correction of stage II gives it excellent additional properties.

To these results should be added the conclusions of Morton who has compared seven different algorithms in somewhat greater detail than we have attempted here [4]. These seven algorithms include the basic methods in common use today. In addition to the three basic methods discussed above, he includes treatments of the Crowley fourth-order method [12], the Crank-Nicholson method [10], the Fromm extension of Lax-Wendroff [13] and the Robert-Weiss fourth-order method [5]. His paper contains two major results:

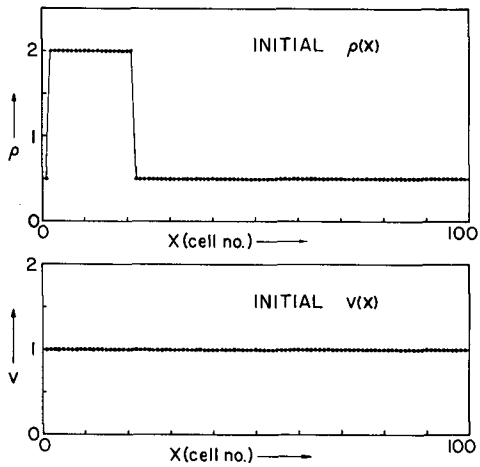


FIG.4. Initial conditions for the square-wave comparison runs. The system is 100 cells long and periodic; the velocity is constant, and $v\delta t/\delta x = 0.2$. The square wave is 20 cells across.

(1) The leapfrog algorithm is the best of the generally applicable basic methods.
(2) Implicit and semi-implicit algorithms of the Crank-Nicholson type should be restricted to use in diffusion equations where an explicit time differencing would result in a prohibitively small timestep to ensure numerical stability.

Computer calculations confirm the conclusions of the analysis given above. Tests were performed on square-waves travelling with constant velocity. All four algorithms of Tables I, II, and III were tested with identical initial conditions and identical timestep. The initial condition in seen in Fig. 4 and the comparison of the four algorithms at two later times is shown in Fig. 5. The square wave is initially 20 cells wide with height 2.0. The background density is 0.5 and constant throughout the rest of the system. The system is 100 cells long with periodic boundary conditions. The velocity was chosen so that $v\delta t/\delta x = 0.2$ for all cases.

The Lax-Wendroff test was performed with a small additional diffusion to keep the solution from becaming negative. The leapfrog case was run with the same level of diffusion and looks appreciably better because undershoots and overshoots are smaller. No undershoots are visible in the one-sided calculation because of the massive diffusion. The SHASTA one-dimensional FCT calculation shows remarkably good agreement with the exact solution when the k-dependent corrections in the factor "1/8" are included approximately. Figure 6 shows a comparison of the SHASTA subroutine results for both the "1/8" corrected calculation and the "1/8" = 1/8 calculation. The correction, called a "steepener", enhances the antidiffusion coefficient in regions where short wavelengths predominate. The uncorrected calculation has no overshoots or undershoots but residual fourth-order diffusion rounds the corners of the square wave somewhat. Even this result is obviously qualitatively far superior to the three basic methods shown in Fig. 5.

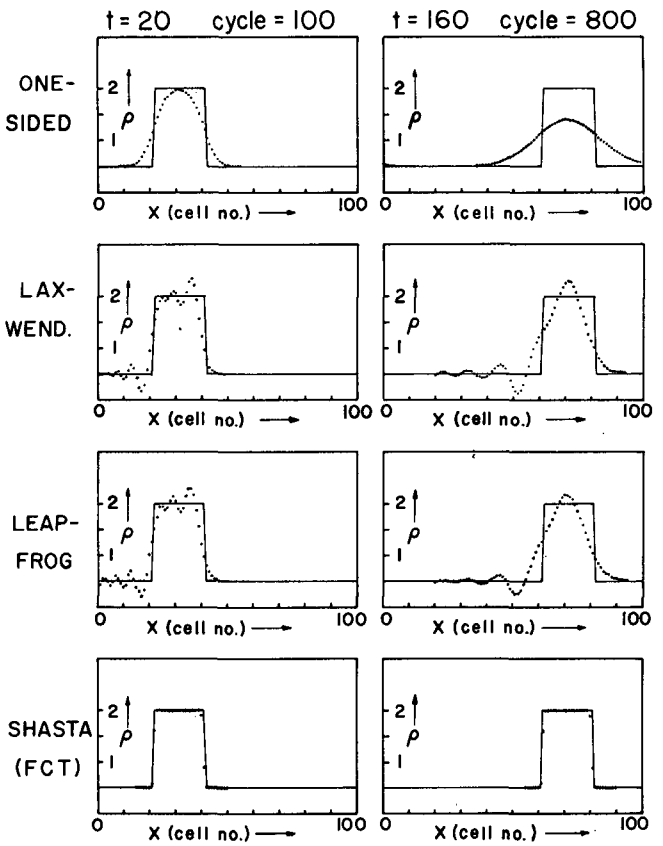


FIG.5. Square-wave test comparisons at two times during the calculation. The solid line is the analytic solution, the dots are computed values. Computed values at the background level of 0.5 are not all plotted.

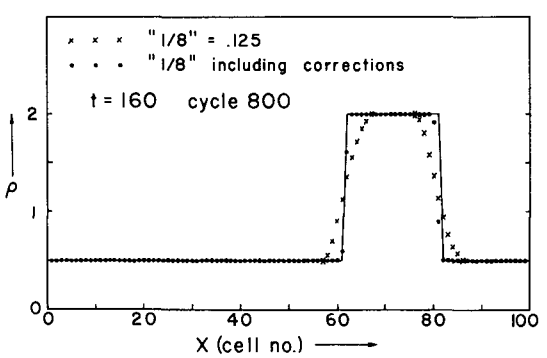


FIG.6. Comparison of the 1 D FCT-algorithm SHASTA with and without k and v dependent corrections to the antidiffusion coefficient. Notice the improvement obtained even by the simple formulation "1/8" = 0.125 in comparison with the standard schemes of Fig.5.

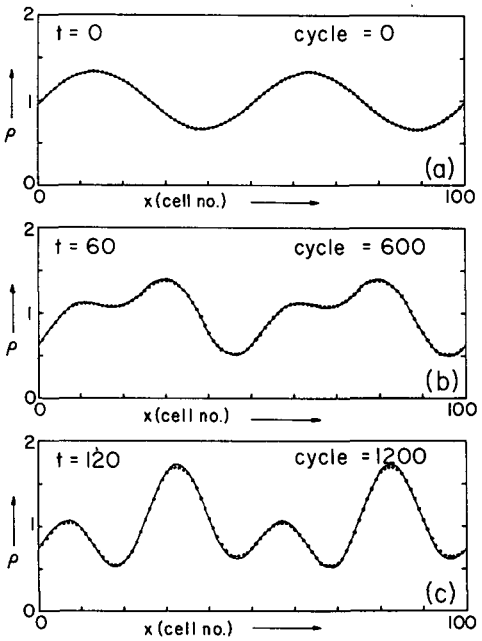


FIG. 7. A test of SHASTA on a problem where new maxima and minima appear physically. Mode 2 is excited initially using an analytic time- and space-dependent velocity and Mode 4 grows with time. The weak clipping phenomenon at maxima and minima is a result of the simple flux-limiting correction of stage II.

Figure 7 shows a further test of the 1 D FCT algorithm. The continuity equation is solved analytically for a time- and space-dependent density of the form

$$\rho(x,t) = 1 + \tfrac{1}{3}\sin\left[k_1 (x - v_0 t)\right] + \frac{2t}{3t_{max}} \sin k_2 x \qquad (29)$$

The corresponding velocity field is:

$$v(x,t) = \frac{1}{\rho}\left[\tfrac{1}{3} v_0 \left\{\sin\left[k_1(x - v_0 t)\right] + \sin k_1 v_0 t\right\} + \tfrac{2}{3}\frac{1}{k_2 t_{max}}\left\{\cos k_2 x - 1\right\}\right] \qquad (30)$$

where v_0, k_1, k_2, and t max are constant and $\rho(x,t)$ is given by Eq. (29).

This exact velocity was used in the SHASTA subroutine to solve the continuity equation numerically for the density. Analytic and computed solutions for the density are plotted for comparison at three times. This

problem has a growing sinusoidal density component and a travelling sinusoidal component. Their interaction creates a situation in which new maxima and minima are continually forming and moving physically. This test would also be handled reasonably well by the leapfrog and Lax-Wendroff algorithms. It shows that flux-limiting, as performed in stage II of the algorithm, does not preclude the appearance and disappearance of extrema in the solution when they occur physically.

Great progress in treating full sets of fluid equations on strong shock problems has been made recently, two-dimensional versions of FCT algorithms have been devised and tested, and a cylindrical MHD-model has been programmed. These algorithms and calculations will be published in a series of three articles in the near future.

4. CONCLUSIONS

A new class of algorithms for solving continuity and continuity-like equations has been presented. It features a second-order treatment of the advective terms, is conservative, and is non-negative. The FCT-algorithm for one-dimension, as employed in a program called SHASTA, consists of two distinct stages. The first, or transport, stage solves the continuity equation by a (linear interpolation) three-point formula which has an appealing geometric interpretation. The second, or antidiffusion stage, corrects the numerical errors introduced during the first stage.

The strong diffusion of stage I, coupled with the antidiffusion and simple flux-correcting procedure of stage II makes the new FCT algorithms work. The flux correction allows the overall algorithm to remain non-negative and stable while reducing spurious numerical diffusion, the usual stabilizing element of most methods, to a very low level. The condition that no new maxima or minima be generated by the antidiffusion of stage II seems to be the crucial factor. The exact details of the stage I algorithm seem to be of only minor importance as long as the effective diffusion is sufficiently great to ensure that the transport of stage I is non-negative. This means qualitatively that the diffusion introduced in stage I must be larger than any dispersive error. Then the local residual diffusion, taken to be the diffusion of stage I minus the limited and hence smaller antidiffusion of stage II, can always be large enough, in principle, to cancel the dispersion. That is, the FCT algorithms leave a large residual diffusion behind locally which is equal and opposite to the local dispersion error. Both terms are of zero order but combine to give an accurate solution.

Although the particular version of stage I used in the SHASTA subroutine has excellent phase properties to recommend it, there are many adequate transport algorithms with sufficient diffusion. The basic conservative one-sided method may not possess adequate diffusion on its own to remain positive, but this additional diffusion could be added externally. This approach has also been tried with the leapfrog method with good results. Hain has pointed out that the Lax-Wendroff two-step method with an additional diffusion is equivalent to the stage I transport algorithm in the special case of constant velocity.

The original motivation for developing this new algorithm arose from considerations of large-scale multi-fluid, multi-dimensional calculations on computers just now becoming available. It is clear that the CPU-speeds

of the newer machines are increasing faster than the size of the fast core storage. Therefore, if one does not wish to buffer sections of the calculation into and out of core from external storage, improvements in accuracy will not come principally from increases in resolution. The necessary improvements will come from achieving a greater accuracy per grid point by expending more CPU-cycles. By this argument the FCT algorithms will be used to improve accuracy or the newer faster machines without requiring longer or larger runs than on the slower existing machines.

Thus it is appropriate to close this paper with a few timing considerations. In the simple 1 D forms used for the tests of section 3, the Lax-Wendroff method is marginally slower than the leapfrog method; the one-sided method, without tests, is 1.4 times slower than leapfrog, and the SHASTA subroutine is ~2.9 times slower. When effective resolution is taken into account, however, the FCT-algorithms regain this factor easily. In a shock problem, e.g. comparable calculations using one of the basic methods with a von-Neumann viscosity would require roughly 3-5 times more cells to achieve comparable resolution. This required decrease in δx implies a corresponding decrease in δt. Thus 9-25 times more grid-point-time-steps would have to be performed using a standard method than would be required of the FCT algorithm. This means, in practical calculations, involving steep gradients that a given error tolerance can be achieved by FCT in 3-8 times less computer execution time. The savings in multi-dimensions could be even more substantial. An even more compelling argument for the FCT approach can be derived by considering strong-shock problems. The use of a sufficiently large artifical viscosity to suppress ripples at the shock in conventional algorithms usually introduces a severe diffusive stability restriction on the time-step because the viscous diffusion dominates the flow. Thus, even on grids with the same δx, the FCT algorithms may consume less computer time by taking longer time-steps – and give a much better answer in the bargain.

ACKNOWLEDGEMENTS

I would like to acknowledge discussions on this method with D.L. Book, K. Hain, C.E. Wagner, B.E. McDonald and D.V. Anderson of the Naval Research Laboratory. The substance of this more recent joint research will be contained in a series of three collaborative papers which expand the applicability of the SHASTA FCT method to other difference schemes, to multi-dimensions, and to complex physical problems. I would particularly like to thank David Book for his help in preparing this manuscript and in all phases of FCT-research since joining NRL.

REFERENCES

[1] RICHTMYER, R.D., MORTON, K.W., Difference Methods for Initial Value Problems, Interscience Publishers, New York (1967).

[2] Methods in Computational Physics 3 Fundamental Methods in Hydrodynamics (ALDER, B., FERNBACH, S., ROTENBERG, M., Eds), Academic Press, New York (1964).

[3] LAX, P.D., WENDROFF, B., Systems of Conservation Laws, Communs pure appl. Math. 13 (1960) 217.

[4] MORTON, K.W., Stability and Convergence in Fluid Flow Problems, Proc. R. Soc. A323 (1971) 237.
[5] ROBERTS, K.V., WEISS, N.O., Convective Difference Schemes, Maths Comput. 20 (1966) 272.
[6] HALTINER, G.J., Numerical Weather Prediction, Naval Weather Research Facility Report NWRF 30-0768-142, July 1968.
[7] BORIS, J.P., BOOK, D.L., Flux-Corrected Transport – I. SHASTA, A Fluid Transport Algorithm That Works (to be published), this paper contains an expansion of the present material and an extension to complete systems of fluid equations. It is the first of three articles in a series.
[8] EMERY, A.F., An Evaluation of Several Differencing Methods for Inviscid Fluid Flow Problems, J. Comput. Phys. 2 (1968) 306.
[9] von NEUMANN, J., RICHTMYER, R.D., A Method for the Numerical Calculations of Hydrodynamical Shocks, J. appl. Phys. 21 (1950) 232.
[10] CRANK, J., NICHOLSON, P., A Practical Method for Numerical Integration of Solutions of Partial Differential Equations of Heat Conduction Type, Proc. Camb. phil. Soc. 43 (1947) 50.
[11] One-sided difference schemes of the type we test here are attributed to R. Le Levier by Richtmyer and Morton (1953).
[12] CROWLEY, W.P., Mon. Weath. Rev. 96 (1968) 1.
[13] FROMM, J.E., A Method for Reducing Dispersion in Convective Difference Schemes, J. Comput. Phys. 3 (1968) 176.

PART III: QUANTUM COMPUTATIONAL PHYSICS

STATISTICAL METHODS FOR BUBBLE CHAMBER ANALYSIS

E. LOHRMANN
Deutsches Elektronen-Synchrotron DESY,
Hamburg, Federal Republic of Germany

Abstract

STATISTICAL METHODS FOR BUBBLE CHAMBER ANALYSIS.
1. Introduction; 2. Display of data; 3. Estimating parameters from experimental distributions; 4. Maximizers (minimizers).

1. INTRODUCTION

Bubble chamber pictures contain a large amount of information. After the care taken in measuring and reconstructing events in the bubble chamber, one wants to retrieve as much of this information as possible in terms of quantities of immediate physical interest. This is done with the help of programs which can display and statistically analyse large samples of bubble chamber data. These programs consume more than half of the total computer time devoted to bubble chamber analysis. So, although most of them are, in principle, quite simple, they have to be engineered very well to be optimally adapted both to the user's needs and to the computer installation.

2. DISPLAY OF DATA

As a first step in the statistical analysis one wants to look at one- and two-dimensional frequency distributions of experimental quantities. We shall describe two program systems which accomplish this. The following difficult points have to be solved by these programs: handling large amounts of input (typically up to ~100 tapes), making optimum use of available computing resources (storage space, peripherals, software assistance), and making them easy to modify and to use. The last point is most important. For programs which are meant for a large number of users, the interface between man and program requires careful engineering and a knowledge of the psychology of the physicist.

2.1. Examples of plotting one- and two-dimensional distributions

This is to give a few examples of the type of output of plotting programs. Consider, e.g. a bubble chamber experiment, which has yielded a number of events of the following reaction: $\gamma p \rightarrow p\,\pi^+\pi^-$.

One wants to study resonance production in the $\pi^+\pi^-$ system. The program should therefore take the following steps: Take the experimental data, which in the simplest case will be just the four-momenta of all the particles, calculate the $\pi^+\pi^-$ invariant mass

$$M^2_{\pi\pi} = (E_{\pi^+} + E_{\pi^-})^2 - (\vec{P}_{\pi^+} + \vec{P}_{\pi^-})^2$$

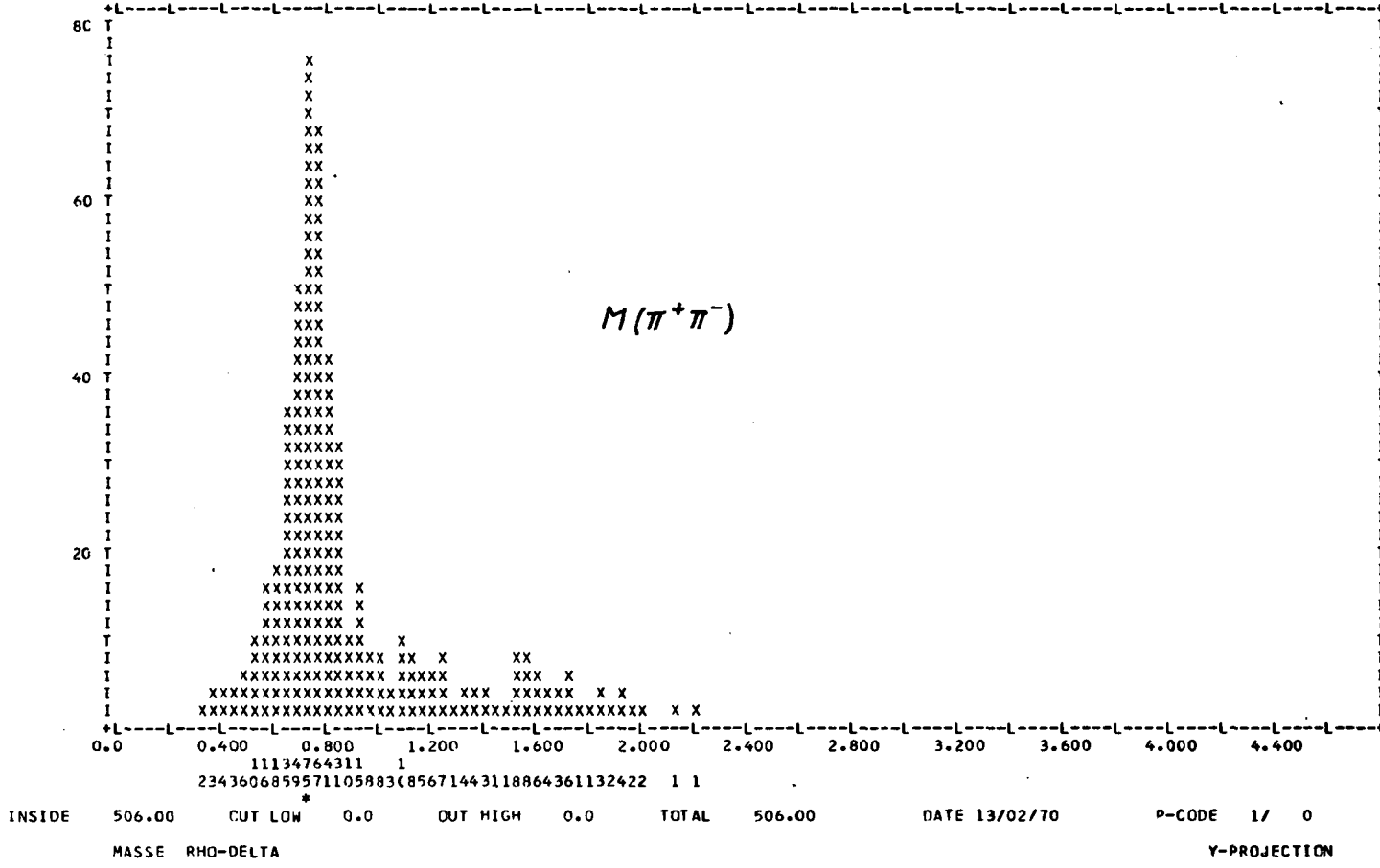


FIG. 1. Mass distribution of $\pi^+\pi^-$ from the reaction $\gamma p \to p\pi^+\pi^-$ at 4.5 GeV. Data from the Aachen-Hamburg-Heidelberg-München Streamer Chamber Collaboration.

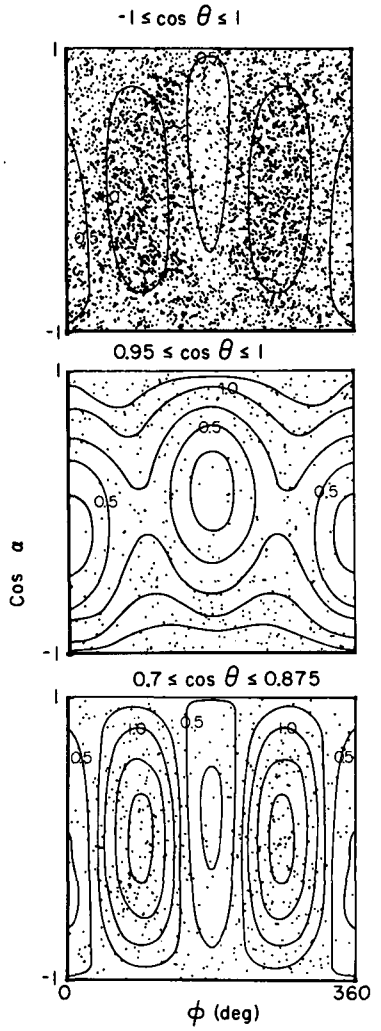


FIG. 2. Scatter plot of polar cosine versus azimuth angle of $K^*(890)$ decay, observed in the reaction $K^-p \to K^0p\pi^-$, shown for three slices in the cosine of the production angle θ (from Ref. [4]).

and plot the distribution of the masses of the $\pi^+\pi^-$ system. Figure 1 shows this distribution, which demonstrates the presence of the ρ^- resonance.

In a two-dimensional or scatter plot two quantities are plotted against each other, each event being represented by one point. Figure 2 shows such a plot, displaying the decay distribution (polar versus azimuthal angle) of the K^* resonance decay, observed in the reaction $K^-p \to K^0 p \pi^-$. These plots can be produced conveniently by a CRT display and photographed. However, for large numbers of events, these pictures become messy and

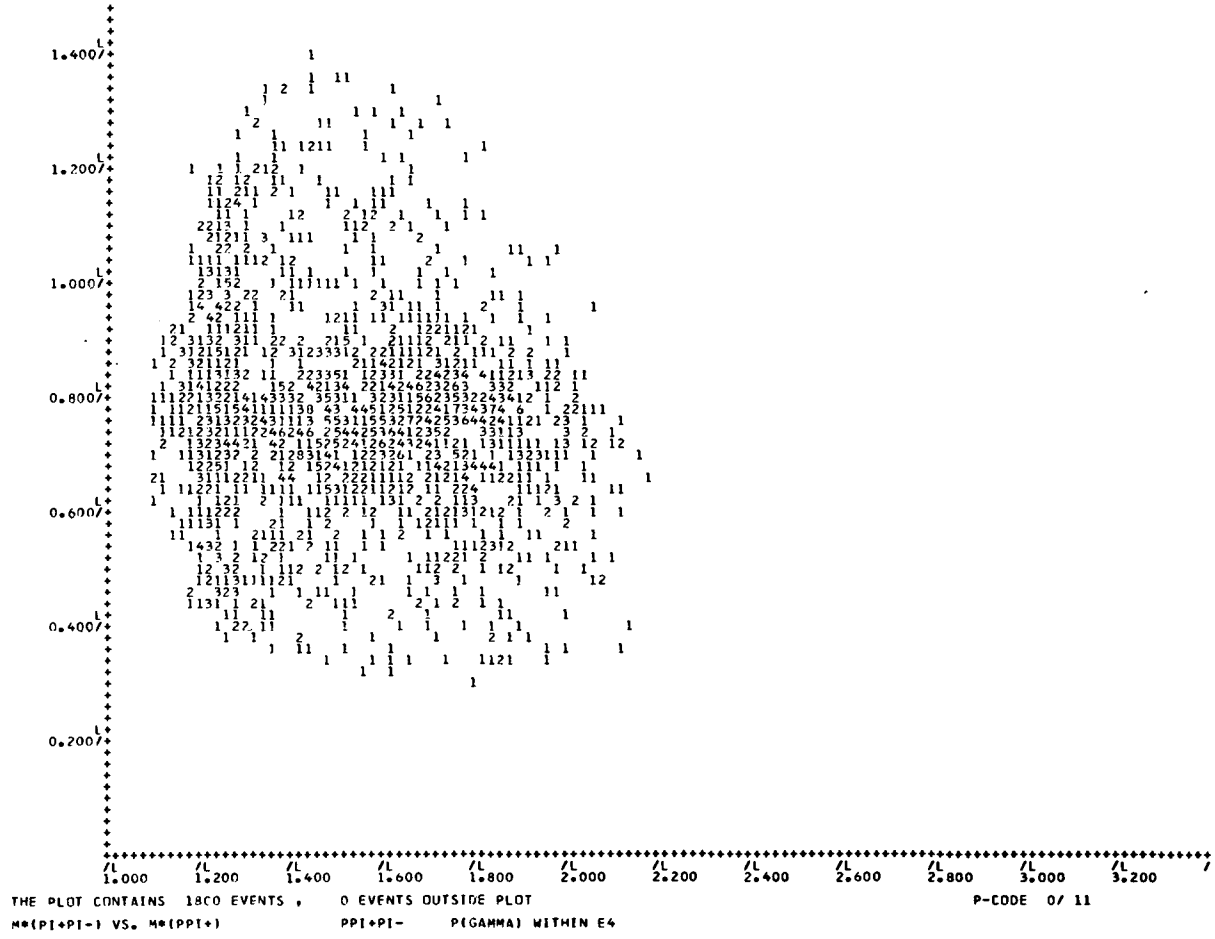


FIG. 3. Scatter plot of the masses ($\pi^+\pi^-$) versus ($p\pi^+$) from the reaction $\gamma p \to p\pi^+\pi^-$ at 2 GeV, showing the ρ and Δ resonance bands. Data from the Aachen-Bonn-Hamburg-Heidelberg-München Bubble Chamber Collaboration.

hard to evaluate numerically. Then it is better to represent the two-dimensional distribution by printing the number of events in a two-dimensional grid of points on a line-printer. Figure 3 shows an example.

We shall now describe two program systems which have been designed to plot data. It should be emphasized that these programs are completely general. They can be used and have been used outside the bubble chamber field.

2.2. SUMX

SUMX was first designed at the UCRL at Berkeley.

A modified, further developed and well documented version exists at CERN [1].

SUMX reads a data summary tape (DST) (or possibly other tapes) event by event into a COMMON area. One event is formally defined to be just a block of words. For specified words in each event SUMX can make histograms, two-dimensional scatter diagrams, ideograms, output lists and ordered lists and calculate mean values and variances. SUMX works through a number of processors = blocks = subroutines, which are controlled by a main program and the information supplied by control cards. The following are examples of a few of these subroutines:

TAPE	reads input DST and stores one event in COMMON BOUT (Xloc)
BLOCK 4	manipulates, combines histograms made by BLOCK 6
BLOCK 6	makes histograms and ideograms
BLOCK 7	makes two-dimensional scatter plots
BLOCK 13	makes a mini-DST
SELECT	defines truth-function by means of specified tests on the data
CHARM	allows the user to include subroutines for auxiliary calculations.

Of course SUMX contains many more processors than those described here. Example of a set of control cards to handle a problem:

```
*   NEW PASS                430,0                                   1)
    EXAMPLE PASS
*   TAPE 20                                                         2)
    11                      DST             EXP 93                  3)
*   SELECT                                                          4)
    TEST 3
         124                BETWEEN 12.     15.
*   BLOCK       6                                                   5)
    'TITLE OF HISTOGRAM
     30         1           9.5                                     6)
    210                                                             5)
    220                                                             5)
    'TITLE OF NEXT HISTOGRAM
     50         2.5         -10             3                       7)
    430.
*   ALL DONE
```

Comments:

1) Only the first 430 words of each event are of interest.
2) Only 20 events are read from DST (for debugging purposes).

3) Logical tape number 11.
4) SELECT is invoked. It defines a test number 3 as having the value = true, if the 124th word is between 12 and 15, i.e. $12 < BOUT(124) < 15$. Test 3 has the value = false otherwise.
5) BLOCK 6 is involved to plot the words BOUT (210) and BOUT (220).
6) The histogram has to have 30 bins, of width 1, with starting value 9.5.
7) Similar to 5) and 6), but here BOUT (430) is only plotted, if test 3 = true.

The following facilities of SUMX merit special comment:

1) SELECT:

SUMX offers a very concise way to define conditions under which words are included in plots. It is done by defining a truth-value for the outcome of a number of tests, which are identified by a test number NT. The truth-values for each event are evaluated for all test numbers NT and are stored for each event in a test vector. This test vector can be included in the DST.

Example:

```
TEST 31
      15 BETWEEN  71.31          73.85
      25 BETWEEN  10.2           11.0
AND   39 BIGGER   63              19.2
```

This means: TEST 31 = true, if

$$((71.31 < BOUT(15) < 73.85) .OR. (10.2 < BOUT(25) < 11))$$

$$AND . (BOUT(39) > (BOUT(63) + 19.2))$$

TEST 31 = false otherwise.

2) BLOCK 13:

This routine allows producing a smaller DST, containing only those events which are actually used for plotting, and only those words for each event, which are of interest for plotting.

This can greatly reduce costly tape-handling.

3) The following facilities are, among other things, interesting:

A facility to produce a series of similar control cards, a diagnosis facility which identifies all formally incorrect control cards, a facility to include new information on the DST and a facility to define multiple tests.

4) Storage space:

SUMX needs about 25K storage space. In addition, it needs a working space to assemble the one- and two-dimensional plots. This can be a serious limitation since runs requiring more than 100 such plots are not unusual. The problem is solved by 1) giving SUMX a maximum of storage space by dynamical storage allocation and 2) breaking a SUMX pass up into several physical passes if necessary. Each pass produces a number

of plots allowed by the available storage space, then the DST is read again, more plots are made, etc. This is a costly process and encourages a policy to include all information ever to be plotted on the DST and not to rely too much on the CHARM routines.

2.3. HYBRID

HYBRID was developed by the Hamburg bubble chamber group, mainly by H. Butenschön [2].

Its main two differences to SUMX are the following:

1) Rather than having a set of fixed subroutines inside the system, the user must write his own subroutines to read in, test, manipulate and plot data. This makes the use less concise, but offers more versatility.
2) The storage problem is handled differently, as shown below. HYBRID needs only storage space for one one-dimensional and one two-dimensional plot. This allows more generous use of CHARM-like subroutines to do auxiliary calculations on the data and encourages the use of a smaller amount of input data, which may be of interest, if tape handling becomes a bottleneck.

HYBRID works through the following four steps:

1) Read input tape, select data to be plotted, write new tape.
2) Read new tape, calculate plot indices (i.e. bin numbers) and plot number (i.e. number of plot) for each event, combine the numbers in one computer word. Each word contains information to store one event in one plot. Output on new tape, blocked to 1500 words.
3) Use IBM sorting routine SORT to sort these words according to plot number.
4) Build up plots, one plot at a time, change of plot number signals output of plot.

The saving of time accomplished is shown by the following example: To produce 180 two-dimensional plots with 8000 events took 540 minutes on the IBM 7044 for a straightforward procedure, and only 40 minutes with the procedure described above.

HYBRID offers the high-energy physics user the following facilities: Four-vectors of particles can be stored in COMMON P (100, 10). The first index I of P (I, K) identifies the particle by its number. The second index K means the components of the four-vector: K = 1: P_x, K = 2: P_y, K = 3: P_z, K = 4: E (total energy), K = 5 : M (mass).

The following subroutines are, among others, available:

CALL ADD (N1, N2, N3): Adds the four-vectors of particles stored in P(N1, K) and P(N2, K) and stores the resulting four-vector in P(N3, K).

CALL ANG (N1, N2, ACOS, PHI): calculates the angle PHI, ACOS = cos (PHI) between the three-momenta of particles N1 and N2.

CALL CROSS (N1, N2, N3): Same for the cross-product.

CALL DOT (N1, N2, D)
D = dot-product between N1 and N2.

CALL LENGTH (N1, S) calculates length S of three momentum of particle N1.

CALL LOR (N1, N2, N3) makes Lorentz-transformation of particle N1 into rest system of particle N2, stores the result in N3.

CALL PLOT (A, M1) enters A into histogram number M1.

CALL DPLOT (A, B, M2) enters A and B into two-dimensional scatter plot number M2.

Example programs:

1) input

```
      SUBROUTINE RBT (ENDE)
      COMMON A(100)
      DATA EOF/3 HEOF/, M/0/
      IF (M.EQ.50) GO TO 200
      READ (10) A
      CALL CALCUL
      M = M + 1
100   RETURN
200   ENDE = EOF
      GO TO 100
      END
```

This program will read 50 events into the array A. Each event is modified by CALCUL.
One could also omit the READ statement and use CALCUL to produce a simulated input of Monte-Carlo events.

2) Suppose we have a reaction with the particle numbers chosen as follows:

$$K^+(1) + p(2) \rightarrow \pi^+(3)\, p(4)\, K^+(5)\, \pi^-(6)$$

Suppose one wants to look at the scatter plot of Mass (π^+p) vs. Mass ($K^+\pi^-$) to study Δ -K^* production, and one wants the $K^+\pi^-$ mass distribution for events which have possibly also a $\Delta(1236)$ produced. This is accomplished by the following program:

```
SUBROUTINE WAHL 1
COMMON P(100,10)
CALL ADD (3,4,7)
CALL ADD (5,6,8)
CALL DPLOT (P(7,5), P(8,5), 1)
IF (P(7,5).LT.1.45) CALLPLOT (P(8,5), 1)
RETURN
END
```

3. ESTIMATING PARAMETERS FROM EXPERIMENTAL DISTRIBUTIONS

This is generally called 'curve fitting'. The following is just to remind the reader of the most important statistical formulas used. Those not familiar with the methods should consult the literature [3].

3.1. Maximum likelihood

Let f(x, a) be a distribution density of the (experimental) quantity x, containing a parameter a, and properly normalized:

$$\int f(x,a)\, dx = 1$$

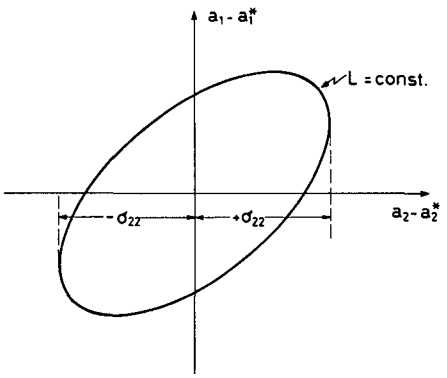


FIG. 4. Definition of standard deviation σ_{22} of the parameter $a_2 L_{max} = L(a_1^*, a_2^*)$, the line is L = const = L_{max} $-(1/2)(\log_e$ used for calculating L).

If a number of (independent) observations is made, yielding values $x_1, x_2, \ldots x_n$, the joint probability is

$$P(x_1, \ldots) = \prod_{i=1}^{n} f(x_i, a) = P(a)$$

The best estimate of a is given by the condition P(a) = maximum. In practice, one prefers to use the likelihood function

$$L(a) = \log P(a) = \sum_{i=1}^{n} \log f(x_i, a)$$

The best estimate of a is given by L(a) = max.
The variance of a (at its most likely value $a = a^*$) is given by

$$\sigma^2 \cong -\left(\frac{d^2 L}{d\, a^2}\right)^{-1}_{a=a^*}$$

These results can be generalized if L depends on several parameters $a_1, a_2, \ldots a_r$. Let us introduce the vector of a-values $\vec{a}^T = (a_1, a_2, \ldots a_r)$ and of x-values $\vec{x}^T = (x_1, x_2, \ldots x_n)$. T = transposed matrix. Then we have $L(\vec{x}, \vec{a})$. The vector $\vec{a}$ is determined by the condition $L(\vec{a})$ = maximum. If $L(\vec{a})$ can be differentiated everywhere, this amounts to

$$\partial L / \partial a_i = 0, \; i = 1, 2, \ldots \text{ or } \nabla_a L = 0$$

The error matrix is given by

$$\sigma^2_{\ell m} = -\left[\frac{\partial^2 L}{\partial a_i \partial a_k}\right]^{-1}\Bigg|_{\ell m} = V_{\ell m}$$

The meaning of the error of a_2 is exemplified in Fig. 4 for the case of two parameters a_1, a_2.

3.2. Method of least squares

Suppose that the values y of a measurement are distributed around a curve $\eta = \eta(x, \vec{a})$, where $\vec{a}$ is a set of parameters and x an independent variable fixed by the conditions of the experiment. More precisely, assume that

$$\langle y(x) \rangle = \eta(x, \vec{a})$$

and

$$\eta(x, \vec{a}) = \sum_{i=0}^{k} f_i(x) \cdot a_i$$

Call the residuals

$$\epsilon_i = y_i - \eta(x_i, \vec{a})$$

Then for a wide class of problems a best estimate of the parameters a is given by the condition

$$S = \sum_{i=1}^{n} \epsilon_i^2 = \text{minimum}$$

If the ϵ_i have a Gaussian distribution, this follows from maximum likelihood (prove this). Introduce the matrices

$$[A] = \begin{bmatrix} f_0(x_1) & f_1(x_1) & f_2(x_1) \ldots \\ f_0(x_2) & f_1(x_2) & f_2(x_2) \ldots \\ f_0(x_3) & f_1(x_3) & f_2(x_3) \ldots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

and

$$\vec{\epsilon} = \vec{y} - A \cdot \vec{a}$$

Then

$$S = \vec{\epsilon}^{T} \cdot \vec{\epsilon}$$

From $\nabla_a S = 0$ follows

$$\vec{a} = (A^T \cdot A)^{-1} \cdot A^T \cdot \vec{y}$$

variance of

$$a = \sigma^2 \cdot (A^T \cdot A)^{-1}$$

where σ^2 is the variance of ϵ (considered constant).

3.3. Simple example (from W. P. Swanson [3])

Consider a double scattering experiment: $\pi p \rightarrow \pi p$.

The proton spin is analysed through a second scatter on carbon. Let θ, ϕ be the polar, azimuth angles of the second scatter. The probability of the second scatter is

$$P_i = (1 + p \cdot a(\theta_i) \cos \phi_i)/2\pi$$

where p is the (unknown) polarization of the proton, $a(\theta_i)$ the analysing power of carbon, *i.e. known function of* θ.

Suppose that we have observed a number of events N. Then the best estimate of the proton polarization p according to maximum likelihood is simply given by

$$L(p) = \sum_{i=1}^{i=N} \log(1 + p \cdot a(\theta_i) \cos \phi_i)$$

For a fixed experimental sample one has to find that value of p, which makes L(p) = max. This can, in principle, be done by calculating L(p) for many values of p and looking where the maximum is. Methods of solving problems of this kind will be discussed below.

With the method of least squares we would have to group our events into boxes of θ_i, ϕ_i values. Call $N(\theta_i, \phi_i)$ the number of events observed in a box centred at θ_i, ϕ_i. Then one would minimize the expression

$$X^2 = \sum_{\text{sum over bins}} \frac{(N(\theta_i, \phi_i) - N \cdot P_i(\theta_i, \phi_i))^2}{N(\theta_i, \phi_i)}$$

This leads to a straightforward determination of p and its error, whereas with the maximum-likelihood method it can be a tedious undertaking to find the maximum of L. However, the maximum-likelihood method is free of the arbitrariness in deciding on the boxes to group the data. Since it treats each event individually, it does not have the difficulty of the least-squares method either, where the number of events in each box must exceed a minimum value (usually ten).

3.4. Sophisticated example [4]

Consider the reaction $K^- p \rightarrow K^0 \pi^- p$ (4300 events).

One wants to study the K^* decay distribution ($K^- p \rightarrow K^{*-} p$) in the presence of other background reactions (like $K^- p \rightarrow \Delta^0(1236)\, K^0$, $K^- p \rightarrow Y_1^{*+}(1600)\, \pi^-$, etc. The probability function is

$$P(x, \hat{k}, \vec{r}, a, b, c) = \sum_i \frac{r_i}{N} \cdot BW(E_i, \Gamma_i) + \frac{1 - \Sigma r_i}{N_p} + \frac{r_{K^*}}{N_{K^*}} \cdot BW(E_{K^*}, \Gamma_{K^*})$$

$$\times [1 + a \mathscr{Y}_2^0(\hat{k}) + b \mathrm{Re} \mathscr{Y}_2^1(\hat{k}) + c \mathrm{Re} \mathscr{Y}_2^2(k)]$$

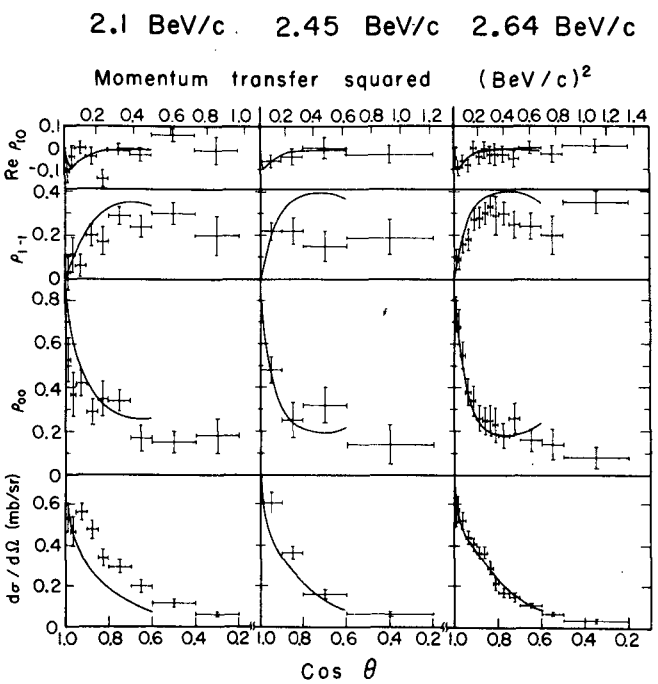


FIG. 5. Differential cross-section and spin-density-matrix elements as a function of K^* production angle in the reaction $K^- p \to K^{*-} p$ for three incoming momentum values (from Ref. [4]). The solid curves are predictions from a model calculation.

r_i = fractions of the number of events coming from background reactions ($\Delta(1236)\bar{K}^0$, $N^*(1688)\bar{K}^0$, $\mathscr{Y}_1^*(1600)\pi^-$, $\mathscr{Y}_1^*(1765)\pi^-$), $N^*(1512)\bar{K}^0$

N_i = total phase space for i-th resonance

N_p = total three body phase space for the event

BW = Breit-Wigner resonance form of the energy E_i and width Γ_i

a, b, c = linear functions of the K^* decay spin density matrix elements ρ_{ik}

$\hat{k}$ = unit vector in the direction of the π^-

Each event is entered into P with its appropriate values of the invariant masses into the BW expressions and the value of $\hat{k}$ into the spherical harmonics. The likelihood function is then maximized for the 9 parameters r_i, r_{k^*}, a, b, c. The resulting values of a, b, c are used to calculate the spin density matrix elements ρ_{ik} of the K^* decay. Actually the procedure was done for various intervals of the K^* production angle θ and the results displayed as a function of θ. This is shown in Fig. 5. For fitting procedures of this complexity it is important to check that the probability distribution with the fitted values of the parameters actually is a good fit to the data. A comparison between the data and the predicted distribution of the K^* decay angles, as calculated from the fitted values of a, b, c is shown in Fig. 6 (see also Fig. 2).

It indicates that the fit is indeed a good description of the data.

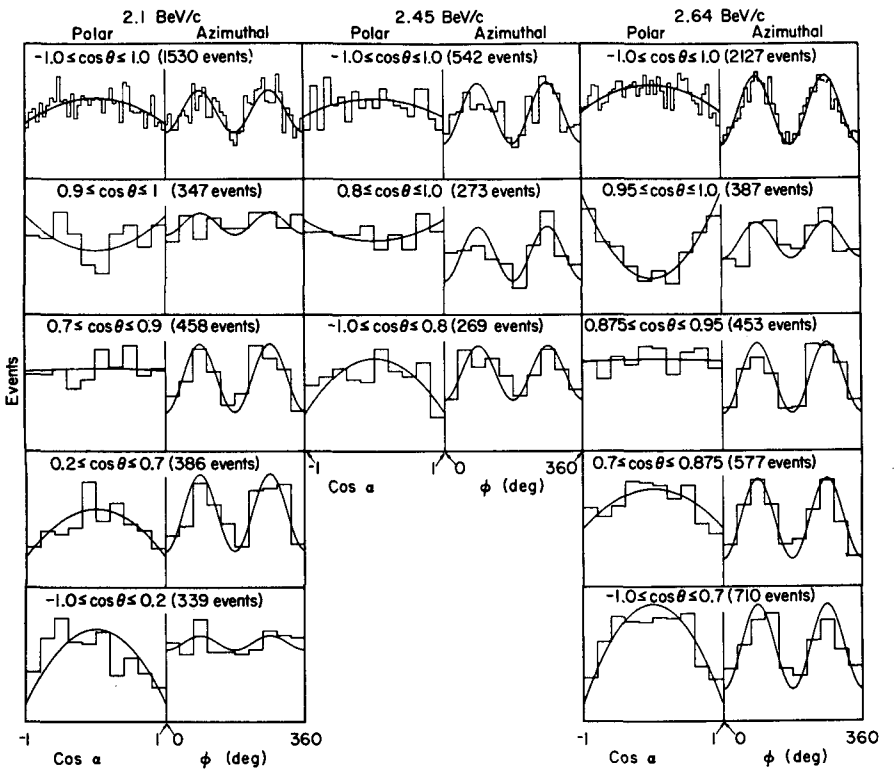


FIG. 6. Polar cosine and azimuth decay-angle distributions of K^{*} (890) at three incoming momentum values and various intervals in production angle. The solid curves are the distributions predicted from the maximum-likelihood solution for the parameters a, b, c (from Ref. [4]).

4. MAXIMIZERS (MINIMIZERS)

An important step in applying the method of maximum likelihood consists in finding an extremum for the likelihood function $L(\vec{a})$, which may depend on many variables. Owing to the wide range in the possible behaviour of these functions, there is no single algorithm, giving an optimal solution in all cases. Even for a given single problem, the character of $L(\vec{a})$ may change as one approaches the maximum.

4.1. MINUIT

An example for the use of different strategies depending on the character of the function is offered by the program MINUIT. It operates in three steps:

1) If $L(\vec{a})$ is very irregular, one just samples the a-parameter space at a number of points in order to find an extremum. To this effect values of all parameters a are chosen randomly by taking them from a Gaussian distribution centred at the starting value of a, with a width = starting parameter error.

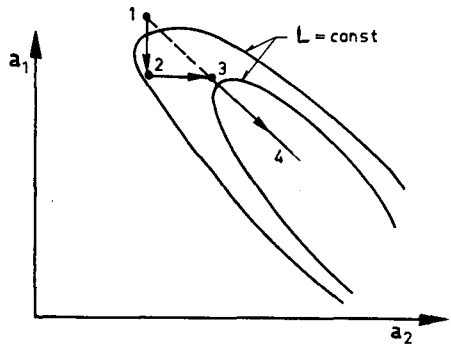


FIG. 7. MINUIT program strategy.

2) If $L(\vec{a})$ is more regular, i.e. $\Delta_a L$ exists, one can use a 'ravine search' method with Rosenbrook's method to locate the extrema [5]. The method is explained below for two variables and $-L(\vec{a})$, i.e. looking for a minimum (Fig. 7).

One starts at some point (1) and varies a_1 for a minimum of $F(a_1, a_2 = \text{fixed})$.: point (2). Next one keeps a_1 fixed and varies a_2 for a new minimum of L:point (3). The line (1-3) defines the approximate direction of a ravine, along which the search is continued. Algorithm to find the minima:

1) Go from $L(a_k)$ to $L(a_k + S)$.
2) If $L(a + S) < L(a)$: : 'success' : replace a_k by $a_k + S$, replace S by dS, go to 1).
3) If $L(a + S) > L(a)$: 'failure' : replace S by $\beta \cdot S$, go to 1). Repeat this until one gets a sequence success-failure, then use the last three points to predict the minimum of a parabola.

MINUIT uses the parameters $\alpha = 2$, $\beta = -0.3$,

3) If $\nabla L(\vec{a})$ and the second derivatives exist and are well behaved, one uses the Davidon's [6] variable matrix algorithm. It has the following steps:

1) based on current estimate of $[V] = [\partial^2 L / \partial a_i \, \partial a_k]^{-1}$ and on ∇L calculate a new minimal value of L, evaluate ∇L there.
2) Form the vector $\vec{r} = [V] \cdot \nabla \vec{L}$, and $\rho = \nabla \vec{L}^T \cdot \vec{r} \cong$ vertical distance to minimum, to check approach to minimum and convergence.
3) New estimate of [V] from old [V] and new ∇L, avoids inversion of derivative matrix $[\partial^2 L / \partial a_i \, \partial a_k]$.

4.2. MLFIT

With this program [7] as an example we explain in some more detail how one can look for an extremum if first and second derivatives are available. Expand $L(\vec{a})$ to second order:

$$L(\vec{a} + \nabla \vec{a}) = L(\vec{a}) + g^T \cdot \Delta a + \tfrac{1}{2} \cdot \Delta a^T \cdot G \cdot \Delta a$$

where $\Delta a^T = [\Delta a_1, \Delta a_2, \ldots]$
$g^T = [\partial L / \partial a_1, \partial L / \partial a_2, \ldots]$
$G_{ik} = \partial^2 L / \partial a_i \, \partial a_k$

Extremum of L:

$$\partial L/\partial a_i = 0$$

leads to

$$g + G \cdot \Delta a = 0$$

$$\Delta a = -G^{-1} \cdot g = V \cdot g \qquad (1)$$

Another strategy consists in just going along the direction of the gradient:

$$\Delta \vec{a} = t \cdot \vec{g}$$

leading to

$$L(\vec{a} + t \cdot \vec{g}) = t \cdot \vec{g}^T \cdot \vec{g} + \tfrac{1}{2} t^2 \cdot \vec{g}^T \cdot G \cdot \vec{g}$$

optimal step length $t: \partial L/\partial t = 0$ leads to

$$t = -\frac{\vec{g}^T \cdot \vec{g}}{\vec{g}^T \cdot G \cdot \vec{g}}$$

$$\Delta \vec{a} = -\frac{\vec{g}^T \cdot \vec{g}}{\vec{g}^T \cdot G \cdot \vec{g}} \cdot \vec{g} \qquad (2)$$

The step actually made by MLFIT is a combination of the two possibilities (1) and (2):

$$\Delta \vec{a} = -K^{-1} \cdot \vec{g},$$

where

$$K = G + \epsilon \cdot (2^{\ell} - 1) \cdot \frac{g^T \cdot G \cdot g}{g^T \cdot g} \cdot I, \quad I = \text{unit matrix}.$$

According to the numerical values of the parameters ϵ and ℓ, the step is more like Eq. (1) or more like Eq. (2). The decision on ϵ and ℓ is made according to a certain strategy, which depends on the result obtained at each step. The process is terminated if the change of the function at the last step is sufficiently small and if a certain limit for the parameter ℓ has been reached.

REFERENCES

[1] CERN, T.C., Program Library SUMX 30.10.68.
[2] BUTENSCHÖN, H., DESY Report 66/29; internal DESY Report R 1 - 69/1.
[3] See, e.g. HUDSON, D.J., CERN 64-18; SWANSON, W.P., DESY Report 66/17 (in German).
[4] FRIEDMAN, J.H., ROSS, R.R., Phys. Rev. Letts 16 (1966) 485. For a more recent complicated example see ABRAMOVICH, M. et al., Nucl. Phys. B 23 (1970) 466.
[5] SHEPPEY, G.C., CERN 68-5.
[6] DAVIDON, W.C., Computer J. 10 (1968) 406.
[7] BLOBEL, V., DESY Report 71/18.

DATA PROCESSING FOR ELECTRONIC TECHNIQUES IN HIGH-FREQUENCY EXPERIMENTS*

S.J. LINDENBAUM
Brookhaven National Laboratory, Upton, N.Y.
and
Physics Department, City College of
City University of New York,
United States of America

Abstract

DATA PROCESSING FOR ELECTRONIC TECHNIQUES IN HIGH-FREQUENCY EXPERIMENTS.

A discussion on the use of on-line computer techniques with large arrays of directly digitizable detectors is presented.

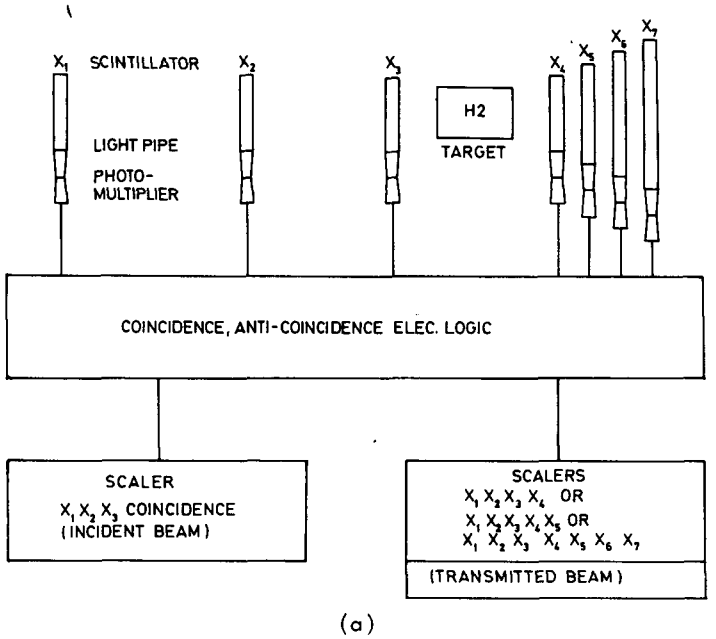
I would like to say that I think the title of the Seminar Course, "Computing as a Language of Physics", is both an appropriate and interesting one. Of course, what constitutes a language and how it is employed are subject to wide variations.

Language can be a varied subject ranging from primitive man's sign signals to a powerful modern language with rich vocabulary, written as well as vocal.

Furthermore, language communications can range from exchange of letters to a rapidly changing dialogue between individuals or public-meeting type of interactive debate. Thus, although there is no doubt that computing is by now a language of physics for all disciplines, I believe that in the field of Data Processing for Electronic Techniques in High-Energy Physics, computing has reached its most advanced and prolific level as "a language of physics". In particular, in the use of the ON-LINE COMPUTER TECHNIQUE coupled with a massive array of directly digitizable detectors; computing can be characterized as an Interactive Language of enormous information exchange capability between the physicist and his rather sophisticated detector devices. In fact, as a result of this intensive dialogue capability, the physicists are now able to make enormous escalations in the complexity of their apparatus and the scope and accuracy of their measurements. Thus the resultant intensification of the investigation of nature by the electronic detector and computer techniques has been truly impressive and fruitful for the research physicist.

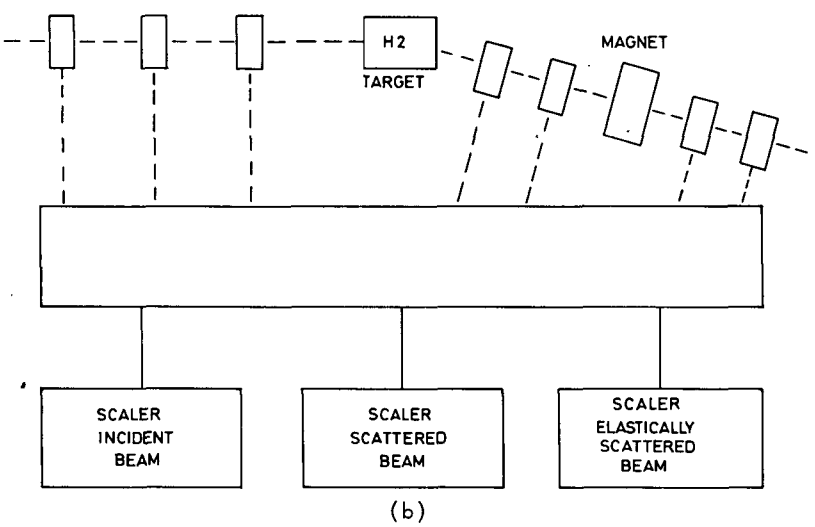
The simplest type of modern (high-energy) detector is the scintillation counter telescope. These devices tell the physicist each time a particle (which he generally would have selected by beam transport to have near-monochromatic momentum and would have identified by using velocity-measuring Cherenkov counters) passed through each counter in time coincidence (after allowing for travel time). Those numbers were displayed on scalers, which one must remember were originally developed in what was,

* Work performed under the auspices of the US Atomic Energy Commission.



(a)

FIG. 1(a). A total cross-section measurement (conceptual schematic only). The incident beam and the transmitted percentage of incident beam are determined (as a function of solid angle subtended by the final counter) when a known amount of liquid hydrogen absorber is placed (target full) or removed (target empty) from the beam. Thus the total nuclear interaction cross-section of a proton for the incident particle can be obtained via extrapolation to zero solid angle.



(b)

FIG. 1(b). The differential elastic-scattering cross-section $d\sigma/d\Omega$ can be determined at a particular (mean) scattering angle, by choosing an appropriate combination of magnetic field and positioning of the last two counters so that only elastically scattered particles can pass through all counters. An inelastic momentum spectrum of particles produced at a particular polar angle can also be obtained by varying the magnet currents. By re-positioning the counters downstream of the hydrogen target, different polar angles can be measured.

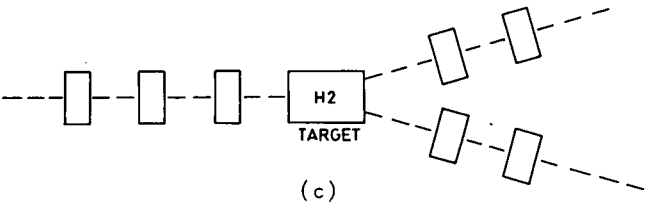


FIG. 1(c). The magnetic-analysis method for elastic scattering shown in Fig. 1(b) can, in principle, be used at all angles or momentum transfer squared. An alternate method which depends on selecting the required kinematics between the two particles after the scattering is shown in this figure. It can be used fruitfully when enough momentum transfer has taken place so that the target proton can escape the hydrogen target without excessive multiple scattering. The combination of magnetic analysis on one or both particles as well as kinematic requirements are also used when rates are low and one wishes to discriminate against the background.

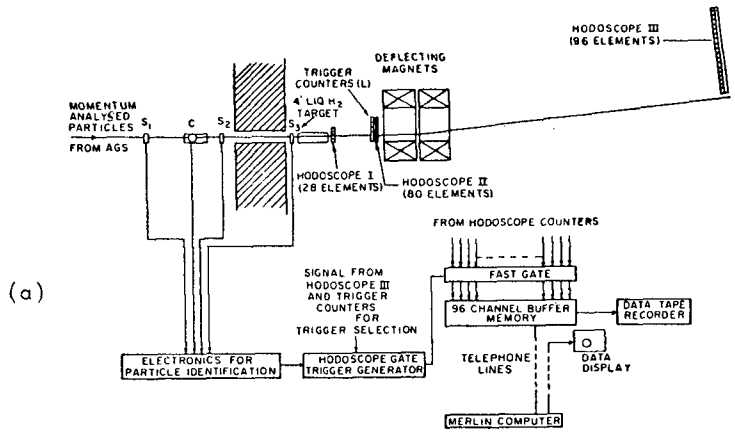
then, high-energy physics (i.e. nuclear physics). These basic scales of two elements made possible the development of the modern digital computer. In the late forties and through the next decade of the fifties, many important experiments were done this way. (See Figs 1(a)-(c) for explanations of these techniques.)

In addition to H_2, physicists used other absorber material targets including a technique called range-curve analysis to determine the nature of the incident beam. Their dedicated off-line computer was usually a slide rule or a mechanical desk calculator. Sometimes an off-line digital computer was used in Monte-Carlo-type simulations or other calculations of corrections. In particular, detailed phase-shift analyses of differential elastic scattering required the use of an off-line digital computer.

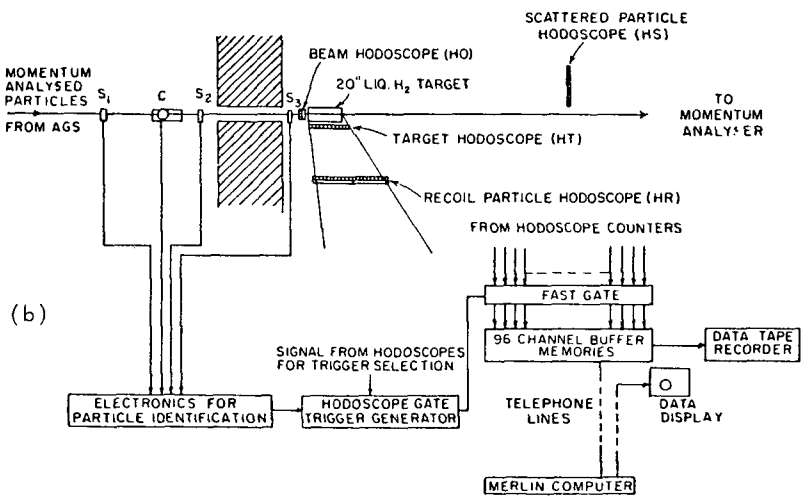
In 1962, after several years of development, a new approach was taken [1] which completely changed the style of doing digitizable electronic detector physics. The motivation for this was simply how does one obtain a large fraction of the solid angle in elastic scattering combined with high resolution in angle and momentum and the high counting rates typical for electronic detectors.

In principle, one could replace each counter in the downstream part of the telescope by a large area of more counters. However, because of technical difficulties and the absolutely impossible task of employing a sufficient quantity of conventional electronic logic and scalers, and the impossibility of using the usual techniques of analysis in such a complex system, a new approach was needed; Figures 2(a)-(d) show the method developed. The counters in Fig. 1(b) and 1(c) were replaced by a set of counter hodoscopes. A hodoscope is an array of adjoining long, thin, rectangular slab detector elements which locate a Cartesian co-ordinate of a particle traversing its plane surface. The actual position of a single particle in space is determined by the intersection of perpendicular elements of two crossed hodoscopes (i.e. x and y hodoscope planes). If there is more than one particle incident on a hodoscope array, a third crossed plane of elements parallel to some intermediate angle (i.e. 45° to the x and y directions) is added to uniquely identify the co-ordinates of each particle.

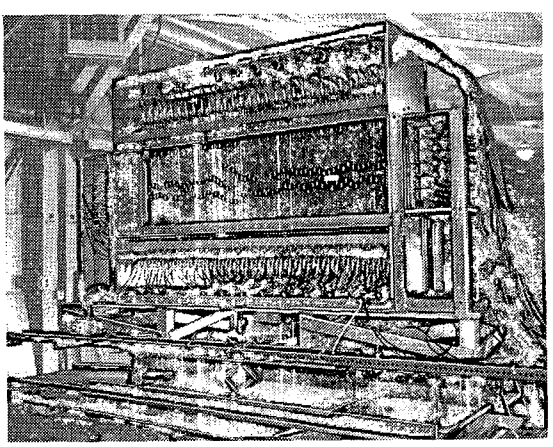
Figures 2(a) and 2(b) show the two counter-hodoscope arrangements used and Fig. 2(c) shows a typical hodoscope.


HODOSCOPE III
(96 ELEMENTS)
DEFLECTING
MAGNETS
TRIGGER
COUNTERS (L)
4" LIQ H2
TARGET
MOMENTUM
ANALYSED
PARTICLES
FROM AGS
S1
C
S2
S3
HODOSCOPE I
(28 ELEMENTS)
HODOSCOPE II
(80 ELEMENTS)
FROM HODOSCOPE COUNTERS
FAST GATE
SIGNAL FROM
HODOSCOPE III
AND TRIGGER
COUNTERS
FOR
TRIGGER SELECTION
96 CHANNEL BUFFER
MEMORY
DATA TAPE
RECORDER
ELECTRONICS FOR
PARTICLE IDENTIFICATION
HODOSCOPE GATE
TRIGGER GENERATOR
TELEPHONE
LINES
DATA
DISPLAY
MERLIN COMPUTER

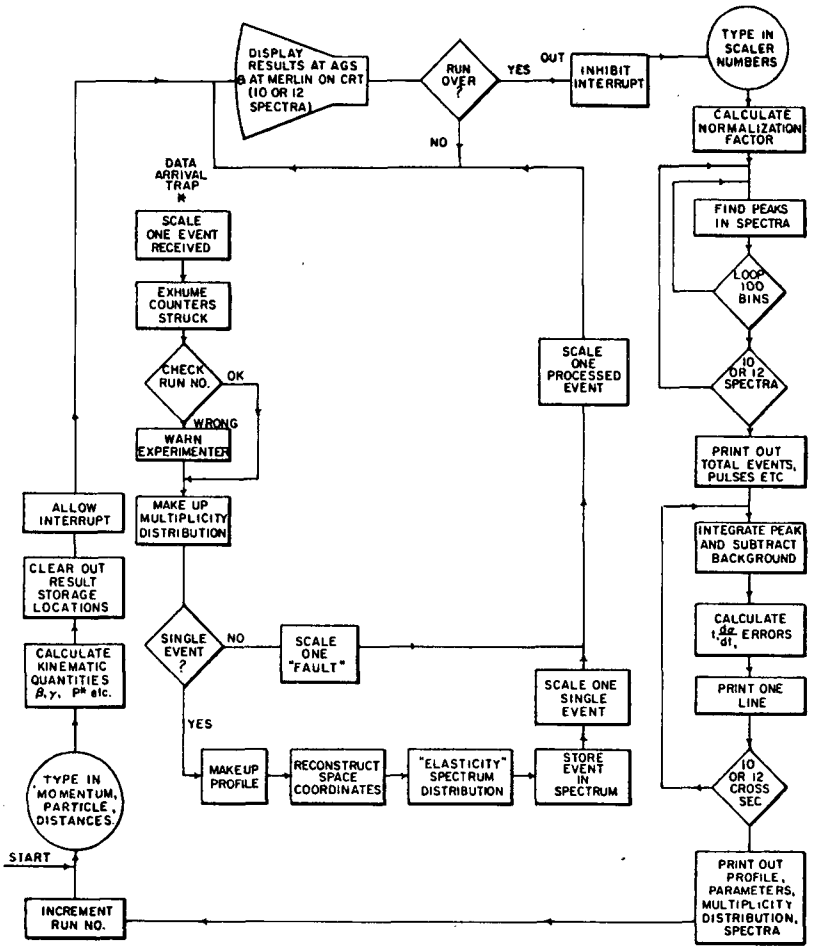
(a)


SCATTERED PARTICLE
HODOSCOPE (HS)
BEAM HODOSCOPE (HO)
MOMENTUM
ANALYSED
PARTICLES
FROM AGS
S1
C
S2
S3
20" LIQ. H2 TARGET
TO
MOMENTUM
ANALYSER
TARGET HODOSCOPE (HT)
RECOIL PARTICLE HODOSCOPE (HR)
FROM HODOSCOPE COUNTERS
FAST GATE
96 CHANNEL BUFFER
MEMORIES
DATA TAPE
RECORDER
SIGNAL FROM HODOSCOPES
FOR TRIGGER SELECTION
ELECTRONICS FOR
PARTICLE IDENTIFICATION
HODOSCOPE GATE
TRIGGER GENERATOR
TELEPHONE
LINES
DATA
DISPLAY
MERLIN COMPUTER

(b)



(c)



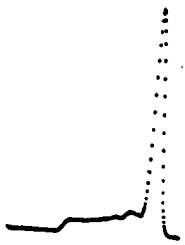
(d)

FIG.2(a). The magnetic spectrometer, counter-hodoscope on-line computer system for detection of (7 - 20) BeV/c small-angle elastic scattering (arrangement I) at the Brookhaven 33-BeV alternating-gradient synchrotron. Hodoscopes HI and HII determine the horizontal projection of the scattered direction of an incident particle after scattering in the hydrogen target. The vertical elements of hodoscope HIII determine its reflection in the magnet and hence momentum, while the horizontal elements determine the vertical projection of the direction of the scattered particle.

FIG.2(b). The counter-hodoscope on-line computer system for selecting larger-angle elastic scattering by determining the space locations of the incident particle (telescope plus hodoscope HO), the forward scattered particle (hodoscope HS), and the recoil particle (hodoscopes HT and HR). Then coplanarity and proper kinematic angles were required to select elastic scattering.

FIG.2(c). A photograph of a counter hodoscope of the type labelled Hodoscope 3 in Fig.2(a).

FIG.2(d). The program flow chart of the first on-line computer counter experiments described in Fig.2(a) and Fig.2(b).



(e)

Fig. 2(e) Typical on-line scope display of momentum spectra showing elastic-scattering peak.

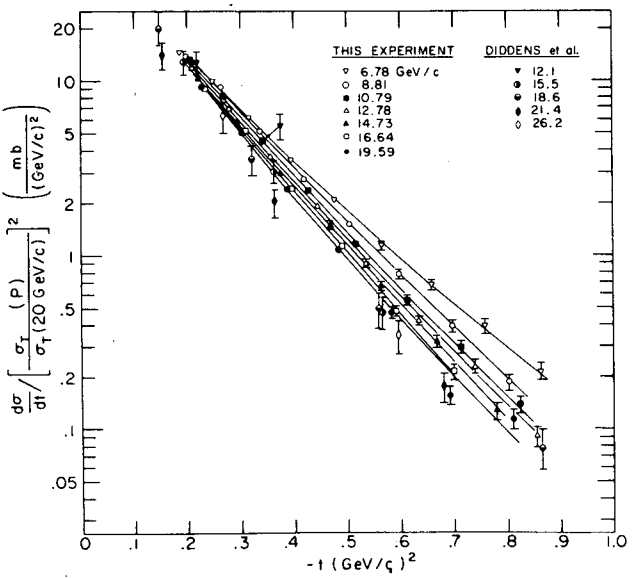
The only practical way of handling the debugging and data from such a complex array of elements, was to use digital-computer data handling and analysis techniques. For this system, the on-line computer technique [2] was first developed. This was the first truly on-line computer experiment in physics since the event trigger selection; the recording of the basic pulse data from the detectors, the transmission of data to the computer, and their evaluation in the desired processed form to give answers to crucial theoretical questions were all controlled by electronic devices without human bias or interruption except for the pre-stored programs, control commands, and entering of summaries of monitors, parameters of location of the counters, the beam momentum, etc. The program flow chart is shown in Fig. 2(d). Figure 2(e) shows a typical on-line scope display feedback to the experimentalists a mile away.

As a result of these on-line computer experiments, it was shown that the high-energy diffraction-type elastic scattering did not shrink universally as predicted by the then popular dominance of the vacuum-pole Regge theory. Rather, as shown in Figs 3(a) and 3(b), p + p did shrink, while $\pi^{\pm}$+p did not.

The experiments just described used the Merlin computer, a general-purpose computer which was patterned after the Maniac II and built at Brookhaven with 8,192 words of 48 bit core and with an average computing speed of about 1/7th of an IBM7094. It was located about a mile from the experiment. A local digital data handler accepted and separately stored the digital data from the experimental detectors each beam burst from the 33-BeV Alternating Gradient Syncrotron (AGS) and then between pulses placed the data on tape and simultaneously transmitted it to the Merlin computer via a two-way telephone line in a link which fed back desired scope display data.

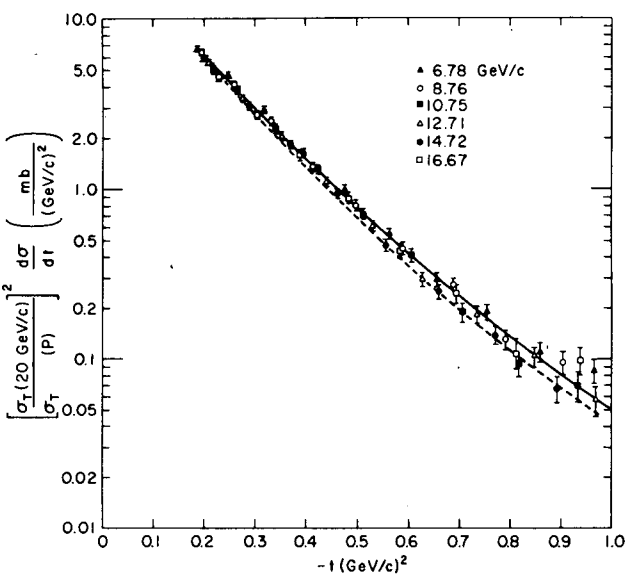
Later applications of the on-line computer technique involved a CERN missing mass spectrometer [3] (Fig. 4) which employed, in addition to counter hodoscopes, sonic chambers as part of the digitizable detectors.

As a result of the rapidly increasing interest from the Brookhaven user community in these new on-line computer techniques, the author and some of his colleagues and the Brookhaven Physics Department organized, in 1964, a general user facility called the On-Line Data Facility which



(a)

FIG. 3(a). The quantity $(d\sigma/dt)/\{[\sigma_t(P)]/[\sigma_t(20\ GeV/c)]\}^2$ in mb/(GeV/c)^2 for 7- to 20-GeV/c p-p elastic scattering, which is proportional to $(d\sigma/dt)/(d\sigma/dt)_{opt}$ plotted versus t.



(b)

FIG. 3(b). The quantity $\{[\sigma_t(20\ GeV/c)]/[\sigma_t(P)]\}^2(d\sigma/dt)$ in mb/(GeV/c)^2 for 7- to 17-GeV/c π^+ - p, which is proportional to $d\sigma/dt/(d\sigma/dt)_{opt}$ plotted versus t. The solid line is a least-squares fit to all the π^+ - p data. The dotted line is a least-squares fit to all the previously reported π^- - p data.

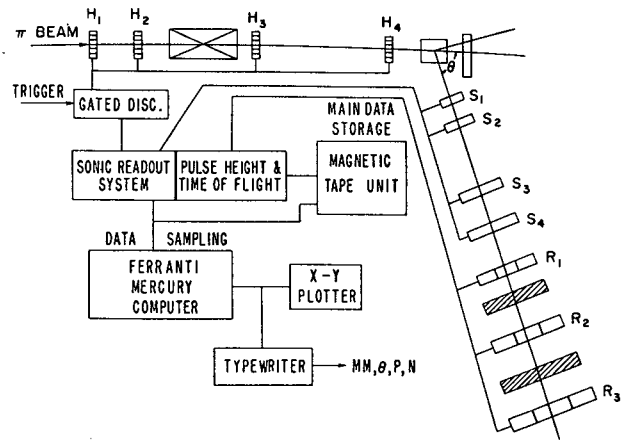


FIG. 4. Simplified schematic of a sonic spark chamber set up (20 d) for studying missing-mass inelastic reaction $\pi^- + p \rightarrow p + x^-$. The direction and velocity of the recoil protons are measured, and then the mass of the missing particle x^- is deduced by using energy and momentum conservation.

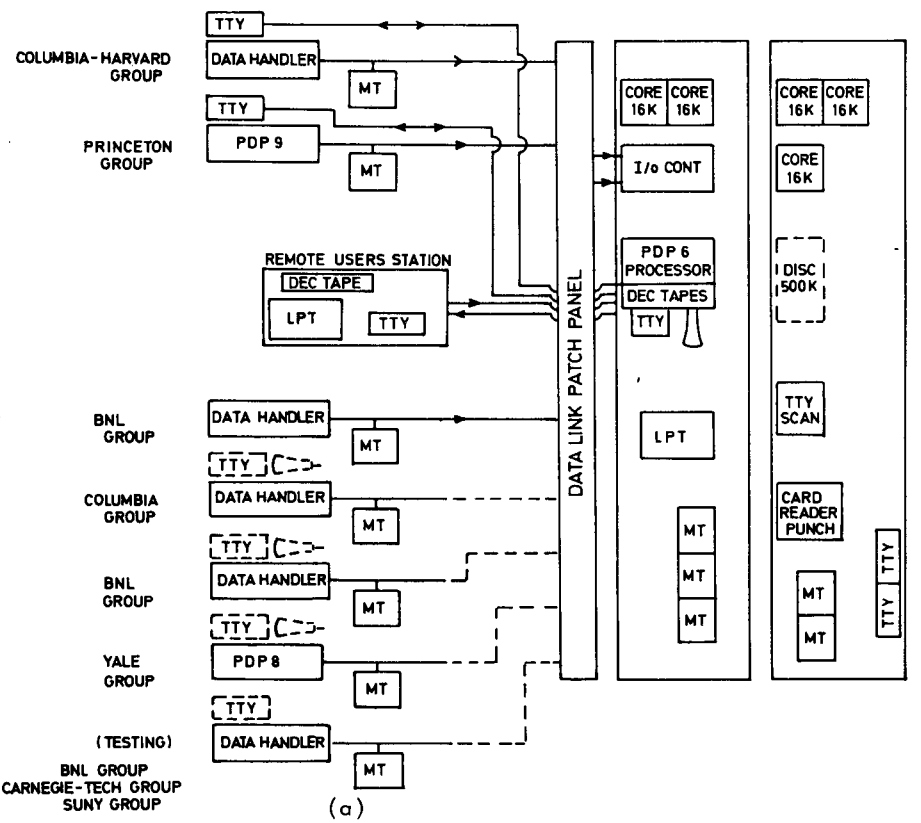
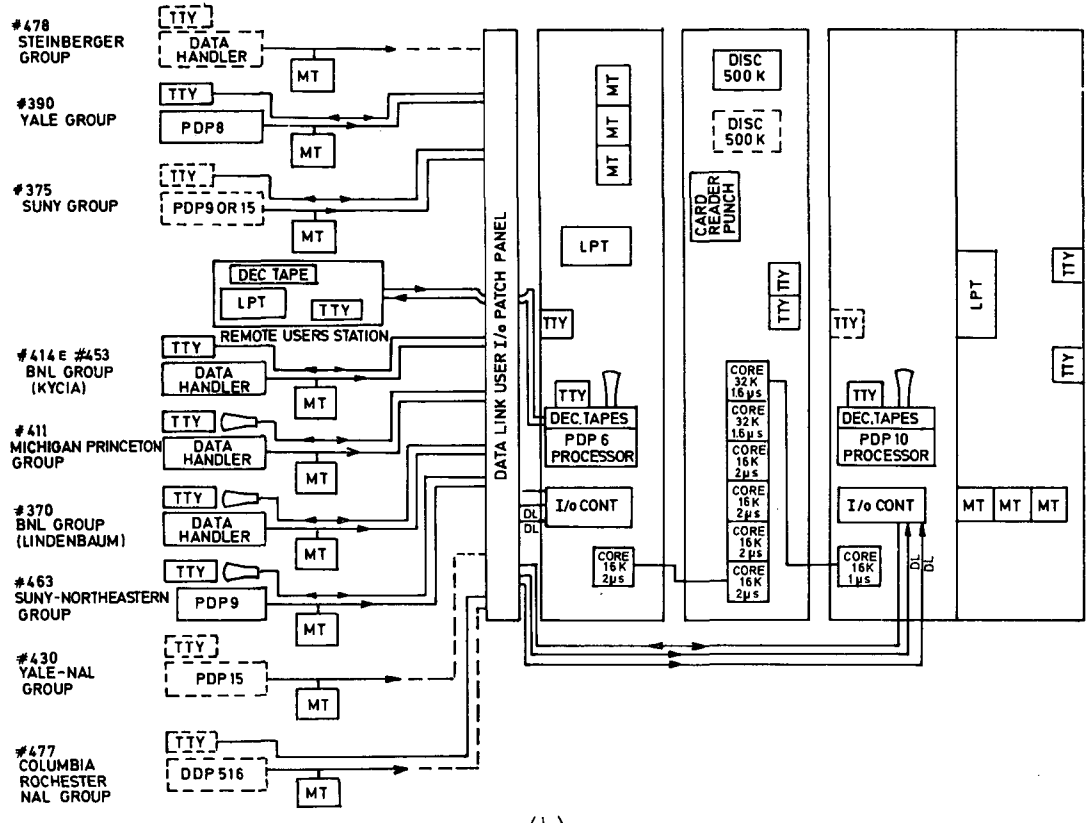


FIG. 5(a). The configuration of the PDP-6 complex of the Brookhaven On-Line Data Facility.



(b)

FIG. 5(b). The configuration of the combined PDP-6/10 complex of the Brookhaven On-Line Data Facility.

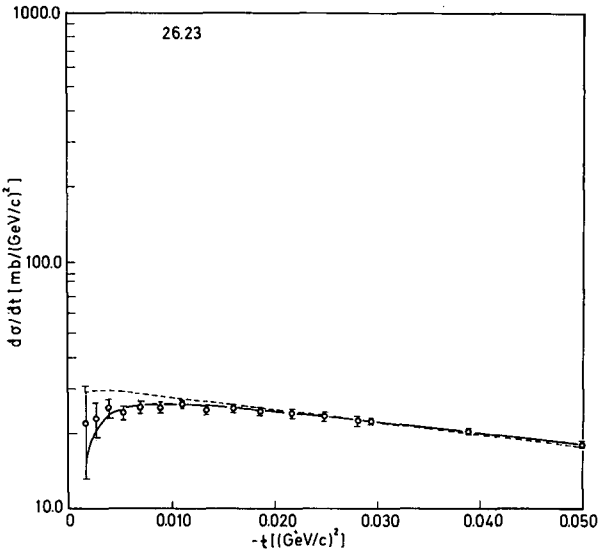


FIG. 6. The elastic differential π^- + p scattering cross-sections after subtraction of single Coulomb scattering. The dashed line represents the best fit with α = ratio $(F_N)_{Re}/(F_N)_{Im}$ = 0.

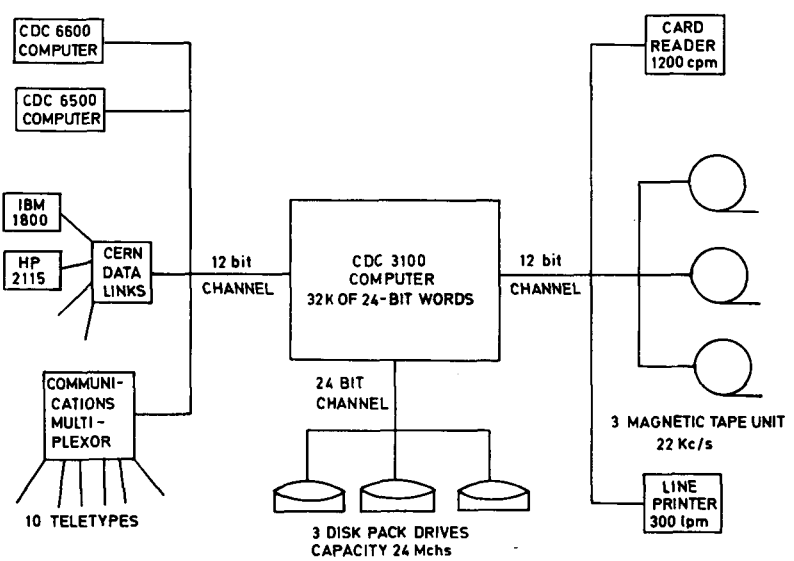


FIG. 7. Block diagram of hardware configuration of CERN focus system.

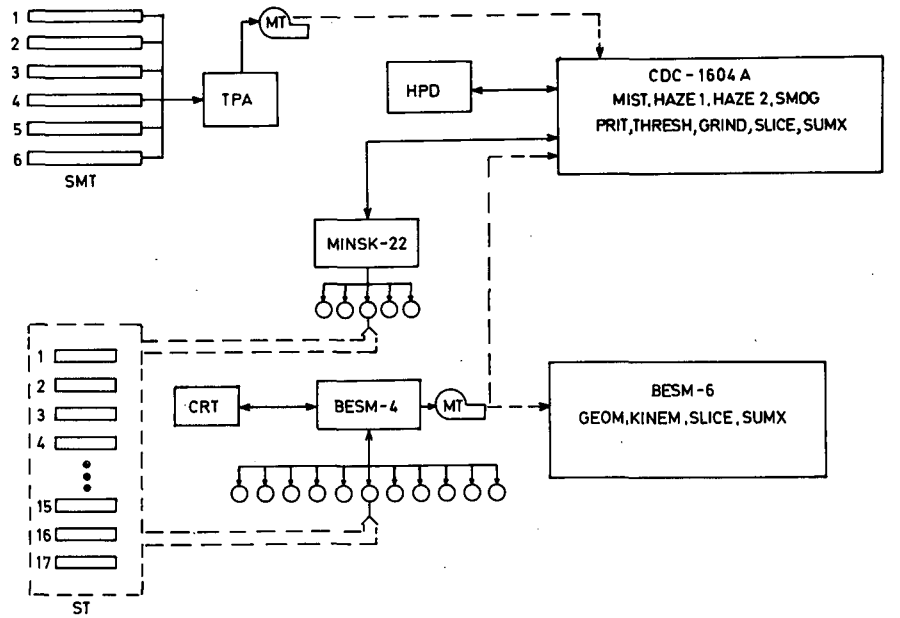


FIG. 8. The system of data processing for track chambers at the JINR and Serpukhov in the USSR.

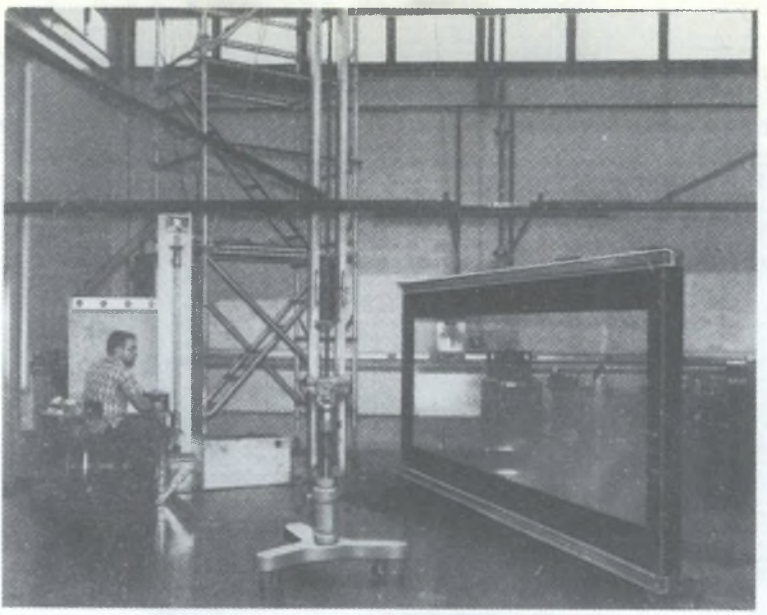
contained a PDP-6 interactive time-sharing computer which was put into service early in 1965 [2]. The computer was located in one, and then eventually two, 40' long by 10' wide road-type truck trailers at the AGS.

Figure 5a shows a typical arrangement of this facility several years later when the bulk of the electronic detector experiments at Brookhaven were utilizing it. The time sharing of two real experiments first occurred in 1965 - 66 and the proliferation of multi-user use by 1969 is indicated in Fig. 5a. Two to three on-line users were accommodated simultaneously and the remainder of users who were debugging and preparing for the next phase went in rotation in an off-line queue via the disc.

The experiments included a study of small-angle scattering [5] of $\pi^+ + p$ and $p + p$ in the Coulomb-interference region, to determine the real part of the scattering amplitude in order to test the pion-nucleon forward dispersion relations.

Other experiments included: a check of CP-invariance in the $K^\pm$ decay via the mode which was consistent with no violation [6]; a study of the effective mass of charged muon pairs which appear to support electronic scaling; a study of nucleon isobars produced in p - p collisions [7]; a precision study of $\pi^\pm$ - p and p - p, $\bar{p}$ - p cross-sections which, together with the small-angle scattering experiments, demonstrated that the Pomeranchek theorem would not come true to somewhere in the range beyond 25 000 GeV to perhaps one million GeV [8, 5]; a study of polarization in $K^+ + p$ by a Yale group [9] (Fig. 6).

Any one of the typical experiments was supported by the following equipment: a local digital data handler or small satellite computer



(a)

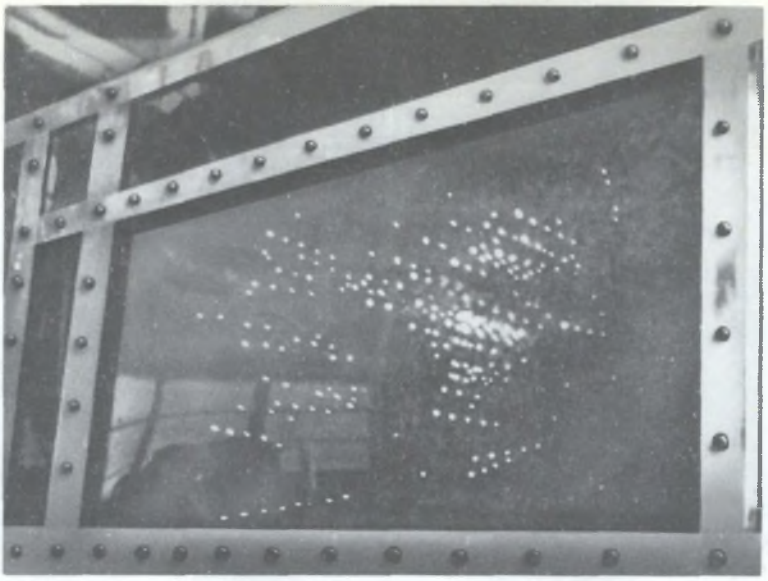
FIG. 9(a). This figure shows the wire spark-chamber hodoscope winder built in the high-bay area of the Physics building. It can wind up to 23' by 23' sensitive area with wire spacing adjusted to any desired distance from 0.020" and up. The wire chamber frame rotates about its vertical axis while it is positioned on ground steel beds. The wire feed point is driven by a lead screw along the vertical slide. The tension on the wire is controlled utilizing a long vacuum column to better than 5%.

(often supplied by the experimental group), an on-line link to the PDP-6 complex when the experiment is running with local teletypes for communication with the PDP-6. Generally, only a sample (typically, at least, 10%) of the data is processed on-line and each user is provided with individual scope displays. Printers and high-performance tapes are located in the computer trailers and in some instances remote I/O stations near users. A typical user on-line program varies from 20K - 45K of 36 bit words and requires from approximately 25% to 50% of the processor time available.

The ever increasing demand for on-line computer services at Brookhaven National Laboratory led to the addition of a PDP-10 computer in the summer of 1969 and Fig. 5(b) (#3-1020-70) shows the total PDP6/10 complex available since the latter part of 1969.

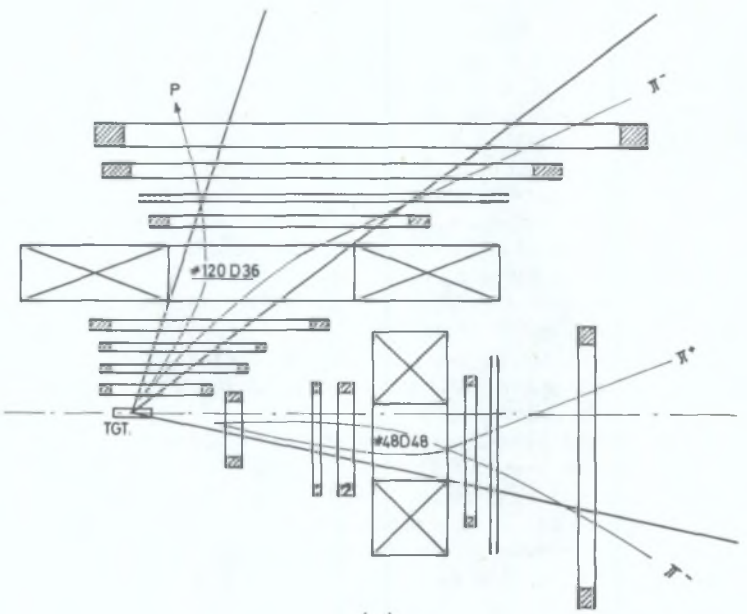
We now have the capability of serving 4 - 6 on-line users simultaneously, with many off-line users also making use of the disc rotation system to debug their programs and process necessary data before their scheduled experimental runs. The time sharing (or multi-programming) capability is a very valuable technique for a scientific user facility.

At CERN, the application of the On-Line Computer technique has been primarily via smaller local computers at each experiment. Earlier attempts to hook on-line to the CDC6600 (6500 complex) led to excessive demands



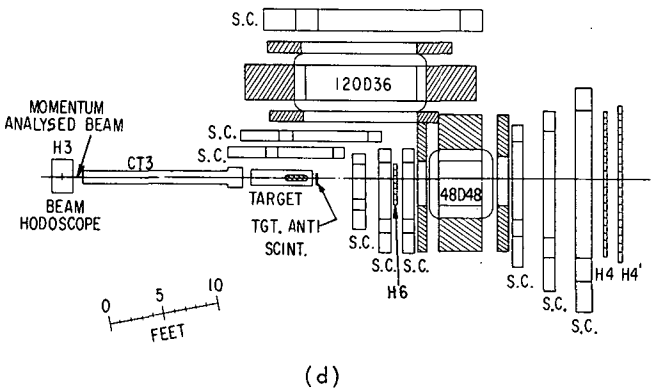
(b)

FIG. 9(b). A photograph of the sparks in the chamber set, downstream of the magnet.



(c)

FIG. 9(c). $\pi^- + p \rightarrow K^0 + \Lambda \rightarrow (\pi^+ + \pi^-) + (p + \pi^-)$.



(d)

FIG. 9(d). Arrangement used in A_2 experiment.

on the central facility and "bycycle on-line", i. e. shorter turn around off-line for tapes delivered to the central facility has been utilized. Recently, more direct on-line use of the CDC6600 has been developed via the Focus remote access system shown in Fig. 7, in which a CDC-3100 computer with 32K of 24 bit words serves as an interface between the on-line user and the CDC6600/6500 complex.

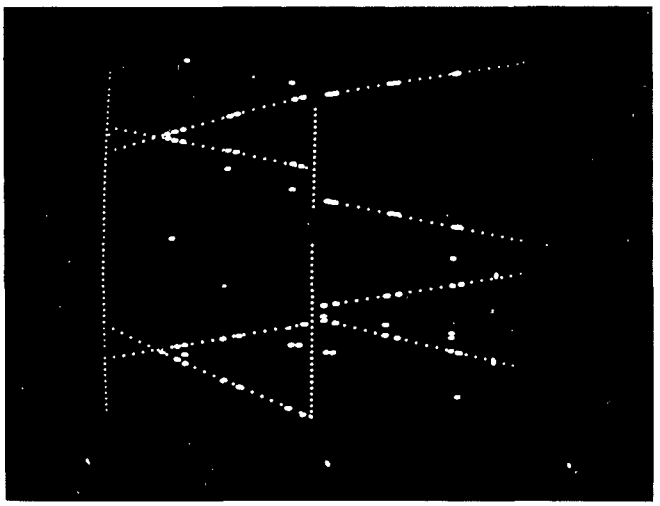
For a number of years attempts to use the Brookhaven CDC6600 complex directly from a local satellite computer, have been in progress. It is the author's opinion that both better and more economical service can be obtained if the CDC6600 complex is reserved for off-line batch with as rapid turn-around time as possible. To this end, a low-price link is being provided from the PDP6/10 complex to the CDC6600 complex. The direct almost instant turn-around, on-line support in this system would still be provided by the PDP6/10 complex but batch processing would be entered routinely and automatically to the CDC6600 complex and returned automatically. This system economizes enormously on the memory and I/O required by the on-line satellites and is far more satisfactory in turn-around time and general flexibility and scope of service provided.

Figure 8 shows a block diagram of the USSR computing facility [10] for track (bubble and spark chambers) picture processing at the JINR and the Serpukhov accelerators, and probably can be taken as representative of the computer facilities available for supporting electronic detector experiments at the various laboratories.

The BESM-4 is a 3-address computer with 8K memory of 45 bit words and a 600k word drum. It operates at about 20 000 instructions per second. The Minsk-22 is a 2-address machine with a core memory of 8K of 37 bit words and a memory cycle time of 24 μs.

The BESM-6 computer works at a speed of one million one-address instructions per second, has 32K core memory of 48 bit words and a 512K word drum.

The TPA is a Hungarian computer with 4K basic memory of 12 bit word length and 1.0 μs cycle time. It can be compared to a PDP 8.



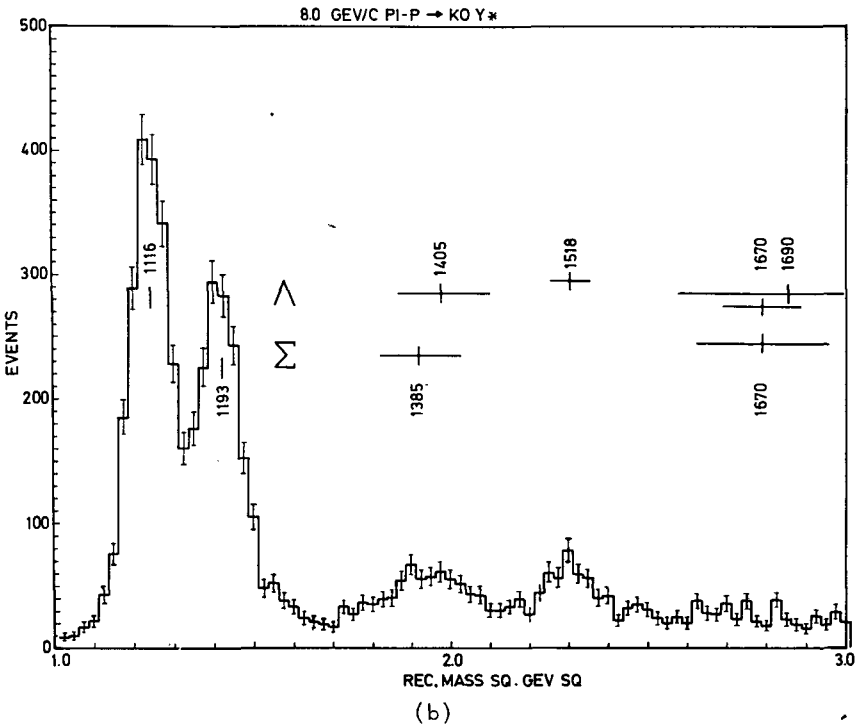
(a)

FIG.10(a). On-line display of four views of a K^0 decay detected in the forward spectrometer. To the left there are the views before entering the magnet, plan view at the bottom, elevation at top. To the right there are similar views after the magnet. The bright, short horizontal lines represent the spark positions in the chamber gaps; the dotted lines through them are the tracks fitted by the computer. On each track in the bottom righthand view, the short vertical line representing the size of the trigger counter in hodoscope H4 struck by the pion can be seen. The vertical lines at the left of each view are rulers for calibration, representing 10 inches (total length) in all views except the lower righthand view, where the ruler is 75 inches long.

In the on-line computer systems, the counter hodoscopes were soon supplemented with the (wire) spark chamber hodoscopes which are more economical and better suited for large solid-angle multi-particle detectors with minimal multiple scattering. However, the time resolution of a spark chamber hodoscope is two orders of magnitude worse, and it must be supplemented with a triggering system (to pulse the spark chambers), which is usually composed of scintillation counter hodoscopes or the newly developed continuously sensitive proportional chamber hodoscopes.

As an example of construction and operation of a very large digitized spark chamber hodoscope system with the on-line computer technique, we can use the BNL Double-Vee Magnetic Spectrometer developed by the author and his co-workers [11].

Figure 9(a) shows how large fiberflass frames are spun on an axis and about twenty 5 mil Al-wires are wound per inch horizontally upon it. The wire is fastened to one side and cut from the other side. Two such planar frames with the parallel wires facing each other and separated by a spacer (with 2 vacuum seals and a partial vacuum between them) make an x, y or w (45°) spark-chamber hodoscope. A magnetic wire runs across (but insulated from) the conducting wire. When a suitable trigger logic pulse occurs, a voltage pulse of about 7 kV is applied between the wire planes of each hodoscope, and a spark appears wherever a charged particle has



(b)

FIG. 10(b). The missing mass-square distribution for the process $\pi^- + p \rightarrow K^0 + MM$. Incident π^- momentum is 8.0 GeV/c. The masses and mass widths shown for the Y*'s are from the table of Particle Properties.

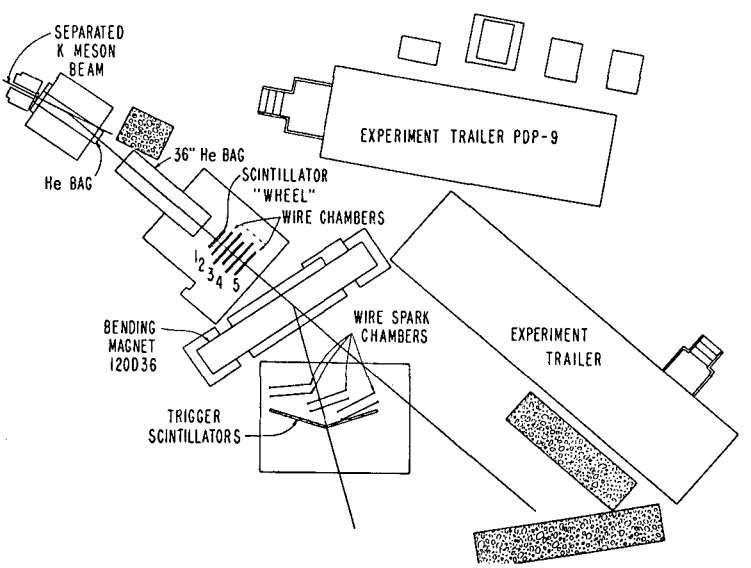


FIG. 11. The set-up of Piroue et al. to measure the dependence of the decay of $K^{\pm} \rightarrow \pi^+ \pi^- \pi^{\pm}$ on the momentum of the "odd" pion.

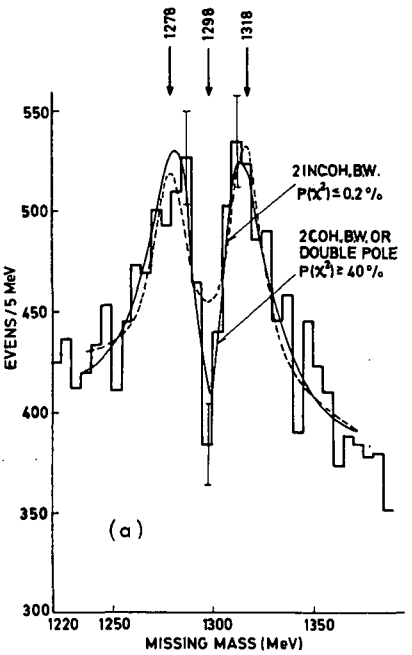


FIG. 12(a). A_2 splitting in the missing-mass spectrum – MM-spectrometer and boson spectrometer, combined data.

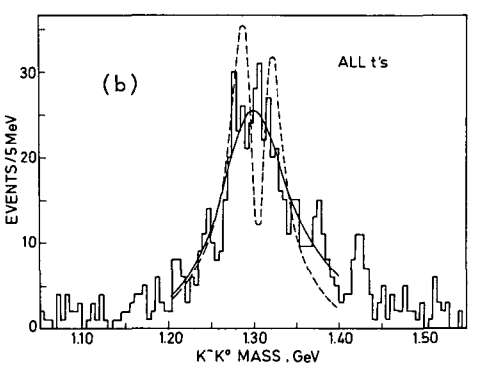


FIG. 12(b). The $K^-K^0_S$ effective (i.e. invariant) mass spectrum observed in the reaction $\pi^- + p \rightarrow K^0 + K^- + p$ (with $K^0 \rightarrow K^0_S$) for 20.3 GeV/c incident π^- on hydrogen. The solid line is a Breit-Wigner-type fit corresponding to $\ell = 2$. The dashed line is a "dipole fit" which obviously is unacceptable.

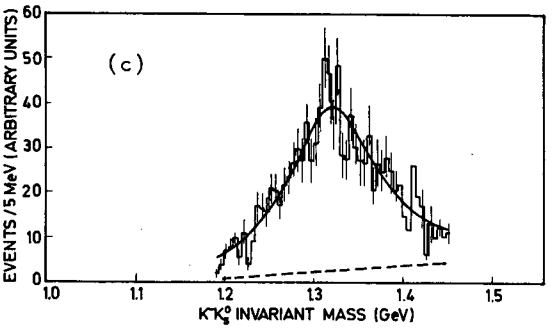


FIG. 12(c). The $K^-K^0_s$ invariant mass (GeV) observed in the reaction $\pi^- + p \rightarrow K^- + K^0 + p$ for 17.2 GeV/c π^- incident on hydrogen. The solid line is a Breit-Wigner-type fit corresponding to $\ell = 2$. The dashed line is a linear fit to the background.

traversed the plane. The appearance of a time-exposed picture of the sparks in a spark-chamber module composed of 2x, 2y and one w-chamber is shown in Fig. 9(b).

Each spark causes a magnetostrictive pulse to occur on the magnetistatic read-out line and the time of transit of each pulse when detected by a transducer at the end allows one to determine where the spark was. This operation is done automatically by electronic devices and the spark read-out is done in both directions by the independent read-out wires and the results averaged. Standard Neelium gas is used at approximately atmospheric pressure with quenching by alcohol vapour.

A schematic illustrating how the Double-Vee Spectrometer could be used to detect an event of the type

$$\pi^- + p \rightarrow K^0 + \Lambda$$
$$\downarrow \quad\quad \downarrow$$
$$\pi^+ + \pi^- \quad p + \pi^-$$

is shown in Fig. 9(c).

Figure 9(d) illustrates the arrangement used in the A_2 experiment to be described later.

Figure 10(a) shows how the on-line computer automatically reconstructs two views of a K^0 going forward before and after the forward magnet, thus allowing complete reconstruction in space (using the w-chambers to uniquely match the tracks in the two views) and hence measuring everything including the momenta.

Downstream Cherenkov counter hodoscopes will be available soon and can be used to directly identify the π^+ and π^-.

Figure 10(b) shows a typical missing-mass spectrum accompanying the K^0 exhibiting the Λ, Σ and various Y^* peaks.

The handling of data, the reconstruction of the event and even the complete analysis are all done automatically by the computer. Typically, $\approx 10\%$ of the data is analysed on line by the PDP10 which takes about one second CPU time to completely process one event. The bulk of the data is reduced off-line by the CDC6600 which takes about 1/4 second of CPU-time per event.

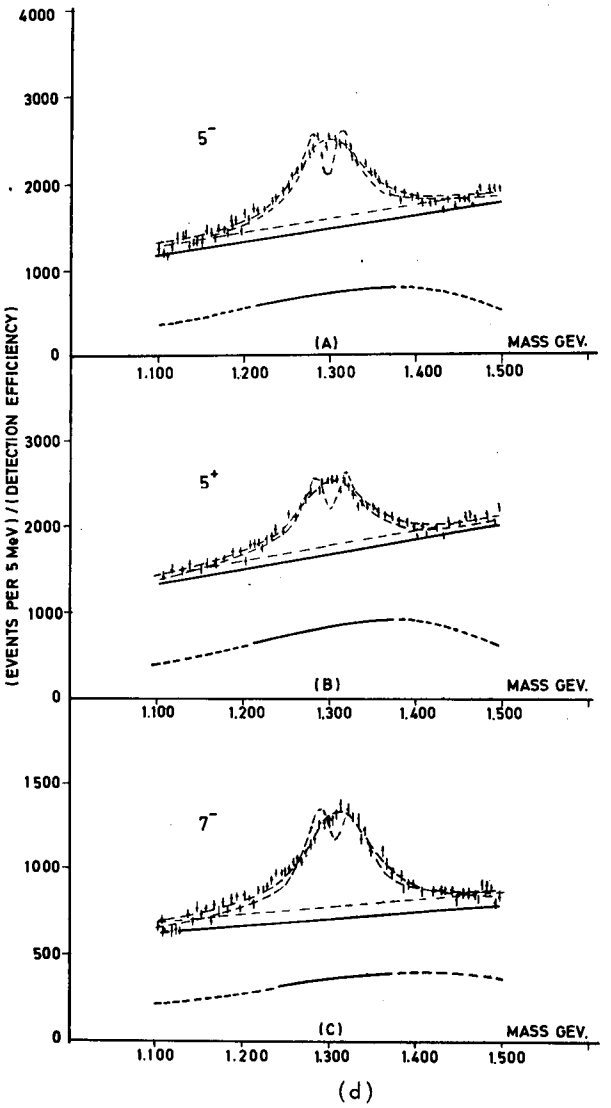


FIG. 12(d). A_2 meson not split.

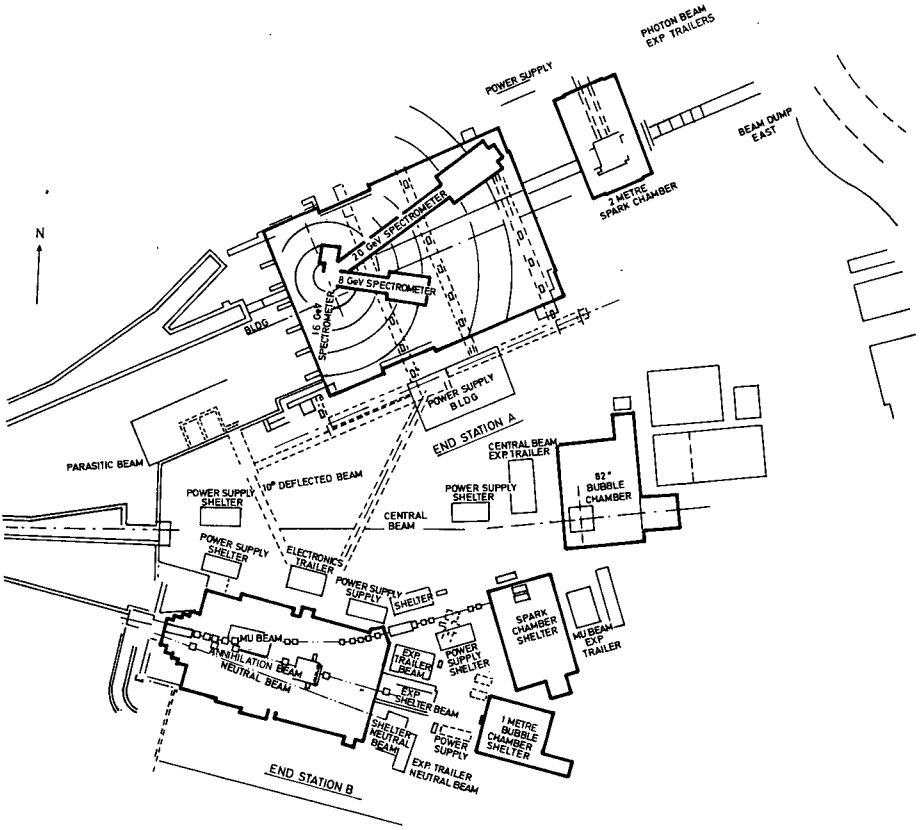


FIG. 13. Plan view of the experimental area at SLAC, showing the three magnetic spectrometers in end station A. The spectrometers pivot on rails about a common target point. A beam of electrons or photons strikes the target, and the angle and momentum of scattered or produced particles ($e^{\pm}$, $\mu^{\pm}$, $\pi^{\pm}$, $K^{\pm}$, p, etc.) are detected by the spectrometers.

The wire spark-chamber apparatus used for looking for asymmetries in the odd-charge pion in $K^{\pm}$ decay via the τ-mode (i.e. $\pi^{\pm}+\pi^{+}+\pi^{-}$) is shown in Fig. 11. In this case a PDP-9, a local satellite, was used for diagnostic and monitoring purposes but the raw data were still sent to the PDP6/10 for the critical on-line and also some off-line analysis. A study of the invariant μ-pair mass produced in proton-uranium collisions was also performed by a Columbia group (Christenson et al.) with a data handler on line to the PDP6/10. The result was consistent with the scaling hypothesis first proposed at SLAC.

An experimental result which caused a great deal of concern was the apparent split in the A_2 meson discovered by a CERN group using the missing-mass technique [3] (See Fig. 12(a)). This led to many speculations including the exotic double-pole hypothesis.

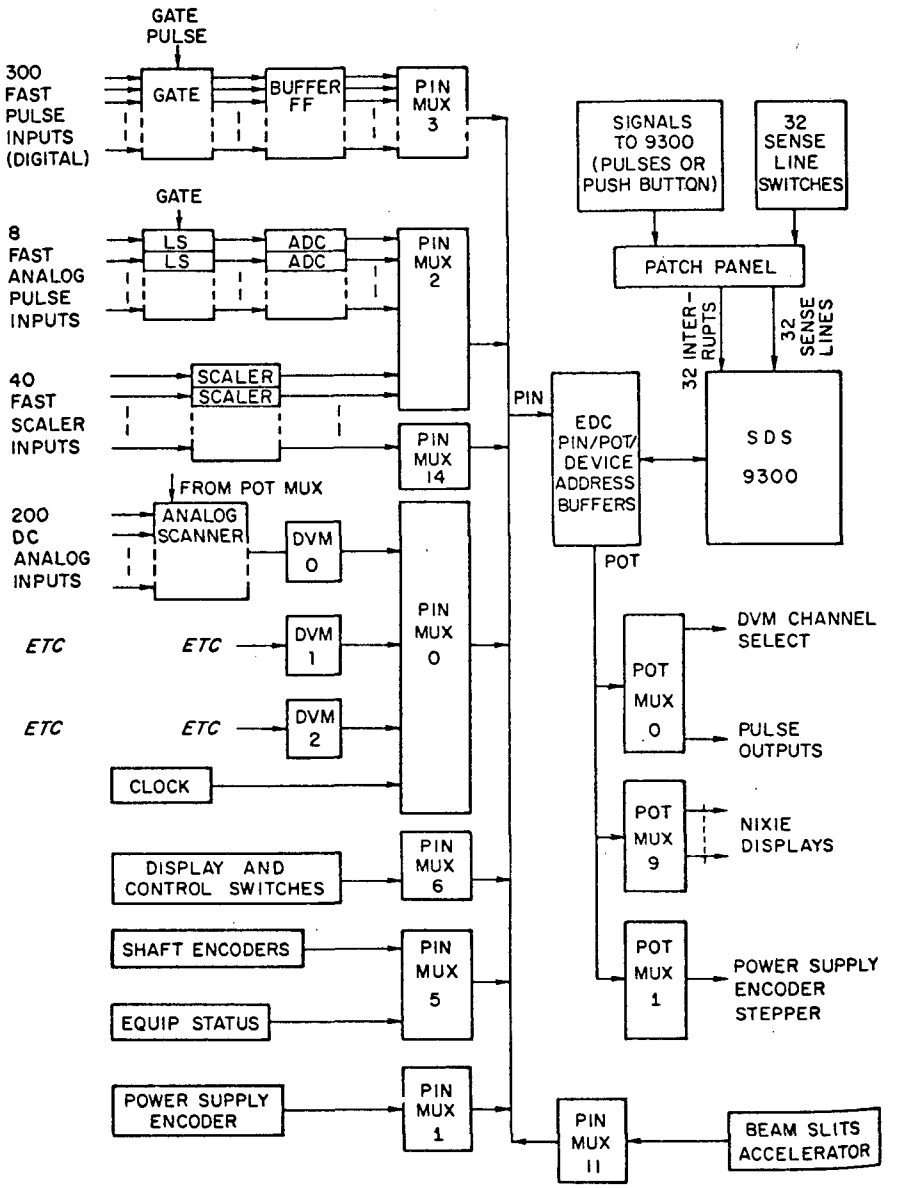


FIG. 14. Block diagram showing the interface to the SDS9300.

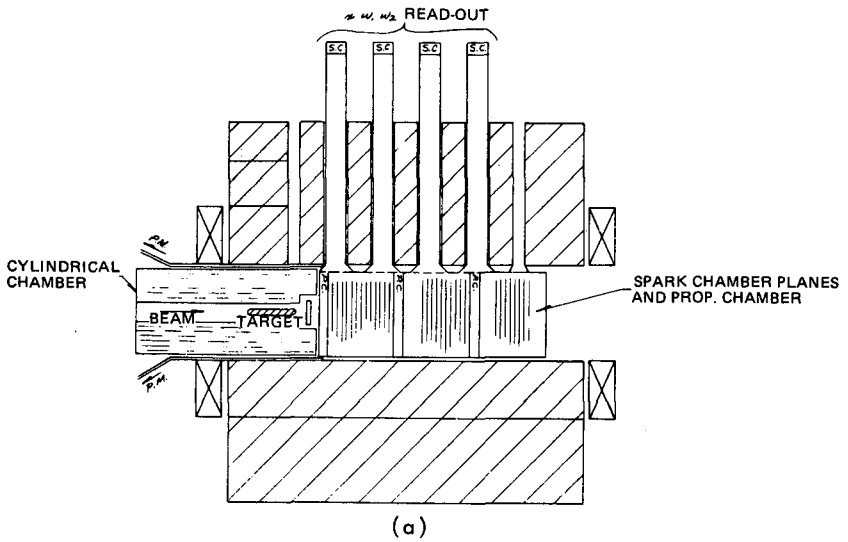


FIG. 15(a). The proposed MK I(b) arrangement with x, w_1 and w_2 wires being read out through the striated top.

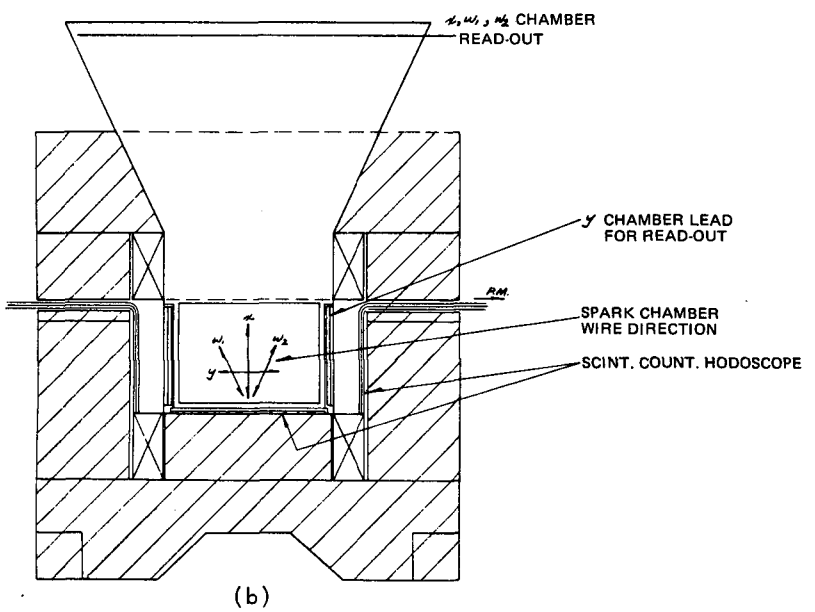


FIG. 15(b). An end view of the MK I(b) readout arrangement. w_1 and w_2 (45° inclined) wire planes alternate in the arrangement.

The BNL M. P. S. is a variant of the above basic design.

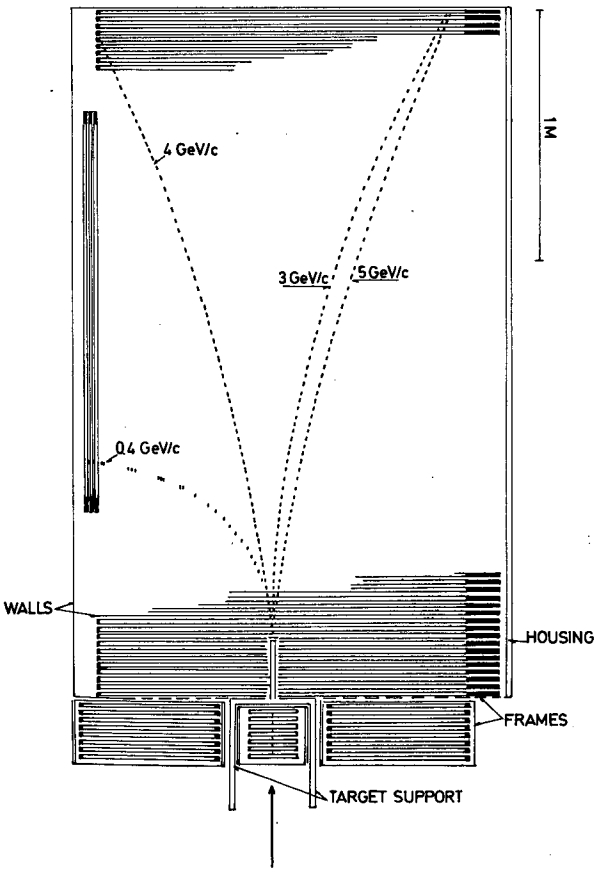


FIG. 16. Top view of target and spark chamber. CERN Omega Project.

Figure 12(b) shows the results obtained. A Berkeley Bubble Chamber group then found that the A_2^+ was not split [12(a)]. The forward leg of the BNL Double-Vee spectrometer was used to detect the decay products of the A_2^- produced in the reaction: (20 GeV/c) $\pi^- + p \rightarrow A_2^- + p$ and demonstrated that the A_2 was not split [12(b)].

A generally similar and simultaneous experiment at CERN which had 25% worse resolution and a 25% better statistical error also concluded the A_2^- meson was not split [12(c)].

A more direct repeat of the CERN missing-mass exponent by a Northeastern-SUNY group [12] using the BNL OLDF also came to the conclusion that the A_2 meson was not split [12(d)].

Thus it is clear that these techniques have had a sizable impact on modern high-energy physics research.

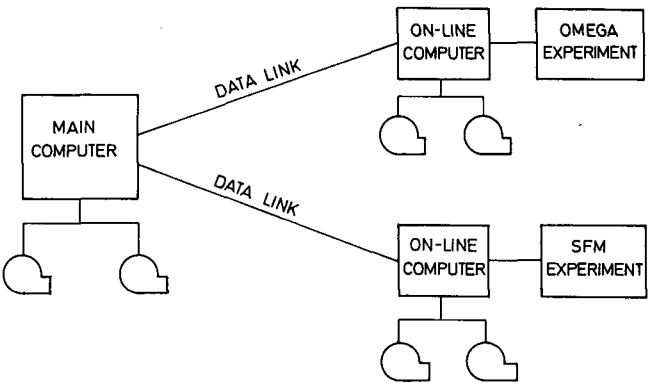


FIG. 17. System layout for computer system associated with the CERN Omega project. The main computer is a C II 10070 (French Σ 7). The smaller on-line computers are EMR 6130.

The SLAC 20-GeV electron accelerator has used a number of magnetic spectrometers with counters to measure the angle and momentum of particles scattered or produced from a target which is bombarded by high-energy electrons or photons [13].

The on-line computer system consists of an SDS9300 with 32K of core, an extensive set of peripherals, a disc monitor, and a multiplexed interface system able to communicate with over 300 devices each having up to 24 bits of information. The computer, however, is dedicated to one experiment at a time. Figure 13 shows a plan view of the spectrometer. Figure 14 shows the interface to the SDS9300.

The next planned important step found in these techniques can be characterized as multi-particle magnetic spectrometers with near 4π-solid-angle coverage. Figure 15 shows such a system being developed at BNL [14]. It is expected to become operational in two to three years. The same automatic computer data analysis will be employed and the BNL OLDF will be utilized. Figure 16 shows the CERN Omega project which is a generally similar device but in its first stage will employ optical spark chambers. A plumbicon video system is being developed to automate the data analysis. Figure 17 shows the on-line computer system. Figure 18 shows one typical module of a modular multi-particle spectrometer system proposed by the author for use at NAL [16].

It is clear that these new techniques already have had an enormous impact on the progress of physics since they allow for more than four or five orders of magnitude increase in data rates accompanied by much greater systematic precision than heretofore attainable and also allow automatic data processing and analysis.

With the advent of the multi-particle spectrometer, another dramatic step forward will be taken. It is clear that only with a sufficiently powerful and versatile on-line computer complex one can possibly undertake the debugging and successful use of these complex devices. Thus the on-line computer technique has come a long way since its inception.

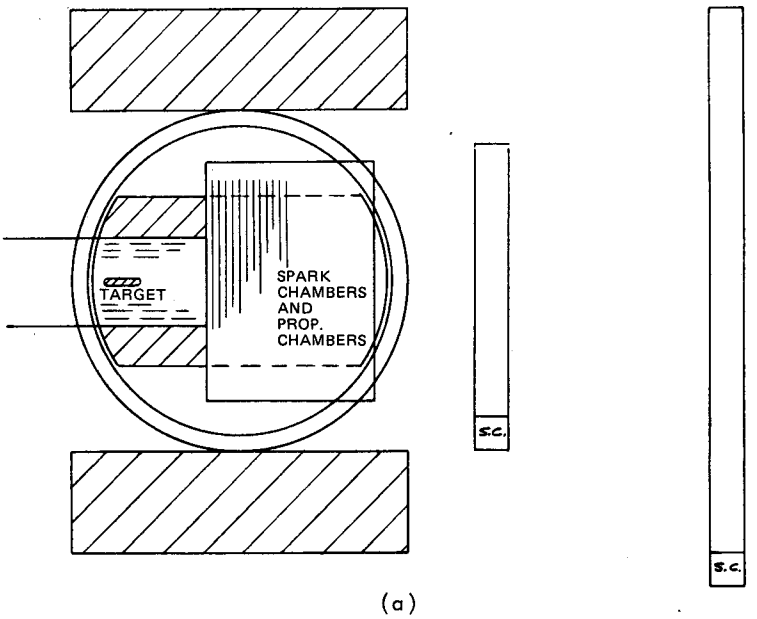


FIG. 18(a). A proposed MK II arrangement.

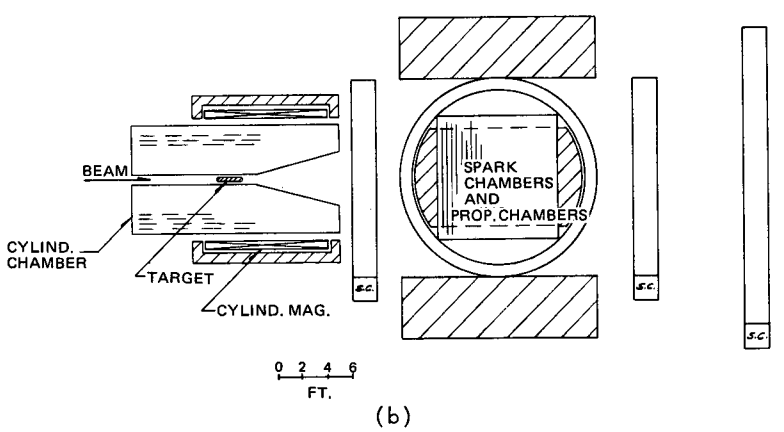


FIG. 18(b). The MK II preceded by a cylindrical magnet (i.e. field along beam) for analysing wide-angle particles.

ACKNOWLEDGEMENTS

The author wishes to thank Professor Abdus Salam, and Professor L. Bertocchi, and the many members of the ICTP staff for their co-operation during his stay there.

REFERENCES

[1] (a) LINDENBAUM, S.J., in Instr. High Energy Physics (Proc. Conf. CERN, Geneva, 1962) North Holland, Amsterdam (1963) 297; Nucl. Instrum. Meth. 20 (1963) 297.
(b) FOLEY, K.J., LINDENBAUM, S.J., LOVE, W.A., OZAKI, S., RUSSELL, J.J., YEAN, L.C.L., Phys. Rev. Letts 10 (1963) 376, 543 and 11 (1963) 425, 503; Nucl. Instrum. Meth. 30 (1964) 45.

[2] For a general review see LINDENBAUM, S.J., Ann. Rev. nucl. Sci. 16 (1966) 619.

[3] MAGLIC, B.S., COSTA, G., BLEIDEN, R., LEFEBVRES, F., FREYTAG, D., ISELIN, F., SLOTTENHAUR, H., in High Energy Physics (Proc. 12th Int. Conf. Dubna, 1964).

[4] LINDENBAUM, S.J., in Nuclear Electronics (Int. Symp. Versailles, France, 1968); LINDENBAUM, S.J., OZAKI, S., in Data Handling Systems in High Energy Physics (Int. Conf. Cambridge, March, 1970); published by CERN, Geneva (1970).

[5] FOLEY, K.J., JONES, R.S., LINDENBAUM, S.J., LOVE, W.A., OZAKI, S., PLATNER, E.D., QUARLES, C.A., WILLEN, E.H., Phys. Rev. 181 (1969) 1775.

[6] FORD, W.T., PIROUE, P.A., REMMEL, R.S., SMITH, A.J.S., SOUDER, P.A., in Data Handling Systems in High Energy Physics (Int. Conf. Cambridge, March 1970); published by CERN, Geneva (1970).

[7] ANDERSON et al., Phys. Rev. Letts 16 (1966) 855.

[8] LINDENBAUM, S.J., Symmetry Principles of High Energies (1965). Invited papers (1967) 122; published by FREEMAN, W.A. and Co.; Contemporary Physics (Proc. Symp. Trieste, 1968) 2, IAEA, Vienna (1968) 123.

[9] REBKA, J.A., ROTHBERG, G.A., ETKIN, A., GLUDIS, P., GREENBERG, J., HUGHES, V.W., KONDO, K., LEE, D.C., MORI, S., THOMPSON, P.A., Phys. Rev. Letts 24 (1970) 160.

[10] GOVORAN, N.N., INKIN, V.D., KHARZHAVIN, Yu.A., MOSCHORYAKOV, M.G., MOROZ, V.I., POSE, R., SHIGAEV, V.N., SKENDENKOV, V.N., in Data Handling in High Energy Physics (Int. Conf. Cambridge, March 1970); published by CERN, Geneva (1970).

[11] Paper presented by LINDENBAUM, S.J., International Conference in High-Energy Physics, (Int. Conf. Dubna, 1964) (SMORDINSKY, Y.A., Ed.) Page 41P, Atomizdat, Moscow (1966); LINDENBAUM, S.J., Argonne Conf. (Oct. 1968); Experimental Meson Spectroscopy, Philadelphia (May 1970).

[12] (a) ALSTON et al., Phys. Letts 33B (1970) 607.
(b) FOLEY, K.J., LOVE, W.A., OZAKI, S., PLATNER, E.D., LINDENBAUM, S.J., WILLEN, E.H., Phys. Rev. Letts 26 (1971) 413.
(c) GRAYER et al., Phys. Letts 34B (1971) 333.
(d) Northeastern-SUNY Collaboration, Bowen et al., Phys. Rev. Letts (in press).

[13] BOGORSKI, A., Sky Top Conf. on Computer Systems in Experimental Nuclear Physics, published by USAEC CONF-690 301 (1969) 139.

[14] LINDENBAUM, S.J., "Multiparticle Magnetic Spectrometer Systems with Electronic Digitized Detectors, Sec. Int. Conf. Experimental Meson Spectroscopy, Philadelphia, May 1-2 (1970)", (BELTAY, ROSENFELD, Eds). The Brookhaven M.P.S. is essentially a further development of the MKI(b) described in this reference.

[15] CERN-OMEGA Project, Working group report.

[16] See Ref.[14].

MULTI-PARTICLE HIGH-ENERGY REACTIONS

S.P. RATTI
Istituto di Fisica
and
Sezione INFN, Milano,
Italy

Abstract

MULTI-PARTICLE HIGH-ENERGY REACTIONS.

1. Introduction; 2. Number of variables involved in a collision; 3. The crucial problem of strong interactions; 4. Some examples of Non-Monte-Carlo methods; 5. Typical computer experiments; 6. Search for sensitive quantities; 7. Longitudinal phase-space (LPS) analysis; 8; An example of the choice of 3n-5 variables; 9. Concluding remarks.

1. INTRODUCTION

The nature of this contribution is fundamentally different from that of the paper by Professor Lindenbaum in these Proceedings.

Professor Lindenbaum has faced the problem of the computer used in performing an experiment in order to collect information as automatically as possible.

Our task is different, if not, in a sense, opposite. Given occasionally an experiment performed by using a fully automatic set-up, the experiment may be finished but the interpretation of the experimental results may be just at the beginning. Sometimes, the crude experimental information has a straightforward direct interpretation in terms of physical quantities; more often, the physical information is included in "an unknown function of all the measured quantities". This is particularly true in the case of high-energy collisions leading to many particles in the final state.

Before going into details, it is, at least, worth mentioning why high-energy collisions leading to many particles in the final state are important [1]. This is simply because at high energies the cross-section for inelastic processes is about 75% of the total cross-section so that the comprehension of these inelastic final states is relevant for an understanding of the phenomena taking place in the collision. Moreover, from a theoretical point of view, since 1961, the contribution of inelastic processes to unitarity has been considered essential to understand the asymptotic behaviour of "two-body collisions".

2. NUMBER OF VARIABLES INVOLVED IN A COLLISION

Let us first face the naive problem of counting how many independent variables are involved in a high-energy collision between particles a and b, producing n particles c_i ($i = 1, 2, \ldots, n$)

$$a + b \to c_1 + c_2 + \ldots + c_n = \sum_{i=1}^{n} c_i \qquad (1)$$

Of each particle we "measure at best the momenta" $\vec{p}_i$ and the energies E_i, i.e. the "four-momenta" $P_i \equiv (p_i, E_i)$.

This adds up to 4N quantities

In a given reference frame $O \equiv (x, y, z)$ we have

$$\vec{p}_i = \vec{p}_{x_i} + \vec{p}_{y_i} + \vec{p}_{z_i}; \quad p_i = \sqrt{p_{x_i}^2 + p_{y_i}^2 + p_{z_i}^2}$$

If we suppose that the masses m_i of each final particle are known we get n equations between the 4n variables (i.e. n constraints) from $E_i = \sqrt{p_i^2 + m_i^2}$.

This leaves $(4N - N) = 3N$ variables

To reaction (1) the energy-momentum conservation laws have to be applied.

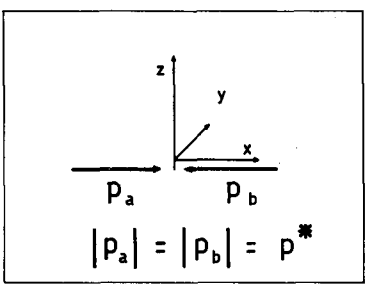


FIG. 1. Reference frame.

For the sake of simplicity, let us assume, once for all, the overall centre-of-mass system, as a reference frame (Fig. 1), so that

$$P_a \equiv (p^*, 0, 0, \sqrt{p^{*2} + m_a^2}); \quad P_b \equiv (-p^*, 0, 0, \sqrt{p^{*2} + m_b^2})$$

In such a reference frame energy-momentum conservation reads:

$$\sum_{i=1}^{n} \vec{p}_i = 0 \qquad \text{(a)}$$

$$\sum E_i = E_a + E_b \qquad \text{(b)}$$

we then have 4 additional constraints and

This gives $(3N - 4)$ independent variables

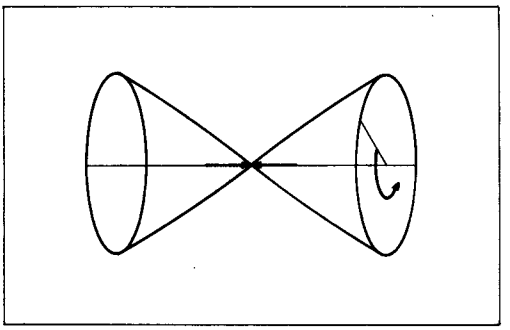


FIG.2. Rotation around the beam axis.

Finally, in most experiments neither the incoming particle a nor the target particle b has a definitely oriented spin (unpolarized beams and targets) so that rotation around the x-axis of Fig. 1 (rotation around the beam axis) does not change the physics, at all (see Fig. 2).

This assumption introduces an additional constraint which reduces the number of independent variables by one:

This finally gives (3N - 5) independent variables

Thus, the major problem is that of choosing the most convenient (3N - 5) variables on the basis of which one can describe the function (of all measured, independent, variables) containing the physical information. There is, of course, no unique choice, and it is often a pure matter of taste which angles, momenta, invariant masses etc. are chosen.

Professor Lohrmann, in his contributon to these Proceedings, gives a large variety of possibilities.

The common undiscussed starting point is, however, the set of 4 N four-momenta of the N final particles (plus the four momenta of the two colliding particles which are assumed here to be known).

At this point, it is clear that the very elaborate data handling needed requires an intensive use of the computer as an indispensable tool for the analysis of the physical content hidden in an experiment.

It is more than obvious that the complexity of the problem increases as the number of final particles increases.

Let us take a "two-body collision"

$$a + b \rightarrow c + d \qquad (2)$$

and the elastic scattering

$$a + b \rightarrow a + b \qquad (2a)$$

The number of independent variables is

$$N = 3N - 5 = 1$$

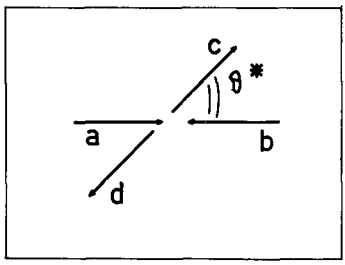


FIG. 3. Scattering angle θ^*.

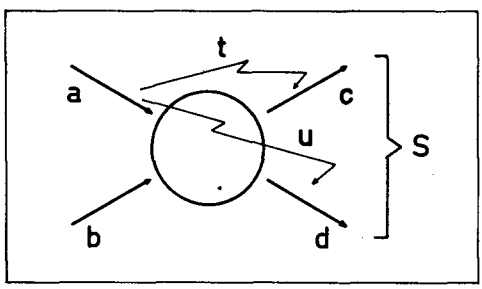


FIG. 4. Mandelstam variables.

One variable is enough to describe the process; e.g. the scattering angle θ^* in the centre-of-mass system (Fig. 3). Nonetheless, as a matter of principle, one can ask two naive questions:

a) which variable (which angle)?

b) which reference frame?

For the simple "two-body" case the use of Lorentz-invariant quantities is very helpful.

Given any four-momentum $P \equiv (\vec{p}, E)$ the dot products are Lorentz-invariant

$$P \cdot P = p^2 - E^2 = -m^2 \qquad (3)$$

In reaction (2), as sketched in Fig. 4, using four-momenta in the centre-of-mass system P_a, P_b, P_c, P_d, for which the energy-momentum conservation reads

$$P_a + P_b = P_c + P_d \qquad \text{(4 equations)}$$

one can define three invariant quantities (Mandelstam variables, see Fig. 4)

$$s = (P_a + P_b)^2 \equiv (P_c + P_d)^2 \quad \text{(square of the total energy in the centre-of-mass system)} \tag{4}$$

$$t = -(P_c - P_a)^2 \equiv -(P_d - P_b)^2$$

$$u = -(P_d - P_a)^2 \equiv -(P_c - P_b)^2$$

It is easy to show that the following equation holds:

$$s + t + u = m_a^2 + m_b^2 + m_c^2 + m_d^2 \tag{5}$$

Now since we suppose that s is known (experiment performed at a given energy $\sqrt{s}$, or at a given momentum, in the centre-of-mass p^*), most often the Lorentz-invariant variable $-t$ is used instead of θ^* (Fig. 3) which is not relativistically invariant.

From Eqs (4) and for the elastic scattering (2a) it is straightforward to derive the formula

$$-t = 2p^{*2} \quad (1 - \cos\theta^*) \tag{6}$$

An enormous amount of experiments has proved that for "two-body collisions" t is very sensitive to the dynamics of the process.

The situation is, however, not so simple for higher multiplicity of particles in the final state.

For reaction (1), we have

$$n = 3 \qquad N = 3n - 5 = 4$$

$$n = 4 \qquad N = 3n - 5 = 7$$

$$n = 5 \qquad N = 3n - 5 = 10$$

and, taking the definition of s from relations (4), we have n different t's (and, equivalently, n different u's):

$$t_i = -(P_a - P_{ci})^2 \tag{7}$$

Thus, the choice of the N independent variables most sensitive to the dynamics of the process is no longer a naive problem.

3. THE CRUCIAL PROBLEM OF STRONG INTERACTIONS

The bulk of the problem of understanding high-energy collisions (2) is that of understanding what happens when a large amount of energy is concentrated in a very small volume (of order $10^{-13} \times 10^{-13} \times 10^{-13}$ cm^3, the interaction volume, – volume of a nuclear particle) for a very short time (of order 10^{-23} s, the interaction time, i.e. the time needed to cross the nuclear dimensions, 10^{-13} cm at the speed of light). Now,

A) there does not exist, as yet, a complete and well established theory of strong interactions;

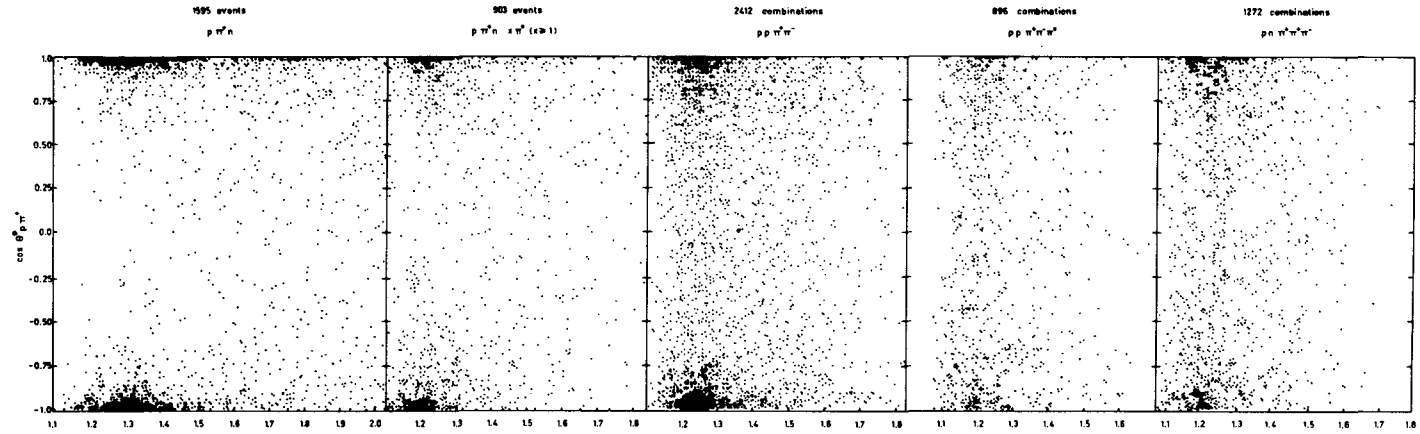


FIG. 5. Scatter diagrams of $\cos\theta^{*}_{p\pi^+}$ versus $M_{p\pi^+}$ for reactions (8) at 4 GeV/c.

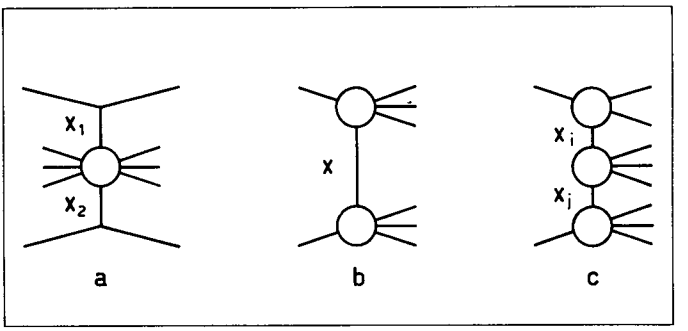


FIG.6. Pionization, fragmentation and more complex processes.

B) the number of independent variables is such that there is no end to the number of physical quantities (effective masses, momentum transfers – squared angular momentum and momentum correlations, etc.) which can be measured experimentally and compared with the theoretical predictions.

Under the present circumstances the computer, through data analysis, is the only tool useful in searching for those variables which most explicitly contain the physical information.

One example [1] may be interesting in order to show a typical picture of a set of high-energy collisions. Consider the following reactions:

$$pp \rightarrow pn\pi^+ \tag{8a}$$

$$\rightarrow p n \pi^+ x \pi^0\text{'s} \qquad (x = 1, 2, \ldots\ldots) \tag{8b}$$

$$\rightarrow p p \pi^+ \pi^- \tag{8c}$$

$$\rightarrow p p \pi^+ \pi^- \pi^0 \tag{8d}$$

$$\rightarrow p n \pi^+ \pi^- \pi^+ \tag{8e}$$

studied at an incident momentum of 4 GeV/c in the laboratory system ($s = 9.2$ GeV^2).

In all reactions (8) there is an abundant production of the nucleon isobar ($p\pi^+$) with mass M = 1236 MeV, width Γ = 120 MeV, spin J = 3/2 and isotopic spin I = 3/2, known as Δ (1236).

Figure 5 shows the correlations between the production angle θ^* of the ($p\pi^+$) system and its mass M ($p\pi^+$) – an old-fashioned way of presenting the data. One observes a striking forward-backward collimation of the produced ($p\pi^+$) system (peripheral collision) and this is only one of the very many aspects of all reactions (8).

Now, how are the things going?

i) Is there essentially pionization (Fig. 6a)?
ii) Is there limiting fragmentation (Fig. 6b)?
iii) Are there more sophisticated mechanisms (Fig. 6c)?
iv) Which "quanta" of the interacting field are exchanged in any of the pictures shown in Fig. 6?

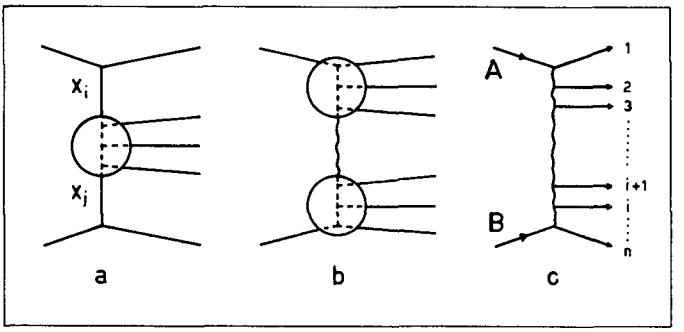


FIG. 7. Structure of "fireballs".

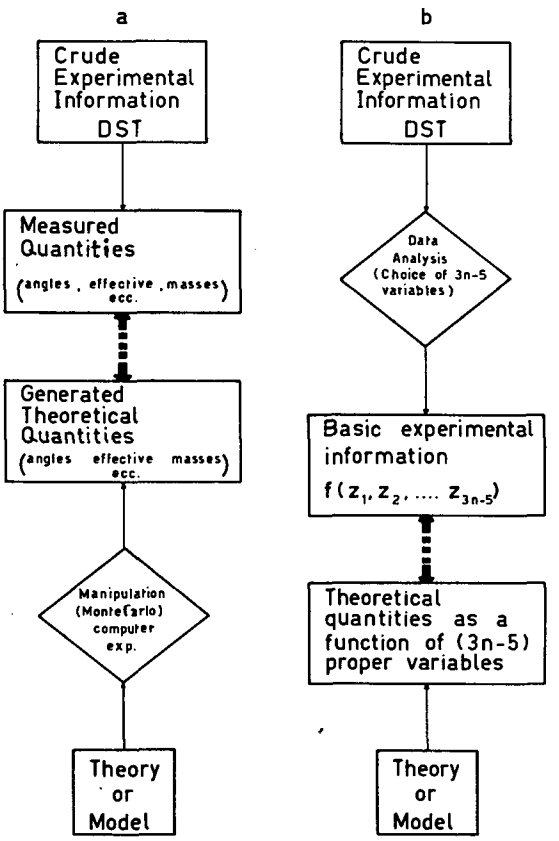


FIG. 8. Ideal flow diagrams of two possible approaches.

The circles drawn in Figs 6 are called "fireballs" by cosmic ray physicists [3]. Several ambitious theorists propose a description of the "inner part" of the fireballs in terms of exchange of definite quanta (quasi-particles, Regge poles, etc.) as sketched in Fig. 7a, b, by suggesting that the real mechanism at work is a "multiperipheral process" as shown in Fig. 7c.

In this framework, the computer may be the key to the possible solution of the problem. Since the only two true statements which can be made are:

A) a set of experimental informations on a given reaction is a data summary tape (DST) containing the n four-vectors ($\vec{p}_i$, E_i) of the n final particles,

B) Any true theory (or, more realistically, any good model) has to describe all experimental features,

the computer might be the key to the possible solution of the problem in two ways, in a sense one opposite to the other (Fig. 8).

The first possibility is, at first sight, the most straightforward:

A) Take the theory (i.e. the explicit form of the proper transition matrix elements), manipulate the basic theoretical ideas to obtain measurable physical quantities. Most often this leads to computer experiments, in which, by means of a Monte-Carlo technique, one generates hypothetical elementary events (i.e. a theoretical set of four-momenta on a DST) to be compared, in all details, with the real ones (Fig. 8a).

B) take the crude experimental information, handle the data in order to build up basic measurable quantities as near as possible to the basic theoretical ideas. This leads to an "a-priori" choice of the (3n - 5) independent variables and to a proper data analysis (Fig. 8b).

Only in very particular cases one can avoid – or reduce to a minimum – Mone-Carlo-generated events and perform explicit theoretical calculations.

In the following, we shall first illustrate the application of simple-minded models to special cases and then briefly discuss the two approaches A) and B) mentioned above.

4. SOME EXAMPLES OF NON-MONTE-CARLO METHODS

In the early sixties, when high energy meant some GeV/c incident momentum in the laboratory system, the typical "inelastic collision" was the production of one or two pions, e.g. something like the reactions (8a) and (8c):

$$pp \rightarrow pn\,\pi^+$$

$$pp \rightarrow pp\,\pi^+\,\pi^-$$

If one goes back to the Yukawa theory of nuclear interactions the "quantum" of the nuclear field was assumed to be the pion; thus, for reaction (8a), a good picture could be the one drawn in Fig. 9 with the exchanged quantum equal to a pion (one-pion exchange model (OPE)) [4].

At the vertex A one has essentially a πN scattering process, while at the vertex B, one has

$$-t_{25} \equiv (P_5 - P_2)^2 \neq -m_\pi^2 \qquad (9)$$

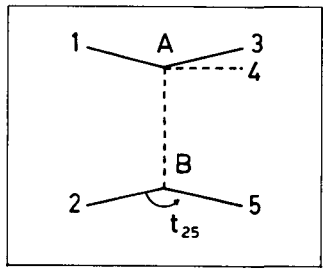


FIG. 9. Pion exchange.

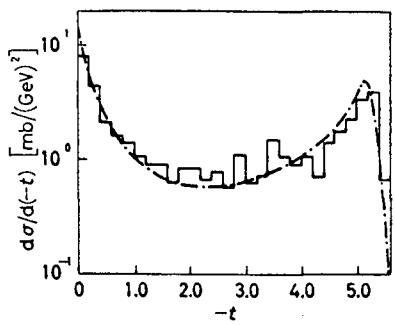


FIG. 10. Differential cross-section $d\sigma/dt$ for reaction (8a) at 4 GeV/c. Dash-dotted line is the OPE prediction.

i.e. i) the exchanged quantum is a "virtual pion" (off-the-mass-shell), ii) the scattering process at the vertex B is not a "real pion-nucleon scattering".

The OPE-model leads to a production amplitude of the type

$$|A(s,t)|^2 = \frac{d\sigma}{d|t|}(s,t) \simeq \frac{F(t)}{(t - m^2_{exch})^2} E_{Lab}^{(2J_{exch}-2)} \qquad (10)$$

where m_{exch} is the mass of the exchanged quantum, J_{exch} is its spin, $F(t)$ is an "a-priori" unknown function of the Lorentz invariant (measurable) quantity t_{25}.

From relation (10) we obtain at fixed s

$$\frac{d\sigma}{d|t|} \simeq F_0 \exp(+Bt) \times \text{small corrections} \qquad (10')$$

On the other hand, given S_{34} (the square of the effective mass of the πN system "3 4") one accounts for the vertex A by using the real πN scattering at different S_{34} (i.e. assuming $-t_{25} \approx m_\pi^2$). This kind of calculations is done by using a computer time of the order of 10 - 100 s.

Figure 10 shows the comparison of the experimental data with the prediction of the model for reaction (8a) at $S = 9.5$ GeV^2 [5] ($P_{lab} = 4$ GeV/c).

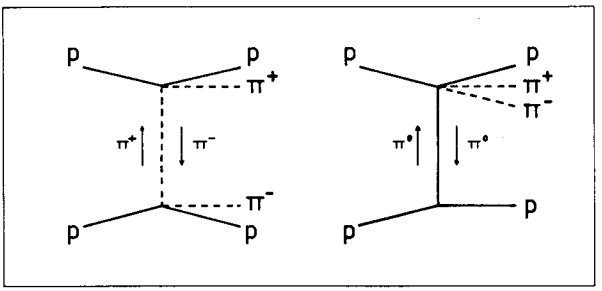


FIG. 11. Two possible OPE diagrams contributing to reaction (8c).

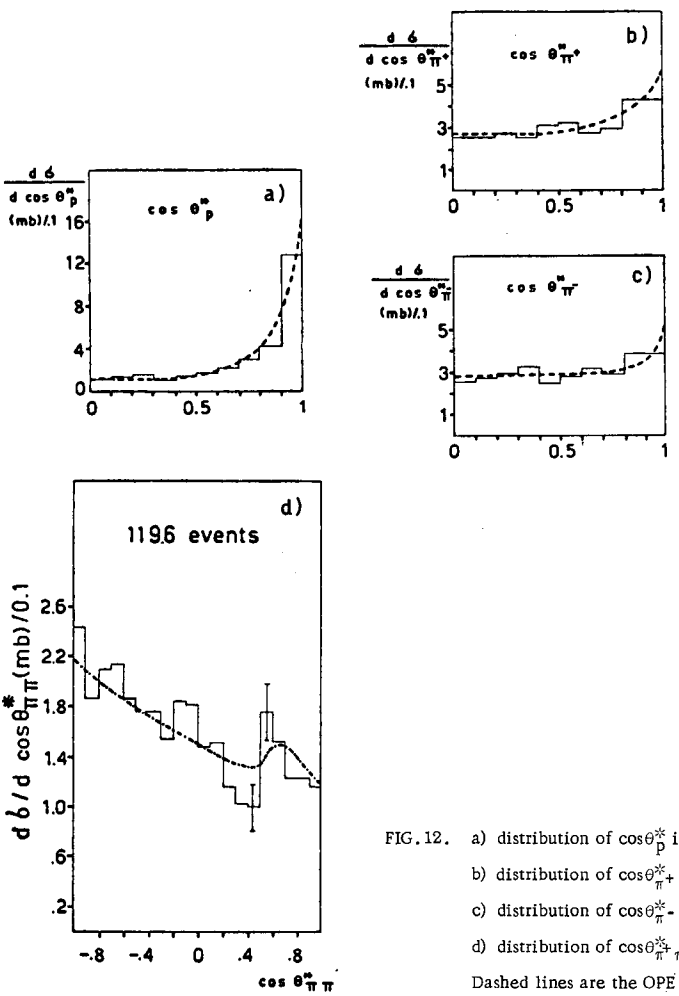


FIG. 12. a) distribution of $\cos\theta^*_p$ in reaction (8c) at 4 GeV/c;
b) distribution of $\cos\theta^*_{\pi^+}$ in reaction (8c) at 4 GeV/c;
c) distribution of $\cos\theta^*_{\pi^-}$ in reaction (8c) at 4 GeV/c;
d) distribution of $\cos\theta^*_{\pi^+\pi^-}$ in reaction (8c) at 4 GeV/c;
Dashed lines are the OPE predictions.

The same procedure can be used for reaction (8c) where there are two possible processes (Fig. 11).

In this case, use can be made of the known processes

$$\pi^+ p \longrightarrow \pi^+ p$$

$$\pi^- p \longrightarrow \pi^- p$$

and, with some more sophisticated manipulation,

$$\pi^0 p \longrightarrow \pi^+ \pi^- p$$

Again with proper off-the-mass-shell corrections and with a computer time of 100 - 1000 s it is possible to reproduce the experimental data reasonably well.

Figure 12 shows the comparison of the prediction of the model with the experimental data at 4 GeV/c [1, 6].

Even the angular correlation between the two final pions (Fig. 12d) is not badly reproduced.

Now, the model makes sense (is it an accident?), as long as $J_{exch} = 0$, otherwise, from relation (10), $\sigma \to \infty$ when $S \to \infty$ if $J_{exch} \neq 0$ and the "one-particle-exchange-model" is a disaster.

5. TYPICAL COMPUTER EXPERIMENTS

To avoid this difficulty and to have

a) $\sigma \to k$ (or, possibly, to zero) when $S \to \infty$,
b) $C(t) \approx \exp(bt)$ (dumping effect)

several theoreticians have suggested the use of the Regge-pole idea [7], i.e. the exchanged quanta are not only "quasi-particles off-the-mass shell" but they are "off-the-angular-momentum-shell", as well. In the amplitude $A(s, t)$, J_{exch} does not enter as an integer but rather as $\alpha_x(t)$, a real function of t (Regge trajectory), which becomes integer only when $-t = m^2_{exch}$ (non-physical region).

In this picture, the production amplitude becomes

$$A(s, t) = F(t)\,(S/S_0)^{\alpha_x(t)} \qquad (11)$$

where x indicates the trajectory exchanged (say, in a graph of the type shown in Fig. 9); S_0 is a scaling factor (usually put equal to 1) and the differential cross-section takes the form

$$\frac{d\sigma}{d|t|} \approx C(t)\, E_{lab}\, (2\alpha_x(t) - 2) \qquad (12)$$

with

$$\alpha_x(t) = \alpha_x^0 + \alpha' t + \ldots \qquad \text{(linear trajectories)} \qquad (13)$$

TABLE I. CROSS-SECTIONS FOR π^-p INTERACTIONS AT 11 GeV/c

No.	Reaction	Cross-section (mb)	Number of events
(1)	$p\pi^-\pi^0$	0.66 ± 0.10	1002
(2)	$n\pi^+\pi^-$	0.80 ± 0.10	1518
(2')	$n\pi^+\pi^-$, no ρ^0, no f^0 (°)	0.60 ± 0.09	1146
(3)	$p\pi^+\pi^-\pi^-$	1.14 ± 0.11	3403
(3')	$p\pi^+\pi^-\pi^-$, no Δ^{++}, no ρ^0 (°)	0.51 ± 0.05	1527
(3a)	$\Delta^{++}\pi^-\pi^-$ (°)	0.32 ± 0.03	940
(3b)	$p\rho^0\pi^-$ (°)	0.35 ± 0.04	1053
(4)	$p\pi^+\pi^-\pi^-\pi^0$	1.24 ± 0.11	3922
(5)	$n\pi^+\pi^+\pi^-\pi^-$	0.68 ± 0.05	1993
(6)	$p\pi^+\pi^+\pi^-\pi^-\pi^-$	0.29 ± 0.03	286
(7)	$p\pi^+\pi^+\pi^-\pi^-\pi^-\pi^0$	0.71 ± 0.07	816
(8)	$n\pi^+\pi^+\pi^+\pi^-\pi^-\pi^-$	0.30 ± 0.03	235

(°) Cuts: ρ^0: $0.68 < M(\pi^+\pi^-) < 0.88$ GeV,
where π^- with low momentum transfer from the incoming π^- is taken in the case of reactions (3') and (3b).
f^0: $1.17 < M(\pi^+\pi^-) < 1.37$ GeV.
Δ^{++}: $1.12 < M(p\pi^+) < 1.34$ GeV.
(The cross-sections correspond to the number of events used in the histograms.)

The dumping factor can be preserved and lot of work has been done in recent years following such an approach.

If one is ambitious enough one can even try to describe multi-particle processes by means of graphs similar to those sketched in Fig. 7c [8].

However, a lot of complication arises.

Let us give an explicit example. Among others, a large collaboration between the laboratories of Genova, Hamburg, Milano and Saclay [9] has analysed the reactions shown in Table I (which includes also a proper "reaction label" as a guide-line to the following Figs 14 and 15 at 11 GeV/c incoming momentum of the negative pions, using a model proposed by Chan, Łoskiewicz and Allison [10].

The production amplitude for each multiperipheral graph is (in a rather hidden way, it includes the principle mentioned above)

$$A_n = \prod_{i=1}^{n-1} \left(\frac{g_i s_i + K}{S_i + 1} \right) (s_i + 1)^{\alpha_i^0} \left(\frac{s_i}{b_i} + 1 \right)^{\alpha_i' t} \qquad (14)$$

where n is the multiplicity of the final particles, $s_i = (P_i + P_{i+1})^2 - (m_i + m_{i+1})^2$, $t_i = (P_A - \sum_{r=1}^{i} P_r)^2$, and g_i, b_i, k are constant and almost arbitrary.

Not enough, the production amplitude A_n^* for the process leading to n final particles has to be obtained by adding all possible permutations of the final particles, i.e.

$$A_n^* = \sum_{j=1}^{m} |A_n^j|^2 \qquad (14')$$

where j = 1, 2, ...m, are the permutations of the final particles. (To be specific in the simplest case, for the reaction $\pi^- p \to \pi^- \pi^0 p$ one has three amplitudes $A_3^{(1)}$, $A_3^{(2)}$, $A_3^{(3)}$ corresponding to the permutations sketched in Fig.13.)

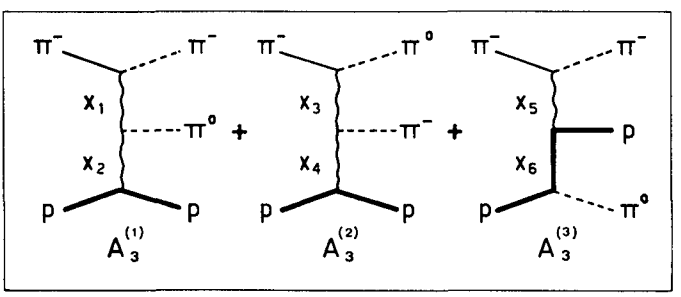


FIG. 13. Double Regge diagrams contributing to reaction (1) of Table I.

Now what is worth pointing out here is that such a computer experiment requests of the order of some hundred hours of IBM 360/75.

In addition, the number of free parameters (unknown) is increasing rapidly.

Nonetheless, if one considers comparisons of the experimental data with the computed expectations (Figs 14-17), some of them are satisfactory, the rest is not.

The following comments can be made:

1) Single-particle distributions (Figs 14 to 20) are relatively well reproduced;
2) The transverse momenta (Figs 14-17, left-hand distributions) are most insensitive to the various final states;
3) The computed expectations have statistical errors (shown explicitly by the full lines in Fig.18) which are not negligible in spite of the enormous amount of computer time used.
4) Some correlations, such as resonant states, are obviously not reproduced if they are not explicitly introduced in the amplitudes (Figs 19-20).

With respect to this last item, it turns out that the explicit introduction of resonant states does improve the agreement between experimental data and the computer experiment, but, on the other hand, does not change

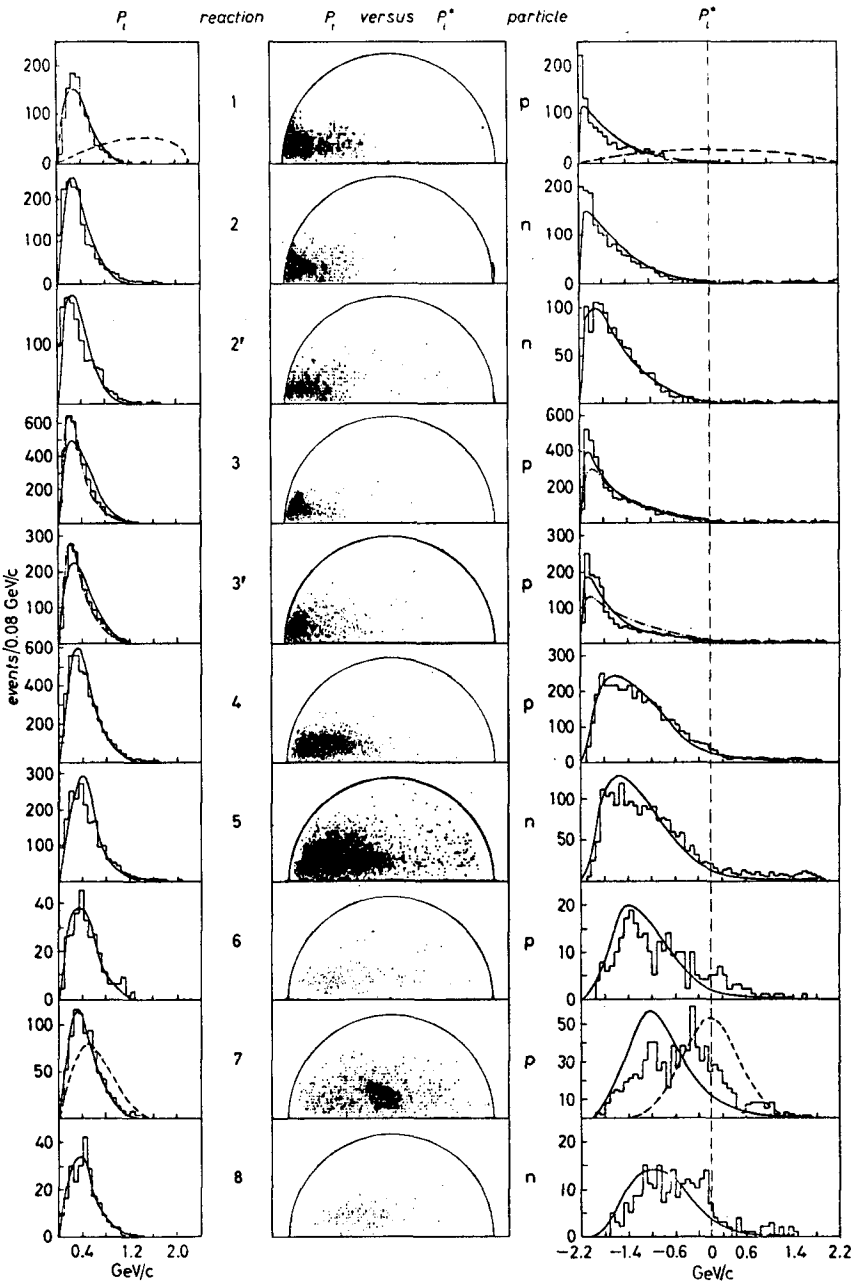


FIG. 14. Transverse momentum q versus longitudinal momentum r_l in the centre-of-mass system together with the projections on both axes for the nucleon of the indicated reactions. Predictions of the multi-Regge model (——), the statistical model (---) and the OPE model (-.-.-) are given in this and in the following Figs 15-17 (reaction number indicated in Table I).

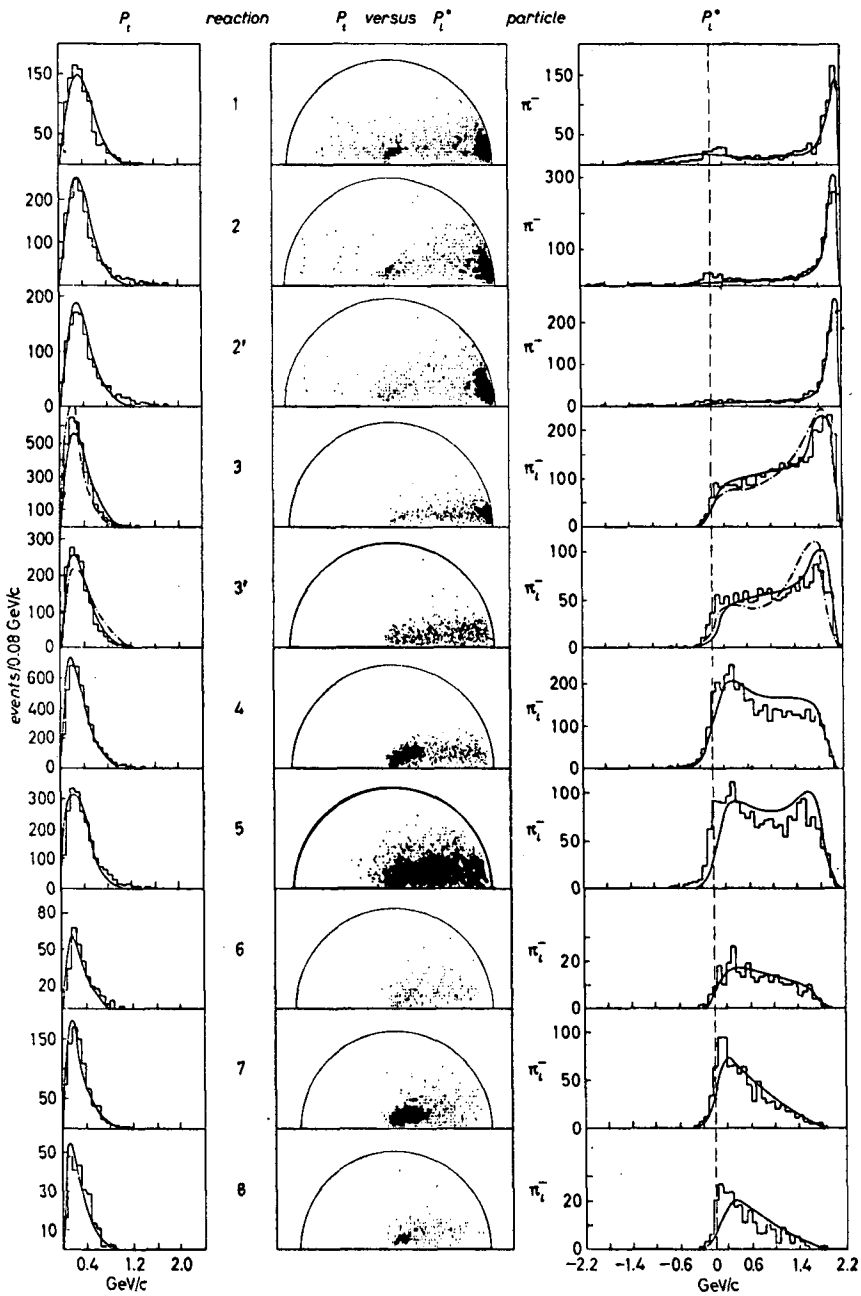


FIG. 15. q versus r_i for the "leading" π^- of the indicated reactions. (Reaction number indicated in Table I.)

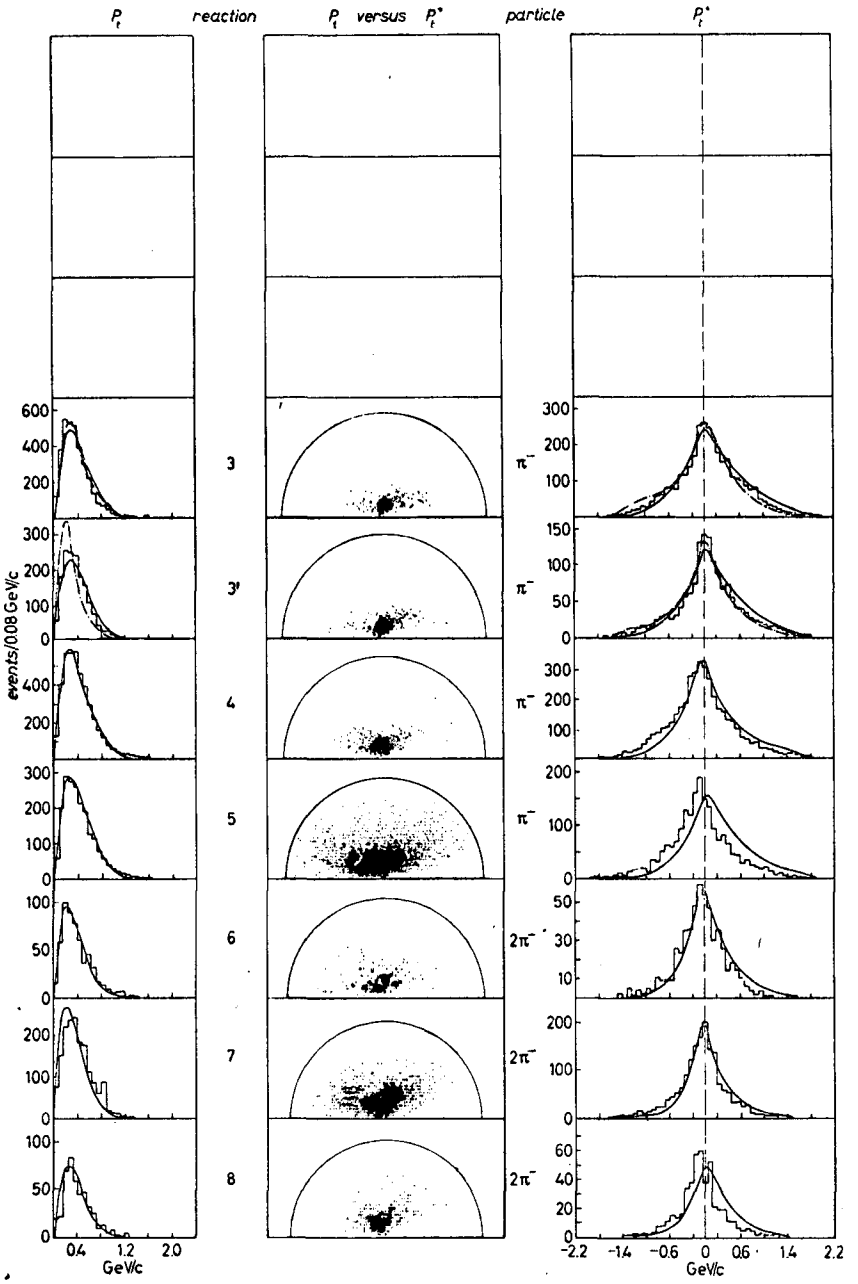


FIG. 16. q versus r_i for the "remaining" π^-'s of the indicated reactions. (Reaction number indicated in Table I.)

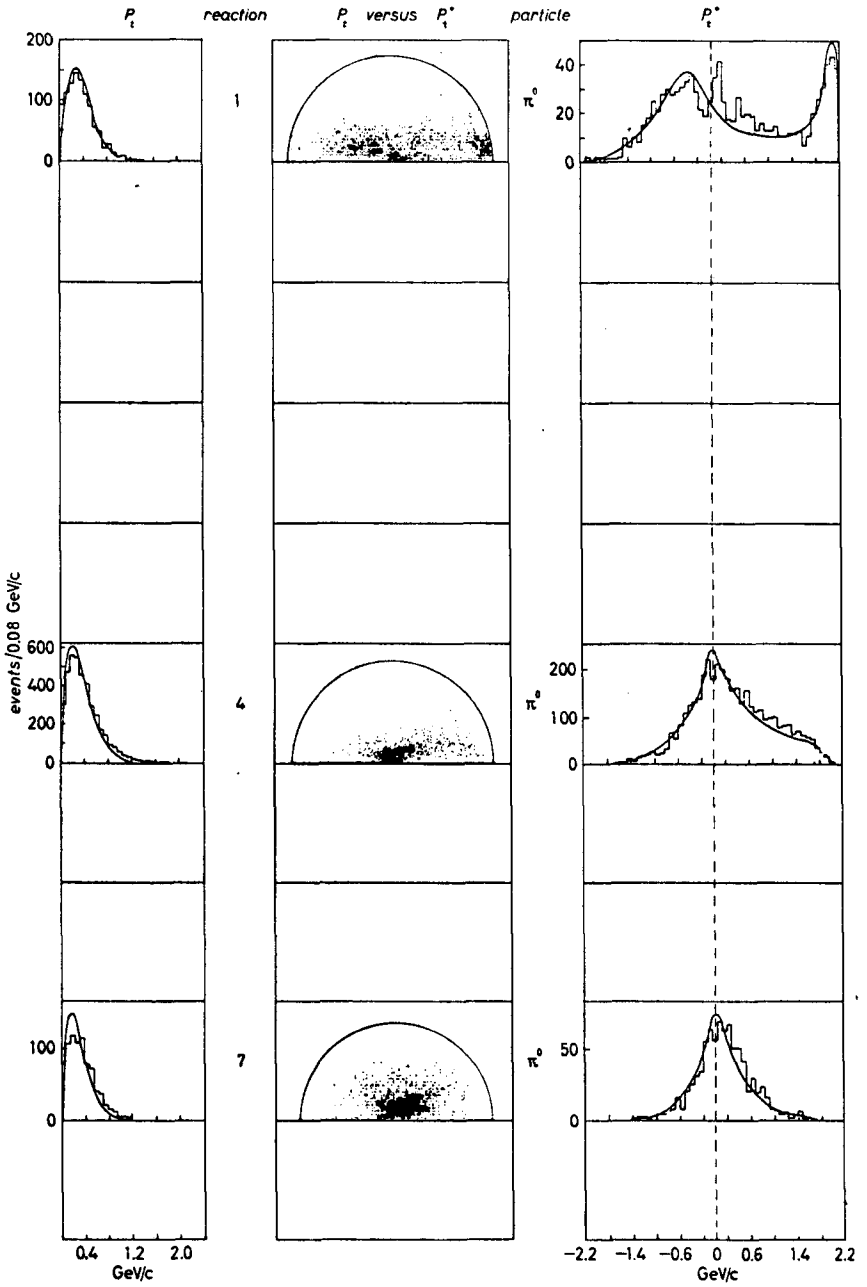


FIG. 17. q versus r_i for the π^0 of the indicated reactions. (Reaction number indicated in Table I.)

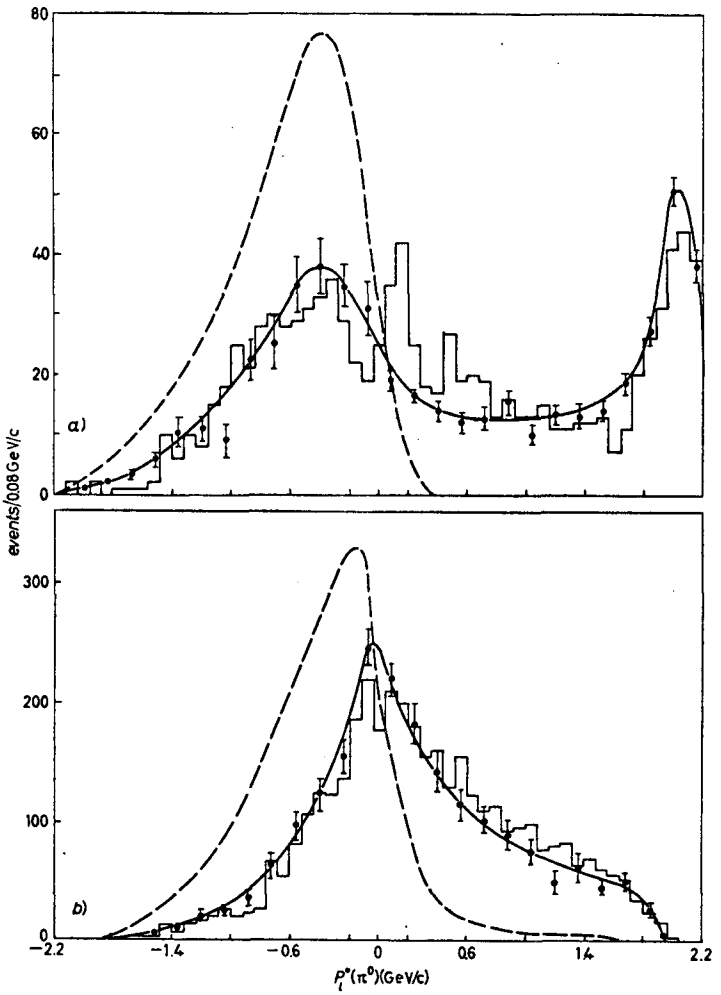


FIG. 18. Longitudinal momentum for the π^0 of a) $\pi^- p \to p\pi^-\pi^0$ 1002 events; b) $\pi^- p \to p\, 2\pi^-\pi^+\pi^0$ 3922 events. Calculations with P (---) and ρ (——) exchange. The points with error bars are the results of the Monte-Carlo calculations for ρ-exchange.

the situation dramatically, e.g. Fig. 21 shows the effect of the explicit introduction of the $\Delta(1236)$ isobars in the reaction

$$pp \longrightarrow \pi^+ \pi^- pp$$

at 8 GeV/c on the single-particle distributions [11].

Is this situation satisfactory? Of course not, because

A) too many parameters mean too few basic physical information;

B) single-particle distributions are not enough "selective" to improve our knowledge; and finally,

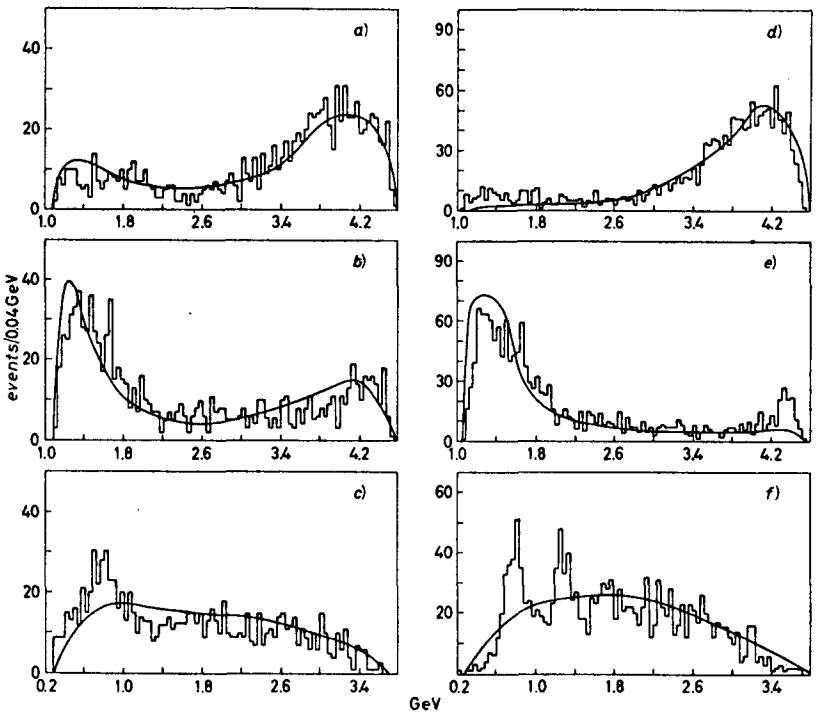


FIG. 19. a) $M_{p\pi^-}$; b) $M_{p\pi^0}$ and c) $M_{\pi^-\pi^0}$ for $\pi^-p \rightarrow p\pi^-\pi^0$ d) $M_{n\pi^-}$, e) $M_{n\pi^+}$; and f) $M_{\pi^+\pi^-}$ for $\pi^-p \rightarrow n\pi^+\pi^-$. The full lines are the Monte-Carlo predictions of the model.

C) the peripheral idea derived from the experimental observation of limited transverse-momentum distributions gave origin to the peripheral models. Now ask the question: is there a "non-peripheral process"? The answer may be: yes, the annihilation process! But even the annihilation process [12]

$$\bar{p}p \longrightarrow n\pi \quad (n = 3, 4, 5, 6, 7) \quad \text{at } 5.7 \text{ GeV/c}$$

shows a transverse momentum distribution (Fig. 22) of the type

$$\frac{d\sigma}{dp_\perp} = k \exp(-p_\perp/0.17) \tag{15}$$

which is quite similar to the dozens of reactions shown in Figs 14 to 17!

Thus, the "direct approach" of the computer experiment may not be the best way of reaching the truth, and an alternative method has to be adopted.

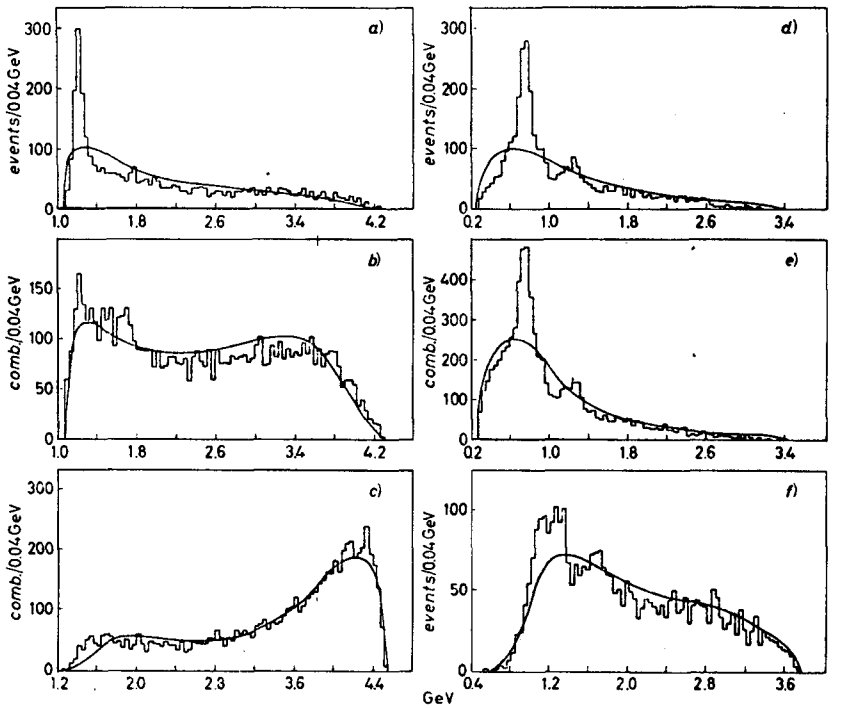


FIG.20. a) $M_{p\pi^+}$; b) $M_{p\pi^-}$; c) $M_{p\pi^+\pi^-}$; d) $M_{\pi^+\pi_f^-}$; e) $M_{\pi^+\pi_s^-}$; f) $M_{\pi^+\pi^-\pi^-}$ for $\pi^-p \to p\pi_f^-\pi_s^-\pi^+$. The combinations with both negative pions are taken in Fig. 20b. The full lines are the Monte-Carlo predictions of the model.

6. SEARCH FOR SENSITIVE QUANTITIES

As was mentioned in the previous section, to study collisions of elementary particles leading to high multiplicity of final bodies (particles or resonances), computer experiments implying heavy manipulation of the basic theoretical ideas (Fig. 8a) and going down directly to the crude experimental information show two weak points:

a) They are computer-time-consuming and relatively inaccurate (statistical errors shown in Fig. 18);

b) They introduce a number of almost-free parameters which are not quite kept under control throughout the manipulation.

On the other hand, experimental results show that some measurable quantities are not very sensitive to the theoretical assumption (i.e. transverse momenta), while some other quantities depend strongly on the mechanisms at work.

The observation that the transverse momenta ($\vec{p}_\perp$) of the final particles are not dramatically affected by the dynamics of the high-energy process is relevant (although not strictly necessary) for the following considerations.

We have now the clear feeling that

a) the parameter s (square of the total centre-of-mass energy) is important;

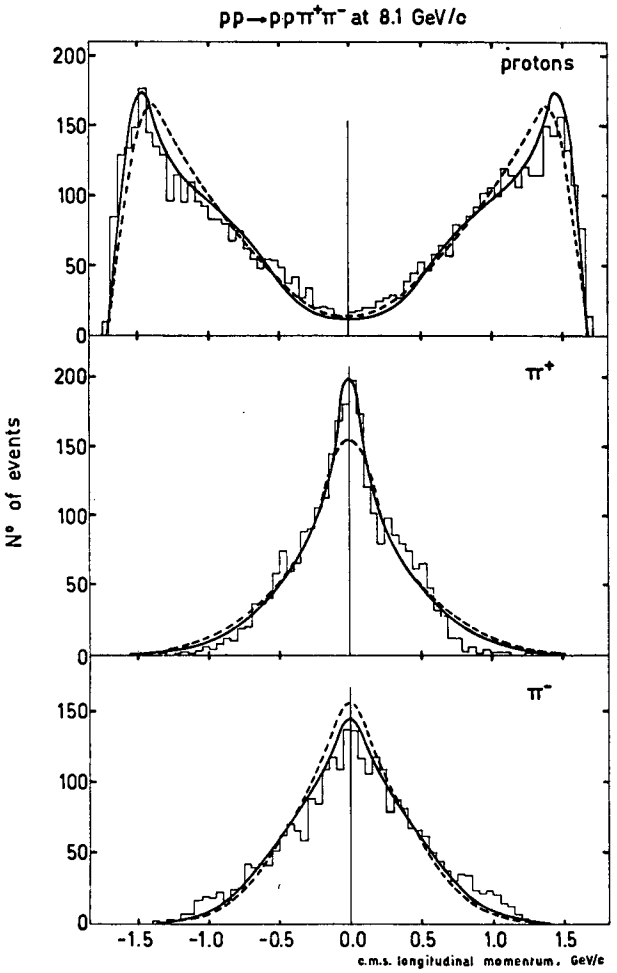


FIG.21. Longitudinal momentum distribution for the final particles of reaction (8c) studied at 8 GeV/c. Dashed lines: Monte-Carlo prediction using formula (14). Full lines: Monte-Carlo prediction correcting for the presence of the $\Delta^{++}(1236)$ and $\Delta^{+}(1236)$ resonances.

b) $|\vec{p}_\perp|$ is not a good parameter, and

c) $P_\parallel$, the longitudinal component of the three-momenta along the beam axis, is most sensitive to the particular characteristics of the collisions.

Following the flow-diagram of Fig. 8b we have now to try a proper handling of the experimental data in order to obtain the experimental information as a function of the most proper 3n-5 variables. To assume transverse momenta as some of the 3n-5 variables is of no primary use to understand the mechanisms at work in high-energy collisions. This experimental fact adds the advantage that the number of crucial variables may be less than 3n-5.

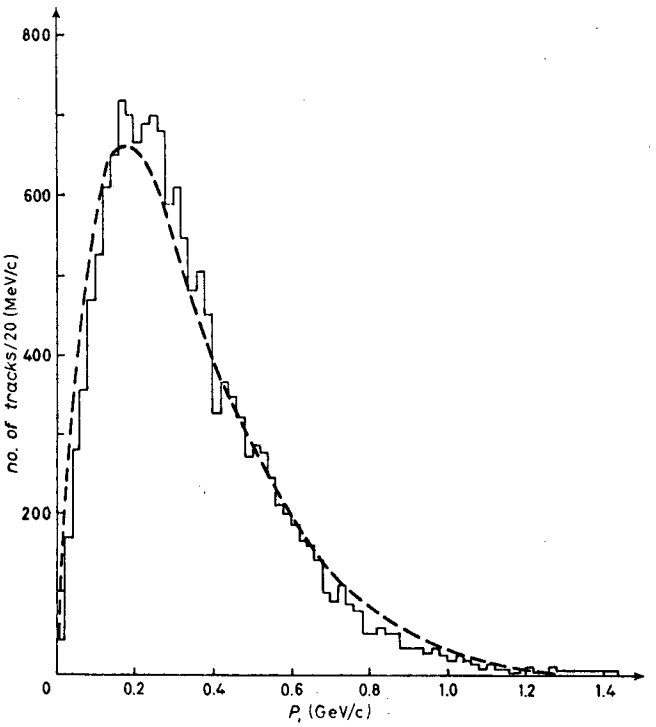


FIG.22. Distribution of the transverse momenta for all charged pions produced in annihilations with four or six charged pions (14916 tracks at 5.7 GeV/c). The dashed line represents the best-fitted curve of the type (15).

Before going into the real choice of the 3n-5 variables, which, as has already been mentioned, is not a unique set, let us consider how far from the crude experimental information (DST) one can go "experimentally".

To do this, we go back to an undergraduate introductory course on elementary particles [13].

Basic knowledge of quantum mechanics and a little mathematics is needed.

All the information about a physical system is contained in a state vector $|\psi\rangle$ [13b]; then for a reaction

$$a + b \rightarrow \sum_{i=1}^{n} c_i$$

let us call $|i\rangle$ the state vector of the initial system (a+b) and $|f\rangle$ the state vector of the final state ($\sum c_i$).

From S-matrix theory [13a] the transition matrix elements read

$$\langle f|S|i\rangle = \delta_{f,i} + i(2\pi)^4 \delta(Q^i - Q^f) N \langle f|T|i\rangle \tag{16}$$

where Q^i is the total four-momentum of initial system, Q^f the total four-momentum of final system, and $\delta(Q^i - Q^f)$ imposes energy-momentum conservation; N is a normalization factor.

In the following, we shall define as transition matrix element (called A in formulas (10), (12), (14), for example) the quantity

$$M = \langle f|T|i\rangle \tag{17}$$

It is clear that M contains all the physical information on the process under consideration.

On the basis of relations (16) and (17), quantum mechanics allows the calculation of the cross-section σ for a given process as

$$\sigma(\text{mb}) = \Phi \int_{\Omega} |M|^2 \, dV_R \tag{18}$$

where Φ is the flux factor given by $1/P_A^{cms}\sqrt{s}$ and dV_R is an element of the phase-space volume.

From formula (18) we shall observe the following:

A) M contains all the physics;

B) Φ contains information on the external conditions (beam-momentum, total energy $\sqrt{s}$, etc.);

C) dV_R contains the kinematics.

Since we know a priori, when we perform an experiment, everything about Φ we have to try to get rid of kinematics — a factor which may mask physical information — in order to reach as directly as possible information on $|M|^2$.

After having done this, we can measure $|M|^2$ as a function of a good set of 3n-5 variables.

To get rid of kinematics, we have to work out the form of the Lorentz-invariant phase-space volume dV_R explicitly.

For reaction (1) we have

$$a + b \rightarrow \sum c_i$$

$\sum_{i=1}^{n} \vec{p}_i = 0$, and $\sum_{i=1}^{n} E_i = \sqrt{s}$ are all quantities in the overall centre-of-mass system.

$$dV_R = \frac{1}{\prod_{i=1}^{n} E_i} dV = \delta\left(\sum_{i=1}^{n} \vec{p}_i\right) \delta\left(\sqrt{s} - \sum_{i=1}^{n} E_i\right) \prod_{i=1}^{n} \left(\frac{d_3\vec{p}_i}{E_i}\right) \tag{19}$$

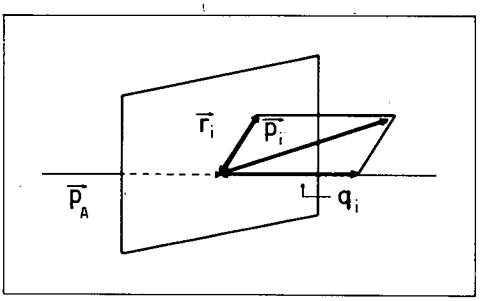


FIG. 23. Vector decomposition.

As a matter of principle our requirements are completely fulfilled if we measure the fully differential cross-section

$$\frac{d\sigma}{dV_R} = \Phi |M|^2 \tag{20}$$

or, equivalently,

$$|M|^2 = \Phi^{-1} \frac{d\sigma}{dV_R} \tag{21}$$

This is easily done in an experiment using an infinite running time and an infinite number of experimental information, to begin with. (That is, having an extremely large number of $(3n-5)^3$ elementary volume units dV_R, the density of experimental points drastically goes to zero.)

7. LONGITUDINAL PHASE-SPACE (LPS) ANALYSIS

We shall postpone to the next section the example of an explicit choice of a full set of 3n-5 variables, at least, in the case of three bodies produced in a collision.

Here we want to point out the possibility of a compromise, on the basis of which one can make progress along the lines of the diagram shown in Fig. 8b.

For the sake of simplicity we shall suppose that all final particles are detected, i.e. all final p_i are known. (Nowadays such a collision is called an exclusive reaction [14].)

Let us decompose the vectors $\vec{p}_i$ into their longitudinal components q_i and transverse components $\vec{r}_i$ (Fig. 23):

$$\vec{p}_i = \vec{q}_i + \vec{r}_i \tag{22}$$

$\vec{q}_i$ is, of course, always, by definition, directed along the beam axis.

The momentum conservation laws split into two parts:

$$\sum \vec{p}_i = 0 \begin{cases} \sum \vec{r}_i = 0 \\ \sum q_i = 0 \end{cases} \tag{23}$$

where the q_i can be neither all positive nor all negative. The energy conservation $E_i = \sqrt{s}$ now reads

$$\sum \sqrt{q_i^2 + r_i^2 + m_i^2} = \sqrt{s} = \sum |q_i| \sqrt{1 + \left(\frac{r_i^2 + m_i^2}{q_i^2}\right)} \tag{24}$$

Note that $\sqrt{s} \cong 2 p_A$ for large s.

Let us comment on Eq. (24): First of all, we can arrange the q_i's:

$$q_i < q_2 < \ldots < q_\ell < 0 < q_{\ell+1} \ldots q_n \tag{25}$$

Secondly, let us keep in mind the experimental idea that the $\vec{r}_i$'s are typically "small" compared to the q_i's. According to relations (25) this cannot be true for "all i"; nonetheless, without losing the generality of the argument, we follow the suggestion put forward by Van Hove [15] and consider the limiting situation $r_i \equiv 0$, $m_i \equiv 0$.

Equations (23) and (25) now become

$$\sum q_i = 0 \tag{23'}$$

$$\sum |q_i| = K \tag{26}$$

where $K \neq \sqrt{s}$.

Equation (26) defines a geometrical polyhedron; the limiting representative points lie on a hyperplane H_{n-1} with n-1 dimensions (because of the constraint (23)) in a space S_n of n dimensions where n is the number of final particles. Now, we have:

A) Since $m_i \neq 0$ for most of the particles, the "true" hypersurface K_{n-1} will be contained in H_{n-1};

B) Since $r_i \neq 0$, the physical points will not actually lie on the hypersurface K_{n-1}, but rather in its vicinity.

Let us explicitly mention two examples for n = 3 and 4. If we take n = 3, Eqs (23') and (26) read

$$q_1 + q_2 + q_3 = 0 \tag{23''}$$

$$|q_1| + |q_2| + |q_3| = K \tag{26'}$$

Equation (26') defines a hexagon (Fig. 24). This is easy to understand from elementary considerations. If the q_i's (i = 1, 2, 3) are defined to be positive,

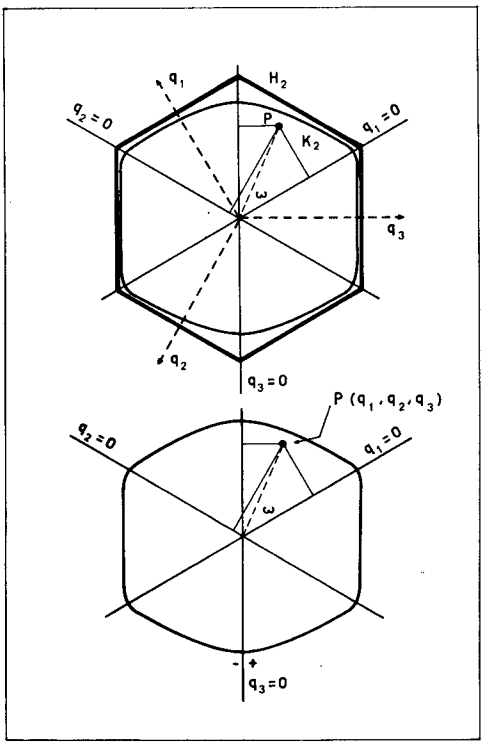


Fig. 24. The hexagon.

Eq. (26') presents the property of the triangle (the sum of the distances of any point, inside a triangle, from the boundary lines, is a constant); since the q_i's are relative numbers (and all of them cannot have the same sign), there are six possible permutations of the particles 1, 2 and 3. From Eq. (23'') immediately the hexagon follows.

As is shown in Fig. 24, the representation of the "point" $P \equiv (q_1, q_2, q_3)$ is obtained by measuring the distances q_i from three axes forming 120° and properly oriented:

a) since $m_i \neq 0$ the actual hypersurface K_2 is inside the hexagon H_2;

b) since $r_i \neq 0$ the real points lie inside K_2.

Now (at fixed s), one variable is sufficient to identify the longitudinal configuration of a three-body collision completely.

It is worth mentioning that, as a matter of principle, we disregard the r_i's only temporarily. If, in addition, we impose, e.g. $0 < r_i < \epsilon$, we select a "shell" or a "slice" all along K_2.

Let us now take n = 4:

$$q_1 + q_2 + q_3 + q_4 = 0 \qquad (23''')$$

$$|q_1| + |q_2| + |q_3| + |q_4| = K \qquad (26'')$$

Equation (26'') defines a cuboctahedron H_3 (Fig. 25).

Triangular faces represent longitudinal configurations with one value of $q_i < 0$ ($q_i > 0$) and the other q_j's negative (positive).

Rectangular faces represent longitudinal configurations having two positive q_i's and two negative q_j's.

a) Since $m_i \neq 0$, the actual hypersurface K_3 is inside the cuboctahedron H_3.

b) Since $r_i \neq 0$, the real points lie inside K_3.

In this case, two variables are sufficient to identify the longitudinal configuration of a four-body collision completely.

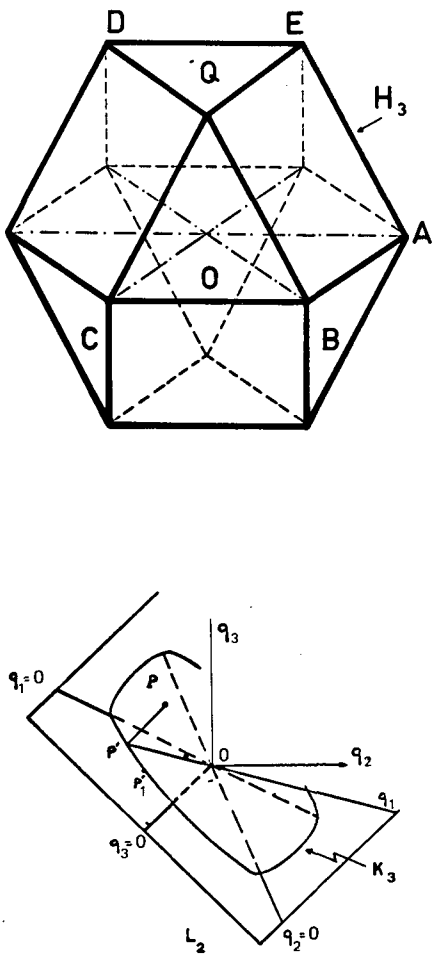


FIG. 25. a) The cuboctahedron; b) the surface K_3 inside H_3 is indicated.

7.1. Further simplification

It is obvious that in Eq. (26) K is related to $\sqrt{s}$. For exclusive reactions one can introduce [16] "reduced longitudinal momenta x_i" which eliminate the explicit dependence of the size of the polyhedron on $\sqrt{s}$.

In fact,

$$\sum |q_i| = 2Q \tag{27}$$

is a known quantity.[1]

Define

$$x_i = \frac{2q_i}{\sum |q_i|} \tag{28}$$

Using the new reduced variables, Eqs (23) and (26) read

$$\sum_{i=1}^{n} x_i = 0 \tag{29}$$

$$\sum_{i=1}^{n} |x_i| = 2 \tag{30}$$

The ordering of the q_i's now becomes

$$x_1 < x_2 < \ldots x_\ell < 0 < x_{\ell+1} < \ldots < x_n \tag{25'}$$

and, from (29) and (30)

$$\sum_{i=1}^{\ell} x_i = -1 \qquad \sum_{j=\ell+1}^{n} x_j = +1 \tag{31}$$

By making use of the new variables, x_i's, we can, in a rather elegant way, get rid of kinematics and explicitly derive Eq. (21) given in section 6. We simply have to transform the variables q_i to x_i by making use of

a) the two constraints (31);
b) the fact that Q is not an overall constant but is known, say, "event by event".

In Eq. (19) we have to transform

$$d_3\vec{p}_i = dq_i\, d_2\vec{r}_i \longrightarrow dx_i\, d_2\vec{r}_i$$

[1] Since $\sqrt{s} = 2p_A$ it is useful to introduce the factor 2 so that for $s \to \infty$ $q = \frac{1}{2} \Sigma |q_i| \approx p_A$, the momentum of the incoming particle in the centre-of-mass system.

From relations (27) and (28) follows

$$q_i = Qx_i$$

and

$$dq_i = Qdx_i + x_i dQ$$

From relation (31) follows

$$\sum dx_i = 0 \qquad \sum dx_j = 0$$

$$dx_1 = -\left(\sum_{i=2}^{\ell} dx_i\right) \qquad dx_n = -\left(\sum_{j=\ell+1}^{n-1} dx_j\right)$$

so that the final transformation becomes

$$\prod_{i=1}^{n} dq_i = Q^{n-2}\, dQ \prod_{j=2}^{n-1} dx_j \tag{32}$$

Energy conservation $\delta(\sum E_i - \sqrt{s})$ now reads

$$\delta\left[\sum \sqrt{m_i^2 + r_i^2 + x_i^2 Q^2} - \sqrt{s}\right]$$

and it has to be evaluated at a given Q', i.e. a new Dirac function $\delta(Q - Q')$ has to be introduced.

This leads to a new multiplication factor

$$\frac{1}{\dfrac{\partial(\sqrt{(m_i^2 + r_i^2) + x_i^2 Q_i^2})}{\partial Q}} = \left[Q \sum_{i=1}^{n} \left(\frac{x_i^2}{E_i}\right)\right]^{-1} \tag{33}$$

Thus, finally

$$\prod_{i=1}^{n} dq_i = \frac{Q^{n-3}}{\sum_{i=1}^{n} \left(\dfrac{x_i^2}{E_i}\right)} \prod_{j=2}^{n-1} dx_j \tag{34}$$

From Eqs (21) and (34) immediately follows the expression of the fully differential cross-section:

$$\frac{d\sigma}{\prod_{j=2}^{n-1} dx_j \prod_{i=1}^{n} d_2\vec{r}_i} = \frac{\Phi Q^{n-3}}{\prod_{i=1}^{n} E_i \left[\sum_{i=1}^{n} \dfrac{x_i^2}{E_i}\right]} |M|^2 \tag{34'}$$

Equation (34) still contains the very high dimensionality of the problem but really removes kinematics from the physical information.

For sake of simplicity we define

$$W^{-1} = \frac{\Phi Q^{n-3}}{\prod_{i=1}^{n} E_i \left[\sum_{i=1}^{n} \frac{x_i^2}{E_i} \right]} \tag{35}$$

$$dV_\perp = \prod_{i=1}^{n} d_2 \vec{r}_i$$

From relation (34) follows

$$\frac{d^{n-2}\sigma}{dx_2\, dx_2 \ldots dx_{n-1}} = W^{-1} \int |M|^2 \, dV_\perp \tag{36}$$

If, for the moment, we give up the ambitious idea of studying the fully differential cross-section we have a good chance of obtaining physical information on the value of the transition matrix element averaged over the transverse momenta. (This last limitation is obviously not needed but it is imposed by the limited number of events available in a high-energy experiment.) From formula (36) follows

$$\int |M|^2 \, dV_\perp = W \frac{d^{n-2}\sigma}{dx_2 \ldots dx_{n-1}} \tag{37}$$

where W is a measurable quantity.

This approach provides the possibility of obtaining model-independent information on the main properties of the matrix elements responsible for a given exclusive collision.

7.2. Some experimental results

It is interesting to show some experimental results obtained by using the LPS-analysis. LPS is essentially, first of all, a new kind of "data presentation" which enables us to condense the experimental information in one single plot.

Figure 26 shows the longitudinal configurations of the reaction [17]:

$$\pi^+ p \rightarrow \pi^+ \pi^0 p \tag{38}$$

studied at 8 GeV/c (Fig. 26a), while Fig. 26b shows the main longitudinal characteristics of the sub-sample

$$\pi^+ p \rightarrow \pi^0 \Delta^{++}(1236) \qquad \Delta^{++} \rightarrow p\pi^+ \tag{38'}$$

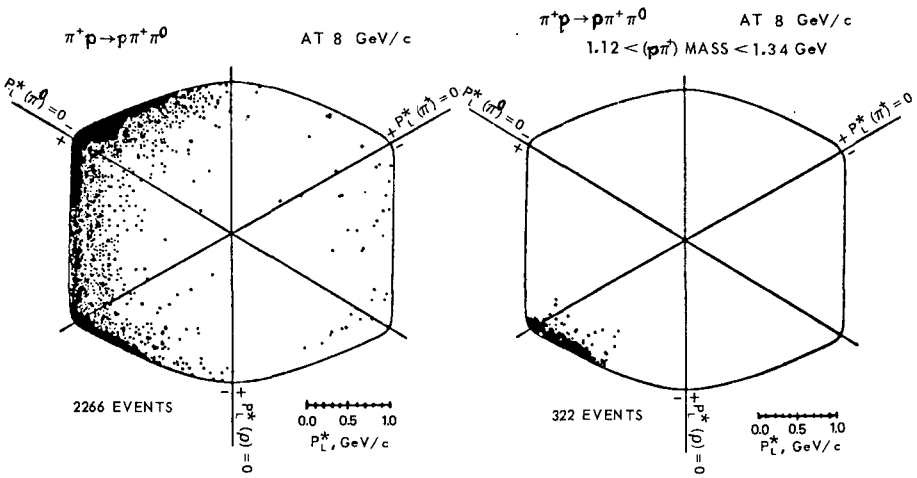


FIG.26. Van-Hove plot for the reaction (38') and $\pi^+p \to \pi^0\Delta^{++}$ at 8 GeV/c.

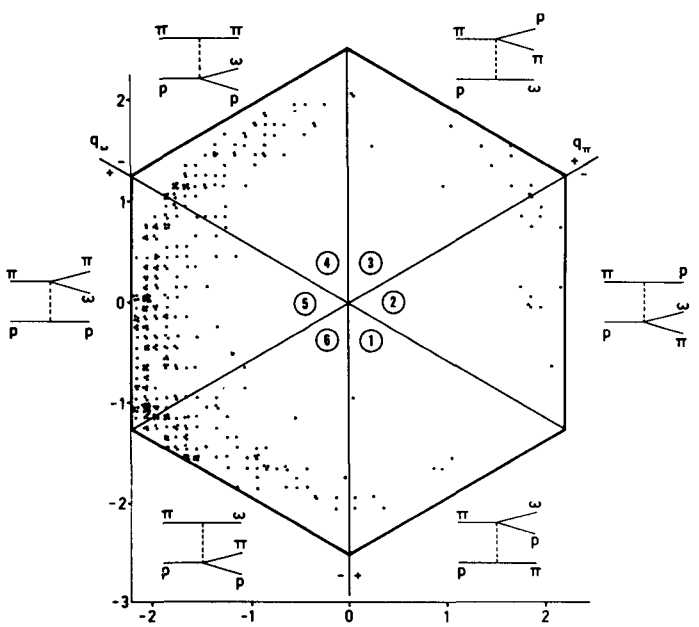


FIG.27. Hexagon for the "quasi-three-body" reaction (38'').

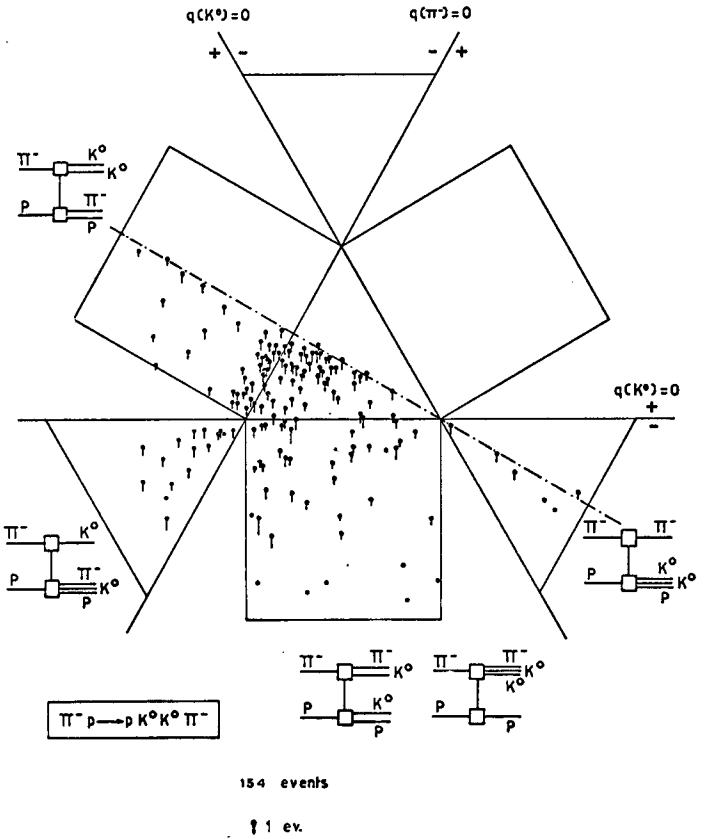


FIG. 28. Unfolded half cuboctahedron for reaction (39).

The forward-backward collimation of particles is clearly seen as well as the almost complete absence of forward going nucleons.

A final particle can be indeed a "resonance" as in the case of the reaction [18]

$$\pi^+ p \rightarrow \pi^+ \omega^0(784)\, p \qquad (38^{\text{II}})$$
$$\omega^0 \rightarrow \pi^+ \pi^- \pi^0$$

studied at 11.7 GeV/c (Fig. 27).

The same kind of "data presentation" can be adopted for a four-body collision. To show an unusual reaction let us take [19]

$$\pi^- p \rightarrow K^0 \bar{K}^0 \pi^- p \qquad (39)$$

studied at 11.2 GeV/c.

First of all, $x_p < 0$ most of the time. By assuming $x_p < 0$, one has to take from Fig. 25 half cuboctahedron.

Such a half cuboctahedron can be unfolded in one plane.

Figure 28 shows the longitudinal configurations of reaction (39). In the figure, the centre-of-mass directions of the final particles are graphically indicated (forward-going particles are coming out from the upper vertex, backward-going particles from the lower vertex). Because of the presence of two identical particles (K^0 and $\overline{K}^0$ are experimentally indistinguishable) half of the figure is empty, i.e. one quarter of the cuboctahedron fully describes the longitudinal configuration of reaction (39). Figure 28 is a three-dimensional representation. In fact, the neutral particles (K^0, $\overline{K}^0$) can escape detection so that the "detection efficiency" introduces non-integer weighting factors.

The equivalent number of events in Fig. 28 is represented by the length of the vertical segments; the basis of the segment indicates the longitudinal configuration of all the four particles representing an event.

So much for the data presentation. Let us now turn to the experimental measurements of the integrated values of the transition matrix elements according to relation (37) in the case of four- and five-body collisions.

Let us first take the reactions

$$\pi^- p \to \pi^- \pi^+ \pi^- p \tag{40}$$

$$\pi^+ p \to \pi^+ \pi^+ \pi^- p \tag{41}$$

Reaction (40) has been compared at 11 and 16 GeV/c [16].

Reactions (40) and (41) have been compared at both energies [20].

To each final particle we assign a proper x_i ordered according to relation (25). There are two identical pions (π_{fast}, π_{slow}), a pion of opposite charge (π_{unlike}) and the proton.

We can then arrange the x_i's as is shown schematically in Fig. 29.

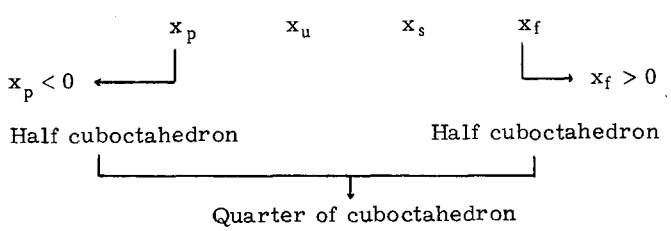


FIG. 29. Arrangement of x_i's.

If we select 'a priori' $x_p < 0$, $x_f > 0$, a plot of x_s versus x_u completely identifies the longitudinal configuration of the collision. From relation (31) and from the definition of the "fast" pion, for the configuration with two "forward"-like pions, we have

$$x_f + x_s = +1$$

and the maximum value of x_s is 0.5.

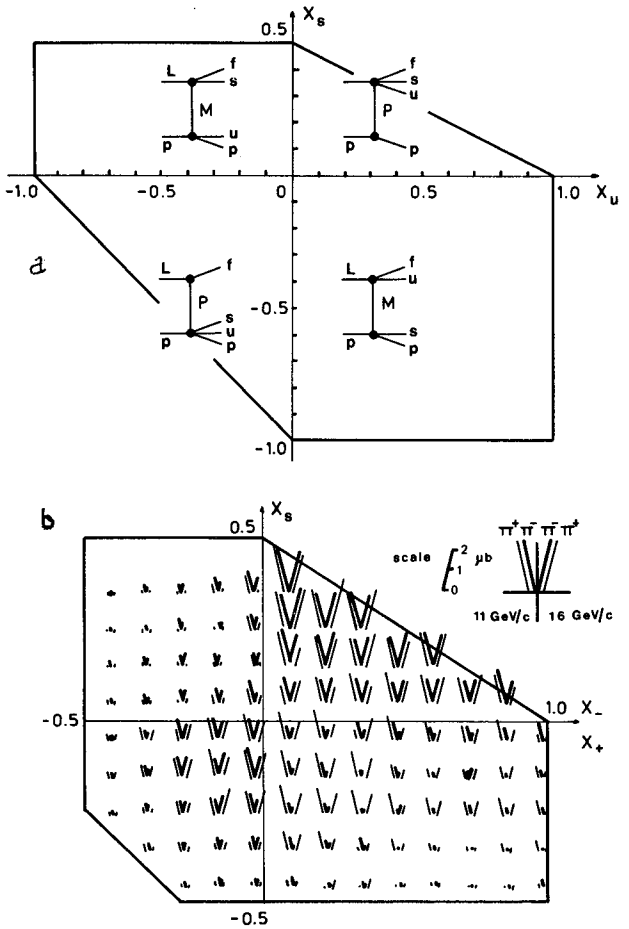


FIG.30. a) Values of σ_W given by Eq.(42) obtained by a plotter and comparing quantitatively reactions (40) and (41) at 11 and 16 GeV/c;

b) Qualitative description of the longitudinal configurations. Upper vertex going forward, lower vertex going backward in the centre-of-mass system.

Figure 30 shows the quantity[2]:

$$\sigma_W = s \int |M|^2 \, dV_\perp = f(x_s, x_u) \qquad (42)$$

for both reactions (40) and (41).

[2] The extra factor s is introduced in order to make the comparison of $|M|^2$ at two different energies easier.

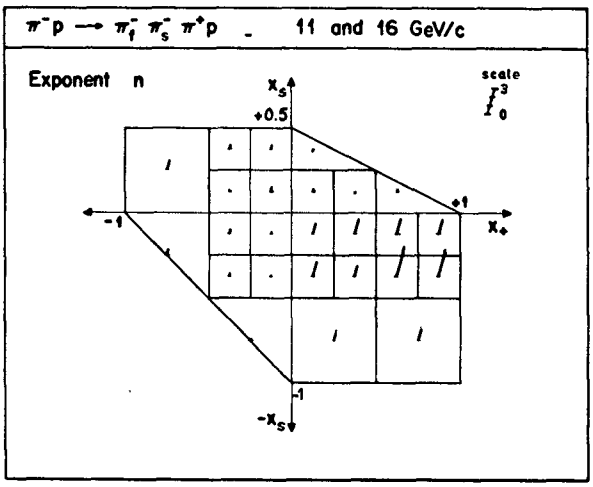


FIG. 31. Slope n of Eq. (44) in different LPS-regions showing the presence of the two diffraction processes ($n \approx 0$ in the triangular regions).

In Fig. 30, one notices immediately that the values of σ_W in the two triangles are energy-independent and isospin-independent, while this is not the case for the values in the two rectangles.

Let us consider the triangles. If there is diffraction dissociation $\Delta_W(11\ \text{GeV}) = \Delta_W(16\ \text{GeV})$ and $\Delta_W(\pi^- p) = (\Delta_W(\pi^+ p))$. Now, Fig. 30 strongly supports this idea. More quantitatively, we measure

$$\Delta_W = \int \sigma_W \, dx_s \, dx_u \tag{43}$$

define

$$\Delta_W = p_{lab}^{-n} \tag{44}$$

and determine

$$n = n(x_s, x_u)$$

for reaction (40). Figure 31 clearly shows the existence of longitudinal configurations (inside the triangles) having $n \approx 0$, giving a model-independent evidence for the diffraction dissociation of the pion

$$D_\pi) \quad \pi p \rightarrow (3\pi) p \tag{45}$$

and the diffraction dissociation of the nucleon

$$D_p) \quad \pi p \rightarrow \pi (p \pi^+ \pi^-) \tag{46}$$

TABLE II. THE RATIOS OF WEIGHTED PARTICLE DENSITIES Δ_W, FOR POSITIVE AND NEGATIVE INCIDENT PIONS, IN THE RESPECTIVE SUB-SECTORS OF THE LPS-REGIONS SKETCHED IN FIG. 30b

Ratio of densities in sector	Value		
	Expected from factorization	observed	
		at 11 GeV/c (Ref.[20a])	at 16 GeV/c (Ref.[20b])
$\Delta^+_W/\Delta^-_W(D_p)$	1	0.89 ± 0.13	1.04
$\Delta^+_W/\Delta^-_W(D)$	1	0.90 ± 0.05	1.03
Δ^-_W/Δ^+_W (I)	Δ^+_W/Δ^-_W(II)	1.94 ± 0.54	2.2 ± 0.4
Δ^+_W/Δ^-_W (II)	Δ^-_W/Δ^+_W (I)	1.63 ± 0.16	2.4 ± 0.3

The comparison of reactions (40) and (41) at the same energy gives other model-independent information. Compare the different regions (in brackets the particles going in the same centre-of-mass direction):

	$\pi^+ p$	$\pi^- p$
D_π)	$(\pi^+ \pi^+ \pi^-)$ (p)	$(\pi^- \pi^- \pi^+)$ (p)
D_p)	$(\pi^+)(\pi^+ \pi^- p)$	$(\pi^-)(\pi^- \pi^+ p)$
I)	$(\pi^+ \pi^+)(p \pi^-)$	$(\pi^- \pi^-)(p \pi^+)$
II)	$(\pi^+ \pi^-)(p \pi^+)$	$(\pi^+ \pi^-)(p \pi^-)$

We measure Δ_W according to Eq. (43) (see Table II).
One immediately obtains $\Delta^+_{D_\pi} = \Delta^-_{D_\pi}$; $\Delta^+_{D_p} = \Delta^-_{D_p}$ and

$$\frac{\Delta^+_I}{\Delta^-_{II}} = \frac{\Delta^+_{II}}{\Delta^-_I} \quad \text{within errors} \tag{47}$$

This experimental observation is quite important because IF M can be factorized, in the two ratios (47) the part of M describing the pion vertex $\pi^\ell \to (2\pi_\ell)$ or $\pi^\ell \to (\pi_\ell \pi_\mu)$ cancels out.

Again, in a model-independent way, one supports factorization properties of M for charge-exchange production processes.

Consider now the five-body reactions

$$\pi^- p \to \pi^- \pi^- \pi^+ \pi^0 p \tag{48}$$

$$\pi^- p \to \pi^- \pi^- \pi^+ \pi^+ n \tag{49}$$

Using the same labelling as in Fig. 29 or Fig. 30 we can cut into slices x_0 (or x^+) and reduce the analysis to the preceding case (Fig. 32).

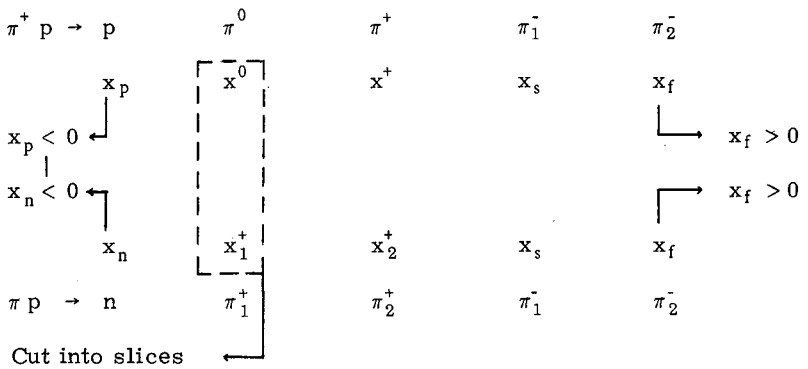


FIG. 32. Cutting into slices.

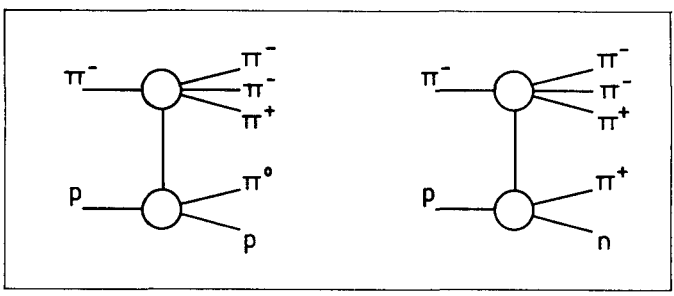


FIG. 33. Possible double diffraction dissociation processes in the reactions (48) and (49).

Now measure Δ_W according to relation (43) for the charge-exchange (Δ_W^n) reaction (49) and the non-charge exchange (Δ_W^p) reactions (48), in the particular configurations shown in Fig. 33.

One may now ask the question: Is there double diffraction dissociation [i.e. $\pi \rightarrow (3\pi)$ and $N \rightarrow (N\pi)$]?

If the answer is yes the ratio of the two cross-sections is that expected for the branching ratio of an isospin state $|\frac{1}{2}, \frac{1}{2}\rangle$ decaying into a pion-nucleon system $|\frac{1}{2}, \frac{1}{2}\rangle$, decaying, in its turn, into $|p\pi^0\rangle = |\frac{1}{2}, \frac{1}{2}\rangle|1, 0\rangle$ and $|n\pi^+\rangle = |\frac{1}{2}, -\frac{1}{2}\rangle|1, 1\rangle$.

Thus, from the Clebsch-Gordan coefficient, we expect

$$R = \frac{\Delta W^p}{\Delta_W^h} = \frac{\sigma_W [(3\pi)(p\pi^0)]}{\sigma_W [(3\pi)(n\pi^+)]} = 0.5 \qquad (50)$$

Figure 34 which presents the value of R at 16 GeV/c [21] as a function of $x^0(x_1^+)$, clearly shows the existence of a particular longitudinal configuration (x^0, $x_1^+ \approx -0.2$) which verifies Eq. (50), proving, in a model-independent way, the existence of a double-diffraction dissociation process [16].

Several other features can be studied by using this technique. What is important to point out here is that a proper choice of the variables (and the

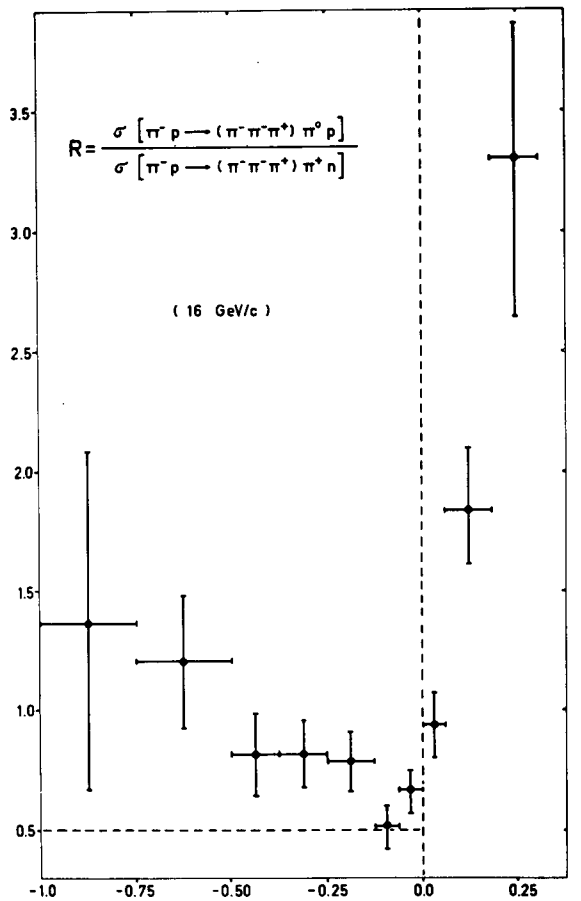


FIG.34. Parameter R of Eq.(50) versus (x^0, x_1^+) indicating the existence of double diffraction dissociation for $(x^0, x_1^+) \approx -0.2$.

computer plays a crucial role in this) allows us to extract extremely important physical information, by properly handling the data accumulated in an experiment.

8. AN EXAMPLE OF THE CHOICE OF 3n-5 VARIABLES

In this section we finally give an example for the choice of a complete set of 3n-5 variables. To reduce the technicalities, we shall limit ourselves to a three-body case. The method has been suggested by Dao et al. [22]. In Ref.[22], also the cases n = 4 and n = 5 are considered explicitly. Of course, the problem is connected with the high dimensionality of the space to be considered and, even if the computer can store properly all the information needed, an easy three-dimensional representation (e.g. in a graphic display) is possible and particularly instructive for n = 3.

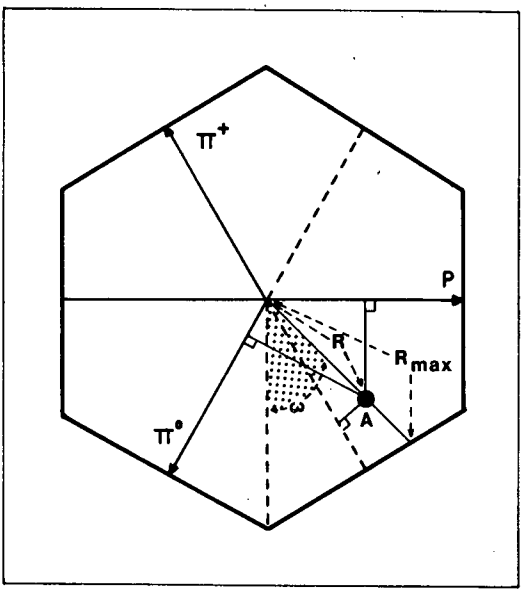


FIG. 35. Explicit representation of a hexagon for reaction (38) indicating the variables defined in Eqs (51) and (52).

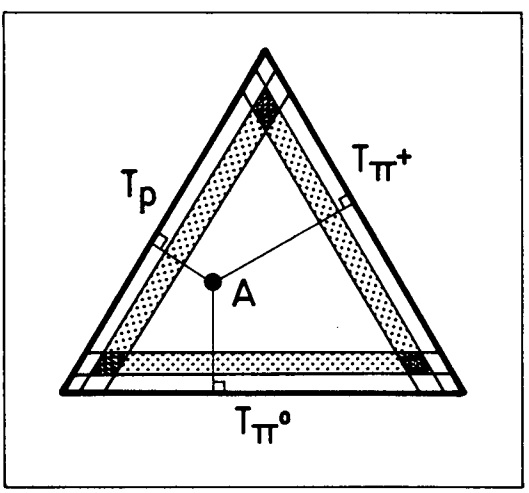


FIG. 36. Dalitz plot for reaction (38). Shaded areas are the "resonance bands"; hatched areas are the "overlapping regions".

Let us go back to our hexagon and consider reaction (38).

As was mentioned in section 7, one variable, the angle ω, is sufficient to identify the longitudinal configuration of the final state.

The Van-Hove angle ω is the first variable

The overall "transverse configuration" can be taken care of by using a parameter depending on the distance R of the representative point from the centre (Fig. 35).

Let us define explicitly the three unit vectors ($\hat{u}_p$, $\hat{u}_{\pi^+}$, $\hat{u}_{\pi^0}$) indicating the "forward" direction of $\vec{p}_p$, $\vec{p}_{\pi^+}$ and $\vec{p}_{\pi^0}$. As was mentioned in section 7, their directions are 120° apart from each other.

$$\begin{cases} \hat{u}_p \equiv (1, 0) \\ \hat{u}_{\pi^+} \equiv \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right) \Longrightarrow (+120° \text{ rotation}) \\ \hat{u}_{\pi^0} \equiv \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right) \Longrightarrow (-120° \text{ rotation}) \end{cases} \tag{51}$$

Let us define, in the reference frame (51) shown in Fig. 35, the distance

$$R = \sqrt{2/3\,(q_p^2 + q_{\pi^+}^2 + q_{\pi^0}^2)} \tag{52}$$

and the Van-Hove angle

$$\omega = \tan^{-1}\left[\frac{q_{\pi^+} - q_{\pi^0}}{\sqrt{3}\, q_p}\right] + \left(\frac{\pi}{2}\right) \tag{53}$$

The "phase" $\pi/2$ is a matter of convention and in Fig. 35 the value $\omega = 0$ is indicated.

Now R_{max} is defined by the "limiting conditions" (26) of Section 7 i.e.

$$|q_p| + |q_{\pi^+}| + |q_{\pi^0}| = \sqrt{s} \tag{26"}$$

obtained imposing $r_i = 0$, $m_i = 0$.

The "transversality" of the interaction is then indicated by the quantity

$$Z = \frac{R}{R_{max}} \tag{54}$$

Z is the second independent variable

In reaction (38) the production of resonances generates "correlations". In the particular case, one can have

$\pi^+ p \rightarrow \Delta^{++}(1236)\,\pi^0$ a) $(\Delta^{++} \rightarrow p\pi^+)$

$\rightarrow p\,\rho^+(750)$ b) $(\rho^+ \rightarrow \pi^+\pi^0)$

$\rightarrow \Delta^+(1236)\,\pi^+$ c) $(\Delta^+ \rightarrow p\pi^0)$

Resonances are best studied in a Dalitz plot (Caption 7 of Ref.[13a]), i.e.

$$T_p + T_{\pi^+} + T_{\pi^0} = m_p^2 + m^2 + m^2 \qquad (56)$$

where T_i is the kinetic energy of particle i. For any "quasi two-body" reaction proper T_j is approximately constant.

For reaction (55a) $T_{\pi^0} \approx$ const since $M_{p\pi^+} \sim$ const,
For reaction (55b) $T_p \approx$ const since $M_{\pi^+\pi^0} \approx$ const,
For reaction (55c) $T_{\pi^+} \approx$ const since $M_{p\pi^0} \approx$ const.

Figure 36 explicitly shows a Dalitz plot which reproduces, because of Eq.(56) and $T_i \geq 0$, the properties of a triangle.

Then,

Two kinetic energies are the additional independent variables

We have reached finally our target.

Figure 37 shows that $Z \approx 1$ and again the transversality is not essential. Monte-Carlo-generated events using a statistical phase space (Fig. 37a) have $\langle Z \rangle \approx 0.5$; real events at 3.92 GeV/c (Fig. 37b) have $\langle Z \rangle \approx 0.9$.

If we then neglect the variable Z, an almost complete description of a three-body reaction is contained in a prism plot in which the Dalitz plot is analysed as a function of ω (Fig. 38).

TABLE III. VARIABLES

n = 3	3n - 5 = 4 variables
	2 kinetic energies
	ω angle
	$Z = R/R_{max}$

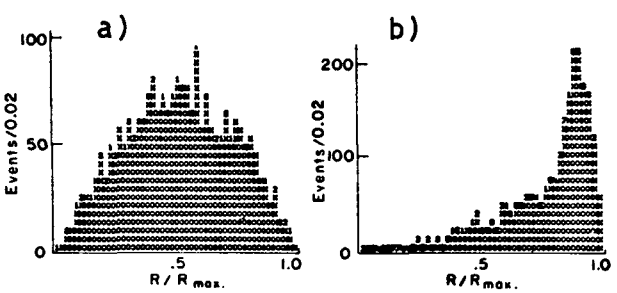


FIG.37. a) Distribution of the variable Z defined by Eq.(54) obtained by Monte-Carlo events of reaction (30) using statistical phase space;
b) Distribution of Z for real events of reaction (38) at 3.92 GeV/c.

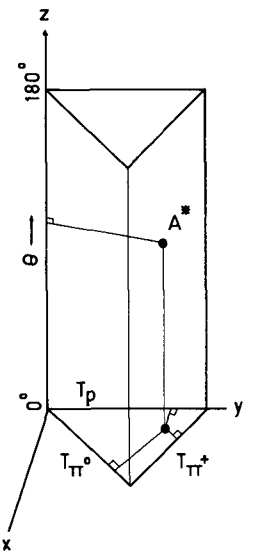


FIG. 38. The prism plot.

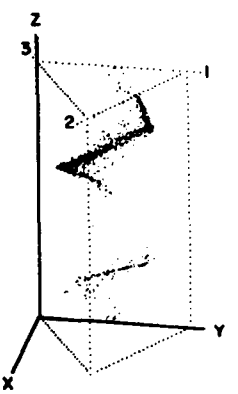


FIG. 39. Experimental prism plot for reaction (38) at 3.92 GeV/c obtained with a graphic display.

It is quite impressive how the experimental results look like in such a plot (Fig. 39). They lie on a kind of spiral around the central axis parallel to ω and the "quasi-two-body" reactions are clearly separated. The "resonances bands" (shaded areas in Fig. 36) are "stretched" along the ω-axis and the "overlapping regions" (hatched areas in Fig. 36) are reduced.

9. CONCLUDING REMARKS

The use of the computer in the analysis of high-energy collisions has become essential in order to obtain significant information from experimental data. In multi-particle production processes the experiment is far from being finished with the collection of the crude experimental data.

Heavy manipulation and intensive use of the computer is needed in any case:

a) for "data presentation";
b) for elaboration of theoretical models;
c) for manipulation of experimental data to measure significant quantities.

High-energy physicists must become computer experts not only in order to rapidly collect experimental data, but also to properly handle them. Any kind of tool (of hardware or software type) suitable to make easier the communication between man and computer (handy languages, graphic display, on-line conversational devices, etc.) will become more and more relevant in the future.

REFERENCES

[1] RATTI, S., "Inelastic collisions of hadrons", Subnuclear Phenomena (ZICHICHI, A., Ed.), Academic Press (1970) 541.

[2] BERTOCCHI, L., FERRARI, E., "High-energy strong interactions of elementary particles", High Energy Physics (BURHOP, E.H.S., Ed.) 2, Academic Press (1967) 72.

[3] KIDD, J.M., "Interactions at very high energies", ibid., 265.

[4] See, e.g. HEARN, A.C., DRELL, S., "Peripheral Processes", High Energy Physics, (BURHOP, E.H.S., Ed.) Academic Press (1967) 219.

[5] COLETTI, S., KIDD, J., MANDELLI, L., PELOSI, V., RUSSO, V., TALLONE, L., ZAMPIERI, E., CASO, C., CONTE, F., GROSSO, C., TOMASINI, G., Nuovo Cim. 49A (1967) 479.

[6] BODINI, L., CASÈ, L., KIDD, J., MANDELLI, L., RATTI, S., RUSSO, V., TALLONE, L., CASO, C., CONTE, F., DAMERI, M., TOMASINI, G., Nuovo Cim. 58A (1968) 475.

[7] See, e.g. a) COLLINS, P.D.B., SQUIRES, E.J., "Regge poles in particle physics", Springer Tracts in Modern Physics No.45, Springer (1968).
b) OMNES, R., FROISSART, H., Mandelstam theory and Regge Poles – Benjamin Press (1965).

[8] See, e.g. RATTI, S., in High Energy Collisions of Hadrons (Proc. Top. Conf., 1968, Report CERN 68-7) 1 (1968) 611.

[9] CASO, C., CONTE, F., TOMASINI, G., BASSLER, E., CORDS, D., DREWS, G., VON HANDEL, P., NAGEL, H., SCHILLING, P., CASÈ, L., MANDELLI, L., RATTI, S., VEGNI, G., DARONIAN, P., DAUDIN, A., GANDOIS, B., MOSCA, L., Nuovo Cim. 66A (1970) 11.

[10] CHAN, H.H., ŁOSKIEWICZ, J., ALLISON, W.W.H., Nuovo Cim. 57A (1968) 93.

[11] BARATOV, M., LE GUYADER, J., SENÈ, M., GINESTET, J., MANESSE, D., TRAN, H.A., VIGNAUD, D., BARTKE, J., Nucl. Physics B20, (1970) 573.

[12] BÖCKMANN, K., NELLEN, B., PAUL, E., WAGINI, B., BORECKA, I., DIAZ, J., HEEREN, U., LIEBERMEISTER, U., LOHRMANN, E., RAUBOLD, E., SÖDING, P., WOLFF, S., KIDD, J.M., MANDELLI, L., MOSCA, L., PELOSI, V., RATTI, S., TALLONE, L., Nuovo Cim. 42 (1966) 954.

[13] See, e.g.
a) KÄLLÉN, G., Elementary Particle Physics, Addison-Wesley (1964), Chapter 1;
b) WILLIAMS, W.S.C., "An Introduction to Elementary Particles", 2nd Ed., Academic Press, (1971) Chapter 1.

[14] FEYNMAN, R.P., Physical Review 23 (1969) 1415.

[15] VAN HOVE, L., Physics Letts 28B (1968) 429.
VAN HOVE, L., Nucl. Physics 9B (1969) 331.

[16] KITTEL, W., RATTI, S., VAN HOVE, L., Nucl. Physics 30B (1970) 333.

[17] BARTSCH, J. et al., Nucl. Physics B19 (1970) 381.

[18] Durham-Genova-Hamburg-Milano-Saclay Collaboration – "Observation of the B-meson in the reaction $\pi^+ p \to \omega^0 \pi^+ p$ at 11.7 GeV/c", presented by RATTI, S., Int. Conf. Meson Resonances and Related Electromagnetic Phenomena (Proc. Conf. Bologna, April, 1971) (to be published).

[19] TOMASINI, G., TREVISAN, U., BORZATTA, P., PELOSI, V., RATTI, S., TALLONE-LOMBARDI, L., GOUSSU, O., VINCENT, M.A., Longitudinal configurations of three- and four-body reactions containing strange particles produced in $\pi^- p$ collisions at 11.2 GeV/c", University of Milano preprint IFUM 121/AE (1971) (to be submitted to Nuovo Cim.).

[20] a) TOMASINI, G., BASSLER, E., SAHINI, M., SCHILLING, P., BORZATTA, P., CECCHET, G., LIOTTA, L., RATTI, S., DAUDIN, A., FACCINI, M.L., JABIOL, M.A., Longitudinal phase space analysis of the reaction $\pi^+ p \to \pi^+ \pi^+ \pi^- p$ and comparison of the reactions $\pi^\pm p \to \pi^\pm \pi^+ \pi^- p$ at the same energy, University of Milan preprint IFUM 120/AE 1971 (to appear in Nuovo Cim.)

b) The comparison at 16 GeV/c is a personal contribution of S. Ratti to the Amsterdam Conference on Elementary Particles (1971), using unpublished data privately communicated by G.J. Bossen.

[21] KITTEL, W., private communication.

[22] DAO, F.T., HODOUS, M.F., PLESS, I.A., SINGER, R.A., Phys. Rev. Letts 27 (1971) 1481.

OPTICAL-MODEL AND COUPLED-CHANNEL CALCULATIONS IN NUCLEAR PHYSICS

J. RAYNAL
Service de physique théorique,
Centre d'études nucléaires de Saclay,
91-Gif-sur-Yvette
France

Abstract

OPTICAL-MODEL AND COUPLED-CHANNEL CALCULATIONS IN NUCLEAR PHYSICS.

Optical-model and coupled-channel calculations basically amount to solving a second-order differential equation without first derivatives or a system of such equations, although the most general direct reaction mechanism leads to a set of coupled integro-differential equations. The main numerical methods used for this problem are presented. They are the Numerov method, the modified Numerov method better adapted to coupled equations than the preceding one, the Störmer and the De Vogelaere methods. Simple expressions for the errors are derived; these methods are discussed from the point of view of stability in the asymptotic region when the step size is increased; practical examples of the effects of the step are given. However, all these coupled-channel calculations are time-consuming. The sequential iteration for coupled equations (ECIS) leads to a drastic reduction in computational time in many cases. This method can be used for sets of integro-differential equations but has been only applied to sets of second-order differential equations including first derivatives. The first iteration is related to the DWBA, but gives results in problems beyond the scope of DWBA. The ECIS method can be used in an integral form (Green function) or a differential form as in the code ECIS70. The automatic search methods are also presented. The optical-model code MAGALI, for spin 0, 1/2 and 1 particles, is described, and also the attempts to introduce automatic search into a coupled-channel code: ECIS71.

The optical-model calculations, and their generalizations, the coupled-channel calculations as well as the so-called distorted-wave Born approximation (DWBA), are almost the only kind of computation used by the physicist interested in nuclear reaction cross-sections and polarizations.

The optical model for elastic scattering is such a simple problem [1] that almost any method can give accurate results in a reasonable time with the high-speed computers available by now. For coupled-channel calculations [2], the question of numerical methods is more crucial because computational time is a limit to the number of channels which can be coupled. So, we shall first present the problem of coupled-channel scattering and then the more useful numerical methods (but, chiefly, the one which I use, of which I have wider experience and which has been applied to very different problems).

The coupled-channel concept, as numerical solution to a set of coupled differential equations, can be criticized as a waste of time in many applications. I have developed an iteration procedure [3] some years ago; it provides the same results in less time. This "sequential iteration method for coupled equations", called ECIS (équations couplées en itération séquentielle), is an approximation between DWBA and coupled-channel computations. The difference between its first iteration and

DWBA is very small (only due to the "sequential" use of the equations). The first iteration can provide good results for multistep excitations; it needs more time than DWBA. The convergence of the iterations is usually fast. Furthermore, the ECIS method can deal with sets of coupled integro-differential equations, although it has only been applied to coupled second-order differential equations including first derivatives [4], for which the usual high-speed integration techniques cannot be applied.

The optical model depends upon some parameters known to, say, 10%. To do a computation with some parameters, to compare with experimental data, to change some parameters and to compare again is a long and boring work. Since a long time, there have existed automatic search programs for optical-model calculations; now a generation of automatic search programs for coupled-channel fits is approaching. The automatic search problem consists of two parts: management of information on the fit and computation of this information. We shall consider the second point in greater detail both in optical-model and coupled-channel calculations, because it is closely related to the problem without automatic search.

If we summarize the problem to the solution of a set of coupled second-order differential equations, the application of these methods is not limited to nuclear reactions involving two particles only. In fact, these methods were originated by astronomers, but the details of their application are quite different because we do not need such an accuracy. They are used in some approaches to the three-body problem, for bound-state calculations [5] and for scattering [6]. I even had the opportunity to give some hints to a physicist interested by electromagnetic optics [7], which, I think, were very useful to him.

1. THE PHYSICAL PROBLEM

We shall first present the optical model for spin-1/2 particles. Then, we shall consider the rotational model, which is its straightforward generalization to some inelastic scattering problems, as an example of coupled equations.

1.1. The optical model

The elastic scattering of a particle by a medium or heavy nucleus of mass A and charge Z, for an incident energy E larger than 10 MeV, is described by a solution of

$$(T - E + V_{opt})\Psi = 0 \qquad (1)$$

where T is the kinetic energy. The usual optical model is

$$- V_1 f(r, a_1, R_1) - i V_2 f(r, a_2, R_2) - 4i a_3 V_3 \frac{d}{dr} f(r, a_3, R_3)$$

where f is the Saxon-Woods form factor:

$$f(r, a, R) = (1 + \exp((r-R')/a))^{-1} \quad \text{with} \quad R' = R A^{1/3}$$

For charged particles, the optical potential includes the Coulomb potential which is usually calculated from a uniformly charged sphere of radius $R'_c = R_c A^{1/3}$:

$$V_{Coul} = \frac{Z e^2}{R'_c}\left(\frac{3}{2} - \frac{r^2}{2R'^2_c}\right) \quad \text{for} \quad r < R'_c$$

$$= Z e^2 \frac{1}{r} \quad \text{for} \quad r > R'_c$$

When the spin of the incident particle is not zero, the optical potential includes also a spin-orbit potential

$$\left(\frac{\hbar}{2M_\pi}\right)^2 \left\{V_4 \frac{1}{r}\frac{d}{dr} f(r, a_4, R_4) + i V_5 \frac{1}{r}\frac{d}{dr} f(r, a_5, R_5)\right\} (\vec{\ell}\cdot\vec{\sigma})$$

where M_π is the pion rest mass.

The solution of Eq.(1) must be a plane wave plus an outgoing wave. Its asymptotic form must be given by

$$\Psi_\sigma = \exp\left(i\vec{k}\cdot\vec{r} + i\eta \ln(kr - \vec{k}\cdot\vec{r})\right)|\sigma\rangle + \sum_{\sigma'} \frac{1}{r} f_{\sigma\sigma'}(\theta) \cdot$$
$$\times \exp\left(ikr - i\eta \ln(2kr)\right)|\sigma'\rangle \qquad (2)$$

where σ is the component of the spin at infinity, k the momentum and η the Coulomb parameter. The angular-momentum expansion of this solution is

$$\Psi_\sigma = 4\pi \frac{1}{kr} \sum i^\ell \exp(i\sigma_\ell) f_{\ell j}(r) \langle \ell s \mu\, \sigma | jm\rangle \langle \ell s \mu' \sigma' | jm\rangle$$
$$Y_\ell^{\mu\,*}(\hat{k})\, Y_\ell^{\mu'}(\hat{r})|\sigma'\rangle \qquad (3)$$

where σ_ℓ is the Coulomb phase-shift.

Introducing this expansion into Eq.(1), we obtain for $r > R'_c$:

$$\left\{\frac{d^2}{dr^2} - \frac{\ell(\ell+1)}{r^2} + k^2 - \frac{2\mu}{\hbar^2}\left(V^{opt}_{central} + \gamma V^{opt}_{sp.\,or.}\right) - \frac{2\eta k}{r}\right\} f_{\ell j}(r) = 0 \qquad (4)$$

where μ is the reduced mass and γ the eigenvalue of $(\vec{\ell}\cdot\vec{\sigma})$. For $r < R'_c$, the Coulomb potential is different. The solution of Eq.(4) must vanish at the origin; beyond the range of the nuclear potentials, it can be expressed by the two standard solutions of the equation: the regular Coulomb function, $F_\ell(\eta, kr)$, and the irregular one, $G_\ell(\eta, kr)$. The linear combination corresponding to the asymptotic form of Eq.(2) is

$$f_{\ell j}(r) = F_\ell(\eta, kr) + C^j_\ell \left(G_\ell(\eta, kr) + i F_\ell(\eta, kr)\right) \qquad (5)$$

Introducing this asymptotic form into Eq.(3) and identifying it to Eq.(2), we obtain

$$f_{\sigma\sigma'}(\theta) = \delta_{\sigma\sigma'} f_c(\theta) + \frac{4\pi}{k} \sum_{\ell} \exp(2i\sigma_\ell) C_\ell^j$$

$$\times \langle \ell s \mu\sigma | jm \rangle \langle \ell s \mu'\sigma' | jm \rangle Y_\ell^{\mu^*}(\hat{k}) Y_\ell^{\mu'}(\hat{r})$$

where $f_c(\theta)$ is the Coulomb amplitude

$$f_c(\theta) = -\frac{\eta}{2k \sin^2 \frac{1}{2}\theta} \exp\left(-i\eta \ln(\sin^2 \tfrac{1}{2}\theta) + 2i\sigma_0\right) \qquad (6)$$

For a spin-1/2 particle, there are two different amplitudes:

$$A(\theta) = f_{\frac{1}{2}\frac{1}{2}} = f_{-\frac{1}{2}-\frac{1}{2}} = f_c(\theta) + \frac{1}{k} \sum_{\ell} \exp(2i\sigma_\ell)$$

$$\times \left\{(\ell+1) C_\ell^{\ell+\frac{1}{2}} + \ell C_\ell^{\ell-\frac{1}{2}}\right\} P_\ell(\cos\theta)$$

$$B(\theta) = f_{\frac{1}{2}-\frac{1}{2}} = f_{-\frac{1}{2}\frac{1}{2}} = \frac{1}{k} \sum_{\ell} \exp(2i\sigma_\ell)$$

$$\times \left\{C_\ell^{\ell+\frac{1}{2}} - C_\ell^{\ell-\frac{1}{2}}\right\} P_\ell^1(\cos\theta)$$

The cross-section is given by

$$\sigma(\theta) = |A(\theta)|^2 + |B(\theta)|^2$$

and the polarization

$$P(\theta) = \frac{2 \operatorname{Im} A^*(\theta) B(\theta)}{\sigma(\theta)}$$

1.2. The rotational model

The interaction between the particle and the target is some potential $V(\vec{r}, \hat{r}')$, where $\hat{r}'$ is the intrinsic axis of the nucleus. This potential is parametrized by quadrupole and hexadecupole deformations β_2 and β_4, using a radius $R(\theta)$

$$R(\theta) = R_0\left(1 + \beta_2 Y_2^0(\theta) + \beta_4 Y_4^0(\theta)\right)$$

where θ is the angle between $\vec{r}$ and $\hat{r}'$. This radius is used instead of the R_i in the usual expressions of the optical potential. The potential can be expanded into multipoles

$$V(r, \hat{r}') = 4\pi \sum_{\lambda,\mu} V_\lambda(r)\, Y_\lambda^\mu(\hat{r})\, Y_\lambda^{\mu^*}(\hat{r}')$$

where there are only even values of λ.

For an initial state of the target $|\Psi_{I_i, M_i}\rangle$ and a final one, $|\Psi_{I_f, M_f}\rangle$, members of a rotational band starting by a 0^+, the potential between an ingoing wave $|\Psi_{I_i, M_i}\rangle\, |\ell_i j_i m_i\rangle$ coupled to J, M and an outgoing wave $|\Psi_{I_f, M_f}\rangle\, |\ell_f j_f m_f\rangle$ also coupled to J and M, is

$$\sum_\lambda V_\lambda(r)(-)^{J+\lambda-\frac{1}{2}}(2\lambda+1)\sqrt{(2I_i+1)(2I_f+1)(2j_i+1)(2j_f+1)}$$

$$\times \begin{pmatrix} I_i & I_f & \lambda \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} j_f & \lambda & j_i \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix} \begin{Bmatrix} I_i & I_f & \lambda \\ j_f & j_i & J \end{Bmatrix}$$

when the spin of the incident particle is $\frac{1}{2}$.

For a $0^+ \rightarrow 2^+$ excitation and for an ingoing wave of given ℓ_i, j_i, the total spin of the system is j_i and its parity $\pi = (-)^{\ell_i}$. For a sufficiently large J and a given parity there are five outgoing waves ℓ_f, j_f; j_f ranges from $j_i - 2$ to $j_i + 2$ and ℓ_f is the value $j_f + \frac{1}{2}$ or $j_f - \frac{1}{2}$ which has parity π. We obtain a set of six coupled equations

$$y_i'' + \sum_j V_{ij}\, y_j = E_i y_i \qquad (7)$$

where

$$V_{ij} = \sum_\lambda G_{ij}^\lambda\, V_\lambda(r) \qquad (8)$$

This sum includes the optical potential for $\lambda = 0$, which is diagonal. The energies E_i are the incident energy or the outgoing energy of the particle.

The multipole expansion of the spin-orbit potential is not straightforward. Some years ago, Blair and Sherif [8] obtained excellent results, starting from

$$\vec{\nabla}\left\{V(r)\right\} \times \frac{\vec{\nabla}}{i} \cdot \vec{\sigma}$$

which leads to [4]

$$\frac{1}{r}\frac{d}{dr} V_\lambda(r)\,\gamma_i + (\gamma_i - \gamma_f)\frac{1}{r}\frac{d}{dr} + \frac{V_\lambda(r)}{2r^2}\left(\lambda(\lambda+1) - (\gamma_f - \gamma_i)(\gamma_f - \gamma_i - 1)\right) \qquad (9)$$

multiplied by the same geometrical coefficient as for the central potential. If V(r) is isotropic, this interaction gives back the usual optical spin-orbit potential. It introduces first derivatives into the coupled equations.

1.3. Number of coupled equations

The number of coupled equations is larger than the number of channels taken into account. Let us consider n channels. In the i-th channel, the spin of the particle is s_i, the spin of the target I_i and the product of the intrinsic parities of the particle and the target ϵ_i. A count of the possible quantum numbers gives, for a total spin J sufficiently large and a parity η,

$$N_i = \frac{1}{2}\left[(2I_i+1)(2s_i+1) + \epsilon_i'(-)^{J+\eta}\right]$$

where $\epsilon'_i = 0$ if I_i or s_i is half-integer and $\epsilon'_i = \epsilon_i(-)^{I_i+s_i}$ if I_i and s_i are integers.

The sum of the N_i can be large: for the scattering of protons on a $0^+ - 2^+ - 4^+ - 6^+$ rotational band, there are 28 equations. For the scattering of α-particles on the same levels, there are 16 equations for parity $\eta = (-)^J$ and only 12 for parity $\eta = (-)^{J+1}$; but the sets of 12 equations have not to be solved because none of them apply to the ground state.

The number of coupled equations decreases when the total J is smaller than the largest value of $I_i + s_i$ because some quantum numbers are forbidden.

1.4. Normalization of the solution and cross-section

The solution must have the asymptotic form of Eq.(2) as long as the initial channel is concerned and pure outgoing waves for all the other channels. The asymptotic value of the radial solution must be

$$\begin{aligned} y_i &= F_{\ell_i} + C_i^i\,(G_{\ell_i} + iF_{\ell_i}) \\ y_f &= C_f^i\,(G_{\ell_f} + iF_{\ell_f}) \end{aligned} \qquad (10)$$

By identification we obtain

$$\begin{aligned} f_{\mu_f\sigma_f\mu_i\sigma_i}(\theta) = \frac{4\pi}{k_i}\sum i^{\ell_i - \ell_f}\exp(i\sigma_{\ell_i} + i\sigma_{\ell_f}) \\ \times \langle \ell_i s\,\nu_i\,\sigma_i | j_i m_i\rangle\langle j_i I_i m_i\mu_i | JM\rangle\langle \ell_f\, s\,\nu_f\,\sigma_f | j_f m_f\rangle \\ \times \langle j_f I_f m_f\mu_f | JM\rangle\; Y_{\ell_i}^{\nu_i *}(\hat{k}_i)\; Y_{\ell_f}^{\nu_f}(\hat{k}_f) \end{aligned} \qquad (11)$$

where σ and μ are the projection of the spins of the particle and the target, and i and f refer to the initial and final states. These amplitudes are simplified in the helicity formalism [9] in which the spins of the particles are projected on their momentum. The transformation is obtained by choosing an axis of quantification along k_i for the initial state and along k_f for the final state; the helicity of the target is then the opposite of the projection of its spin. After summation of the rotation matrix elements for the spins and the spherical harmonics of Eq.(11) we obtain

$$f^{\text{hel.}}_{\mu_f \sigma_f \mu_i \sigma_i}(\theta) = \sum_J f^{(J)\text{hel.}}_{\mu_f \sigma_f \mu_i \sigma_i} R^{(J)}_{\sigma_f - \mu_f,\ \sigma_i - \mu_i}(\theta)$$

with

$$\begin{aligned} f^{(J)\text{hel.}}_{\mu_f \sigma_f \mu_i \sigma_i} &= \frac{1}{k_i} \sum_i i^{\ell_i - \ell_f} \exp\left(i\sigma_{\ell_i} + i\sigma_{\ell_f}\right) C^i_f \\ &\times \langle \ell_i\, s\, 0\, \sigma_i | j_i \sigma_i \rangle \langle j_i I_i \sigma_i - \mu_i | JM_i \rangle \langle \ell_f\, s\, 0\, \sigma_f | j_f \sigma_f \rangle \\ &\times \langle j_f I_f \sigma_f - \mu_f | JM_f \rangle \end{aligned} \qquad (12)$$

The cross-section is given by

$$\frac{d\sigma(\theta)}{d\Omega} = \frac{1}{(2s+1)(2I_i+1)} \frac{k_f}{k_i} \sum \left| f_{\mu_f \sigma_f \mu_i \sigma_i}(\theta) \right|^2$$

Polarization and asymmetries are simple expressions of the amplitudes.

As is shown by Eq.(12), there is an intermediate step between the computation of the coefficients C and the computation of the amplitudes: the coefficients of simple angular functions.

2. THE INTEGRATION METHODS

The integration methods can be separated into two groups: the ones which need only two values at some points to compute the function at the next point and the other ones which need more information on the function at the preceding points. Each group can be divided into two subgroups: they use or do not use the first derivative.

In the first group, with first derivative, there are the Runge-Kutta methods, which are not very appropriate to our problem. In the same group, without first derivative, there are the Cowell method, its simplification, the Numerov method, and what we have called the "modified Numerov" method. We shall mainly be concerned with these last methods. All methods of this group have been studied quite intensively for the optical model [1].

In the second group, with first derivative, there is the De Vogelaere method used at Berkeley for coupled channels and, without first derivative, the Störmer method used in Tamura's code.

With any integration method, one can consider that there are three successive problems: the starting values, the integration itself and the matching [1, 3].

2.1. The starting values

Let us consider a system of N second-order differential equations (without first derivative). For a physical problem, the diagonal couplings are finite at the origin, and the coupling between angular momenta ℓ_i and ℓ_j comes from a multipole λ, at least equal to $|\ell_i - \ell_j|$ and there would be a singular point in the interaction if it does not decrease as r^λ. Around the origin, the set of equations can be written as

$$\left\{ \frac{d^2}{dr^2} - \frac{\ell_i (\ell_i + 1)}{r^2} \right\} f_i(r) + \sum_j a_{ij} r^{|\ell_i - \ell_j|} f_j(r) = 0$$

Out of the 2N solutions, there are only N vanishing at the origin. They can be expanded into Taylor series and chosen as

$$f_i(r) = r^{\ell_i + 1} + b_i r^{\ell_i + 3} + \ldots$$

$$f_j(r) = b_j r^{\ell_i + 3} + \ldots \quad \text{if } \ell_j < \ell_i$$

$$= b_j r^{\ell_j + 3} + \ldots \quad \text{if } \ell_j > \ell_i$$

The differential equations provide recurrence relations among the b's.

If the integration method needs the first derivative, these Taylor series can be used to obtain the starting values at r = h; however, the starting values

$$f_j^i(h) = A \delta_{ij} \qquad f'^i_j(h) = A (\ell_i + 1) \delta_{ij}$$

introduce errors of the order h^2. Here, A is an arbitrary constant, small in general because the functions increase during integration and the superscripts define independent solutions.

If no first derivative is needed, the starting values can be

$$f_j^i(0) = 0 \qquad f_j^i(h) = A \delta_{ij}$$

These starting values can also be used for integro-differential equations.

2.2. The integration

Let us consider a single equation

$$f''(r) = V(r) f(r)$$

and give short indications for coupled or inhomogeneous equations.

The Euler method is the simplest; the second difference of the function is identified with its second derivative. The function at point r+h is given by

$$f(r+h) = 2\,f(r) - f(r-h) + h^2 f''(r) \tag{13}$$

when the truncation error $\Delta = h^4 f^{(IV)}(r)/12$ is neglected. This method, of which the final error is of the order of h^2, has been often used to solve coupled equations.

2.2.1. Cowell, Numerov and modified Numerov methods

The Cowell [10] method is based on the following relation between a function and its second derivative at three equidistant points

$$\xi(r+h) = f(r+h) - h^2 f''(r+h)/12 = 2\,f(r) + 5\,h^2 f''(r)/6 - f(r-h) + h^2 f''(r-h)/12 \tag{14}$$

with the truncation error $\Delta = -h^6 f^{(VI)}(r)/240$. The function at points r-h and r completely defines the solution; the algorithm gives $\xi(r+h)$, from which

$$f(r+h) = \xi(r+h)/\left\{1 - \frac{h^2}{12} V(r+h)\right\}$$

The Numerov method [11] uses exactly the same relation but does not evaluate the function f at any point and considers only the expression ξ. The algorithm is

$$\xi(r+h) = 2\,\xi(r) - \xi(r-h) + u(r)$$
$$u(r) = h^2 f''(r) = \frac{h^2 V(r)}{1 - h^2 V(r)/12}\,\xi(r) \tag{15}$$

Many authors [12] see no difference between these two methods although the second one is faster.

The modified Numerov method [1, 3] replaces the division of the last algorithm by an expansion

$$\xi(r+h) = 2\,\xi(r) - \xi(r-h) + u(r)$$
$$u(r) = h^2 V(r)\,(1 + h^2 V(r)/12)\,\xi(r) \tag{16}$$

The truncation error is now $\Delta = h^6 V^3(r)\,f(r)/144 - h^6 f^{(VI)}(r)/240$. If V(r) is constant (sinusoidal or exponential behaviour) the truncation error is smaller than for the Numerov method. For these two methods, if the function f(r) is needed, it can be obtained by

$$f(r) = \xi(r) + u(r)/12 \tag{17}$$

or by

$$f(r) = (\xi(r+h) + 10\,\xi(r) + \xi(r-h))/12 \tag{18}$$

The application of the modified Numerov method to coupled equations is straightforward. The matrix of the N solutions at the point r+h can be obtained from the matrix of the solutions at the points r and r-h by

$$\xi_i^k(r+h) = 2\,\xi_i^k(r) - \xi_i^k(r-h) + u_i^k(r)$$

$$u_i^k(r) = \sum_j \mathscr{V}_{ij}(r)\,\xi_j^k(r) \tag{19}$$

$$\mathscr{V}_{ij}(r) = h^2 V_{ij}(r) + h^4 \sum_k V_{ik}(r)\,V_{kj}(r)$$

The function is

$$f_i^k(r) = \xi_i^k(r) + u_i^k(r)/12$$

The computation of $\mathscr{V}$ is the product of two N× N matrices, which can be reduced by a factor two, taking into account the symmetry of V. The computation of u needs also a product of two N× N matrices, which cannot be reduced. Each of these one and half multiplications of matrices requires N^3 operations and makes the computation slow.

The Numerov method can also be used. A division of matrices is necessary at each step. The number of operations is the same as for the multiplication at the limit of large N but programming is more difficult.

For an inhomogeneous equation [13]:

$$f''(r) = V(r)\,f(r) + W(r)$$

the definition of $\xi(r)$ can be changed into

$$\xi(r) = (1 - h^2 V(r)/12)\,f(r)$$

and the algorithm becomes

$$\xi(r+h) = 2\,\xi(r) - \xi(r-h) + u(r) + \{W(r+h) + 10\,W(r) + W(r-h)\}/12 \tag{20}$$

with

$$u(r) = \frac{h^2 V(r)}{1 - h^2 V(r)/12}\,\xi(r) \qquad \text{for the Numerov method}$$

$$= (h^2 V(r) + h^4 V^2(r)/12)\,\xi(r) \qquad \text{for the modified Numerov method.}$$

2.2.2. Störmer and De Vogelaere methods

The Störmer [14] method uses the following algorithm:

$$f(r+h) = 2\,f(r) - f(r-h) + h^2\,(299\,f''(r) - 176\,f''(r-h) + 194\,f''(r-2h) - 96\,f''(r-3h) + 19\,f''(r-4h))/240 \qquad (21)$$

with a truncation error $\Delta = 3\,h^7 f^{(VII)}(r)/40$. This method is not self-starting as the preceding ones; some steps of the Euler method with smaller size are usually used around the origin. For coupled equations, only one matrix multiplication is used at each step. This method is used by Tamura's code Jupitor.

The De Vogelaere [15] method uses the function and its derivative at one point plus an approximate value of the function in the middle of the last step. The algorithms are

$$\begin{aligned} f(r+\tfrac{1}{2}h) &= f(r) + \tfrac{1}{2}h\,f'(r) + h^2(4\,f''(r) - f''(r-\tfrac{1}{2}h))/24 \\ f(r+h) &= f(r) + h\,f'(r) + h^2(2\,f''(r) + 4\,f''(r+\tfrac{1}{2}))/12 \\ f'(r+h) &= f'(r) + h\,(f''(r) + 4\,f''(r+\tfrac{1}{2}h) + f''(r+h))/6 \end{aligned} \qquad (22)$$

with truncation errors $\Delta = h^5 f^{(V)}(r)/120$ and $\Delta' = -13\,h^5 f^{(VI)}(r)/1440 + h^5 V(r)\,f^{(IV)}(r)/198$. This method is used by Glendenning's code. It needs two multiplications of matrices at each step and the evaluation of the potentials in the middle of the steps.

2.3. Matching

The integration is performed up to some point R for which the potentials vanish. There, the set of coupled equations is a set of single equations of which the solutions are known: in the general case, they are the Coulomb functions. The matching is usually done with the functions and their first derivatives. However, if the first derivative is not used in the integration, the matching can be done with the functions at the points R-h and R+h.

Let us consider the Numerov method. There are subroutines for the Coulomb functions $F_\ell(\eta, kR)$, $G_\ell(\eta, kr)$ and their first derivatives. From these values, four steps of Numerov integration with half stepsize, together with an interpolation formula for the first derivative, give us $\mathscr{F}_\ell(\eta; k(R \pm h))$ and $\mathscr{G}_\ell(\eta, k(R \pm h))$, where

$$\mathscr{F}_\ell(\eta, kr) = F_\ell(\eta, kr) - h^2\,F''_\ell(\eta, kr)/12$$

and the same holds for $\mathscr{G}$.

The numerical solution k is a linear combination with coefficients α_j^k of the solutions which have ingoing waves only for the equation j:

$$\xi_i^k(R+h) = \sum_j \alpha_j^k \left\{ \mathscr{F}_i(R+h)\,\delta_{ij} + C_i^j \left[\mathscr{G}_i(R+h) + i\mathscr{F}_i(R+h) \right] \right\}$$

$$\xi_i^k(R-h) = \sum_j \alpha_j^k \left\{ \mathscr{F}_i(R-h)\,\delta_{ij} + C_i^j \left[\mathscr{G}_i(R-h) + i\mathscr{F}_i(R-h) \right] \right\} \qquad (23)$$

Let us consider the matrices

$$A_i^k = \frac{\{\xi_i^k(R+h)\ \mathscr{G}_i(R-h) - \xi_i^k(R-h)\ \mathscr{G}_i(R+h)\}}{\{\mathscr{F}_i(R+h)\ \mathscr{G}_i(R-h) - \mathscr{F}_i(R-h)\ \mathscr{G}_i(R+h)\}}$$

$$B_i^k = \frac{\{\xi_i^k(R+h)\ \mathscr{F}_i(R-h) - \xi_i^k(R-h)\ \mathscr{F}_i(R+h)\}}{\{\mathscr{F}_i(R+h)\ \mathscr{G}_i(R-h) - \mathscr{F}_i(R-h)\ \mathscr{G}_i(R+h)\}}$$

The matching equations become

$$A_i^k = \sum_j \alpha_j^k \{\delta_{ij} + i\,C_i^j\}$$

$$B_i^k = -\sum_j \alpha_j^k C_i^j$$

and then,

$$B_i^k = -\sum_j (A_j^k + i\,B_j^k)\,C_i^j$$

The C coefficients are obtained by the division of two N × N matrices. If this division is considered as the resolution of a set of linear equations of which the right-hand sides are one column of the matrix B, the C_i^j for a fixed i are obtained. If there is only one entrance equation, the C_i^j are needed only for the value of j which corresponds to this equation: the Wronskian relations allow the transposition of the matrix C.

With first derivatives, the computation is similar: the matrices A and B are the Wronskians of the numerical solutions with the irregular and the regular Coulomb functions, respectively.

The matrix (A+iB) can be quasi-singular if the numerical solutions do not remain independent. This difficulty is usually not present, but happened to me once lately. It can be avoided by a Schmidt orthogonalization procedure between the numerical solutions, e.g. at each fortieth integration point.

2.4. Wronskian relations

The solutions of coupled equations satisfy relations similar to the Wronskian for a single equation. With these Wronskian relations, the symmetry properties of the matrix C can be deduced from the symmetries of the potential. Furthermore, the errors due to the integration – truncation, round-off and matching errors – can be evaluated.

Let us consider two of the most general systems of integro-differential equations:

$$-\frac{\hbar^2}{2m_i}\frac{d^2}{dr^2} f_i(r) + \sum_j V_{ij}(r)\, f_j(r) + \sum_j \int K_{ij}(r,r')\, f_j(r')\, dr' + W_i(r) = 0 \tag{24a}$$

$$-\frac{\hbar^2}{2m_i}\frac{d^2}{dr^2} g_i(r) + \sum_j \overline{V}_{ij}(r)\, g_j(r) + \sum_j \int \overline{K}_{ij}(r,r')\, g_j(r')\, dr' + \overline{W}_i(r) = 0 \tag{24b}$$

and evaluate the expression

$$W = \int_0^R \sum_i -\frac{\hbar^2}{2m_i}\left\{f_i(r)\, g_i''(r) - g_i(r)\, f_i''(r)\right\} dr$$

$$= \sum_{i,j} \int_0^R \left(V_{ij}(r) - \overline{V}_{ji}(r)\right) f_j(r)\, g_i(r)\, dr + \sum_{i,j} \int_0^R dr \int_0^R dr' \left(K_{ij}(r,r') - \overline{K}_{ji}(r',r)\right)$$

$$\times f_j(r')\, g_i(r) + \sum_i \int_0^R W_i(r)\, g_i(r)\, dr - \sum_i \int_0^R \overline{W}_i(r)\, f_i(r)\, dr \tag{25}$$

If g(r) and f(r) are regular at the origin, the first member of Eq. (25) is

$$W = \sum_i -\frac{\hbar^2}{2m_i}\left\{f_i(R)\, g_i'(R) - g_i(R)\, f_i'(R)\right\}$$

Between the solutions f^j and g^k which have an ingoing wave only for the equations j and k, respectively, taking into account the Wronskian relation of the Coulomb functions, we obtain

$$W = \hbar^2 \left[\frac{k_k}{2m_k} C(f)^j_k - \frac{k_j}{2m_j} C(g)^k_j\right] \tag{26}$$

When the set of Eq.(24b) is identical to the set (24a), and there are no $W_i(r)$ terms, taking into account the symmetry properties of the coupling, W vanishes and we obtain the time reversal invariance:

$$\frac{k_i}{m_j} C_j^i = \frac{k_i}{m_i} C_i^j$$

This relation exists even with complex potentials: it is the reciprocity invariance. It can be used to simplify the computation of the matrix C.

The relation between the imaginary part of the potential and the cross-section is obtained with Eq.(24b), the complex conjugate of Eq.(24a) (also without $W_i(r)$ terms). For a solution which has an ingoing wave for equation j, we obtain

$$W = -2i\hbar^2 \left[\frac{k_j}{2m_j} \operatorname{Im} C_j^j - \sum_i \frac{k_i}{2m_i} |C_i^j|^2 \right]$$

$$= -2i \sum_{i,k} \int \operatorname{Im} V_{ik} \; f_i^{j*}(r) \; f_k^j(r) - 2i \sum_{ik} \iint \operatorname{Im} K_{ij}(r, r') \; f_i^{j*}(r) \; f_k^j(r') \, dr \, dr'$$

where $\operatorname{Im} C_j^j - \sum_i \frac{k_i m_j}{m_i k_j} |C_i^j|^2$ is the residual cross-section.

Looking at the right-hand side, we can say:

1) When all the coupling potentials are real, the residual cross-section vanishes. This property is the unitarity of the S-matrix. The reaction cross-section of the initial channel $\operatorname{Im} C_j^j - |C_j^j|^2$ is the sum of the cross-sections of the channels taken into account.

2) When only the diagonal local potential has an imaginary part of a definite negative sign, there is a positive residual reaction cross-section: the reaction cross-section in the elastic channel is greater than the total cross-section of all the channels taken into account.

3) With a phenomenological imaginary part in all the couplings, as is usual in the macroscopic models, the residual reaction cross-section can be negative. In general, the effect of the imaginary part of the diagonal potential is so strong that the residual cross-section is larger than the sum of the cross-sections in the channels taken into account.

The same method can be used to compute the errors due to truncation, round-off or a matching point too close to the origin. Let us consider the system of equations

$$f_i''(r) = -\sum_j V_{ij}(r) \; f_j(r) \qquad (27)$$

If the integration method does not use first derivatives, the numerical solution is given by

$$\bar{f}_i(r+h) = 2\,\bar{f}_i(r) - \bar{f}_i(r-h) - h^2\,[\,\bar{f}_i''(r)\,]$$

while the true solution is

$$f_i(r+h) = 2\,f_i(r) - f_i(r-h) - h^2\,[\,f_i''(r)\,] + \Delta_i$$

The bracket $[f''(r)]$ is the second derivative only to the order h^2. So, the numerical function is a solution of

$$\bar{f}_i''(r) = -\sum_j V_{ij}\,\bar{f}_j(r) + \Delta_i/h^2$$

The Wronskian relation yields

$$\hbar^2\left[\frac{k_k}{2m_k}\,C_k^j - \frac{k_j}{2m_j}\,\bar{C}_j^k\right] = \frac{1}{h^2}\int_0^R f_i^j(r)\;\Delta_i^k\,dr$$

Using the symmetry property (26) of the matrix C we obtain

$$\Delta\,C_j^k = \bar{C}_j^k - C_j^k = \frac{-2m_j}{h^2\hbar^2\,k_j}\sum_i \int f_i^j(r)\;\Delta_i^k\,dr$$

which is the expression of the total truncation error. Here, Δ_i^k is an expression of the derivatives of the numerical function. The true function can be replaced by the numerical one to compute the error.

If the round-off error is due to chopping, the error at each step is $N\,\epsilon\,f(r)$ where ϵ is the precision of the machine and N a constant which depends on the programming in question. The same method as above yields the total error, which is the same as adding a constant term $N\,\epsilon/h^2$ to all the diagonal potentials. Note that, when the local truncation error is of the order h^6, the total error is $Ah^4 + B/h^2$ and is minimum for some value of h which depends upon the machine. On an IBM-360, we use single precision for the potentials and double precision for the integration algorithm in order to avoid round-off questions. The round-off is also eliminated by a "summed" method, which uses the difference of the function at two neighbouring points and the value at one point instead of the values at two neighbouring points.

When the integration method uses first derivatives, the total truncation error is different and there is no round-off error [1].

The error coming from a matching point chosen too close to the origin is given by

$$\Delta\,C_n^k = \frac{1}{k_n}\sum_{i,j}\int_R^\infty f_i^n(r)\;V_{ij}(r)\;f_j^k(r)\;dr$$

It is obtained in the same way. The order of magnitude of this error is given by the integral of the potential from the matching point to infinity.

All these errors have a similar form: the matrix element of some radial operator. When introduced into the amplitudes (11), their sum is the matrix element of this radial operator between a coupled-channel solution with an incident wave in the initial channel along $\vec{k}_i$ and another solution with its incident wave in the final channel along $\vec{k}_f$. The effect on the amplitudes depends upon the scattering angle but remains, roughly speaking, of the same order as for the phase shifts.

2.5. Stability and stepsize

There is always a region before the matching point where the potentials are very small, but the numerical integration must be performed anyway. This region is reduced as much as possible but can be large in some cases as Coulomb excitation for which the interaction decreases as $1/r^3$. If the centrifugal potential can be neglected, the solution is a sinus (or an exponential for bound channels). So, it is interesting to study the behaviour of the integration method from the point of view of stability and stepsize when applied to the equation $y'' = \pm y$.

Let us consider the n^{-th} point of the Euler method:

$$f((n+1)h) = 2f(nh) - f((n-1)h) \pm h^2 f(nh)$$

If we replace $f(nh)$ by x^n, we obtain a quadratic equation:

$$x^2 - (2 \pm h^2)\, x + 1 = 0$$

where the roots are

$$x_1, x_2 = 1 + \tfrac{1}{2}h^2 \pm h\sqrt{1+h^2/4}$$

in the exponential case $y'' = y$ and

$$x_1, x_2 = 1 - \tfrac{1}{2}h^2 \pm ih\sqrt{1-h^2/4}$$

in the sinusoidal case $y'' = -y$. If, at the first point, the solution is ax_1+bx_2, at the n^{-th} point, it is $ax_1^n+bx_2^n$. After long numerical integration, only the largest root shows up, for any starting value, because there is always a small admixture between the two roots coming from round-off errors. For this reason, in the exponential case (computation of bound states) the integration must be performed backwards in the exterior region and the solution must be matched to the result of a forward integration in the interior region. In the sinusoidal case, the norm of the complex roots is unity and their argument is approximately $\pm h$: the difference between the argument and the step is the error introduced at each integration point.

The corresponding equation is:

for the Numerov (or the Cowell) method:

$$(1 \mp h^2/12)(1 + x^2) - (2 \pm 5h^2/6)x = 0$$

for the modified Numerov method:

$$(1 + x^2) - (2 \pm h^2 + h^4/12)x = 0$$

for the Störmer method:

$$x^5 - (2 \pm 299h^2/240)x^4 + (1 \pm 176h^2/240)x^3$$
$$\mp (194h^2/240)x^2 \pm (96h^2/240)x \mp 19h^2/240 = 0$$

for the De Vogelaere method:

$$x^3 - (2 \pm 23h^2/24 + h^4/12)x^2 + (1 \pm h^2/12 - h^4/24)x \pm h^2/24 = 0$$

for the function and its first derivative and another third-order equation for the function in the middle of the step.

Table I gives the exact values of the argument and the norm of the complex roots of these five methods in the sinusoidal case, for step sizes 0.1, 0.5 and 1. Except for the Euler method, the Störmer method, which is the best for steps of 0.1, becomes the worse for larger steps.

Figures 1 to 5 show the roots of these equations in the sinusoidal and exponential cases for values of the step h up to 4.

The Euler method (Fig. 1) breaks down for $h = 2$; beyond, there is a real root smaller than -1 which gives an oscillatory instability. The Numerov method (Fig. 2) blows up at $h = \sqrt{6}$ and yields an oscillatory behaviour. The modified Numerov method (Fig. 3) blows up at $h = \sqrt{12}$ and gives an exponential solution. The norm of the complex root is unity for all these three methods.

The Störmer method (Fig. 4) has five roots of which one is always real and two always complex conjugate. For h of the order of 1.1, the real

TABLE I. ARGUMENT AND NORM OF THE COMPLEX ROOT OF THE RECURRENCE RELATION IN THE INTEGRATION OF sin x

Step	0.1	0.5	1.0
Euler	0.1000417136	0.5053605103	1.047197609
Numerov	0.1000000208	0.5000657862	1.002186027
Modified Numerov	0.0999999861	0.4999549343	0.998377755
Störmer	0.0999999942	0.4996016588	0.970518542
De Vogelaere	0.0999999688	0.4999049505	0.997294979
The norms of the Euler, Numerov and Modified Numerov methods are 1.			
Norm of Störmer	1.000000037	1.000405218	1.001849737
Norm of De Vogelaere	0.9999999983	0.9999723707	0.998183180

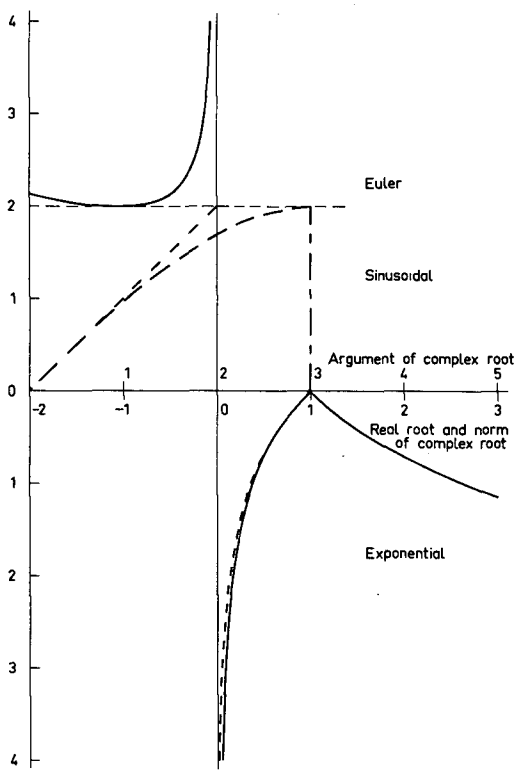


FIG. 1. Asymptotic behaviour of the Euler method. The full curve is the real root (from -2 to +3). The mixed curve is the norm of the complex roots. The long dashed curve is the argument of the complex roots (starting from 0 at the left-hand side of the figure). The short dashed curves are the theoretical values.

root becomes smaller than -1 and gives an oscillatory instability. This real root follows a curve of which a part is the increasing exponential; the decreasing exponential disappears for h = 0.845.

The De Vogelaere method (Fig. 5) has always a real root which becomes dangerous for h = 2.8 in the sinusoidal case. The argument of the complex root is good very far (up to h = 2) but the norm of this root becomes poor above h = 1. The complex root disappears at h = 2.65.

The error of the root related to the increasing exponential with h = 1 is -3.69% for the Euler, 0.20% for the Numerov, -0.12% for the modified Numerov, -0.85% for the Störmer and -0.17% for the De Vogelaere method.

The errors obtained at R = 60 for angular momenta 0, 10, 20 and 40 with different integration methods are shown in Table II. The Störmer method starts with the first points of the De Vogelaere method. This table shows that the Störmer method is best for small steps and low angular momenta but becomes worse when larger steps are used. A repulsive potential reduces the errors as is already indicated in this table by the

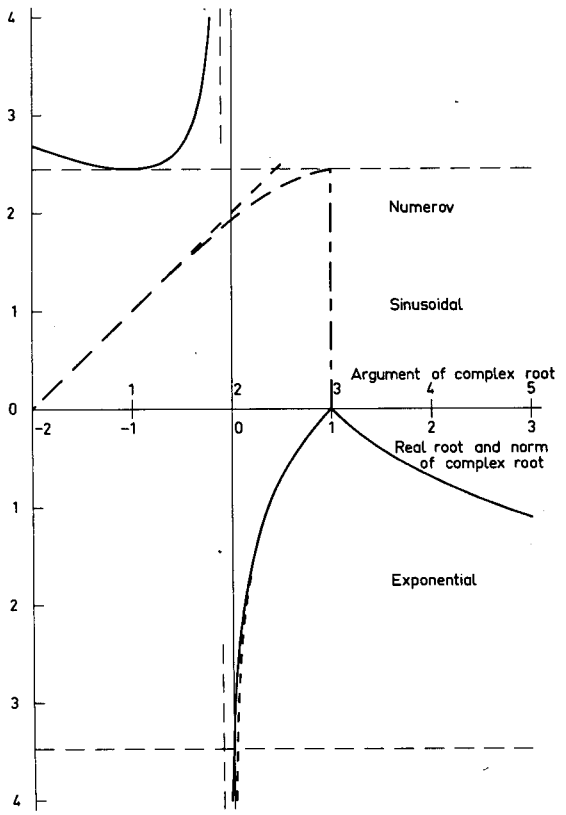


FIG.2. Asymptotic behaviour of the Numerov method. Same notations as for Fig.1.

effect of the centrifugal potential. They are increased by an attractive potential, whereas an imaginary part in the potential reduces them drastically.

This discussion of the stepsize up to so large values is not irrelevant as shown in Figs 6 to 9. The curves of Figs 6 and 7 are the cross-section and the polarization obtained in the rotational model with the program ECIS70, which uses an iteration method, for protons of 24.5 MeV on ^{22}Ne. The real potential is V = 57 MeV, R = 1.05 and a = 0.75, the imaginary potential is a surface potential with W = 6.3 MeV, R = 1.33 and a = 0.55 fermi, the spin-orbit potential is V = 3.95, R = 0.88 and a = 0.31 fermi; the deformation parameters are $\beta_2 = 0.47$ and $\beta_4 = 0.05$. The k values of the channels are, respectively, 1.035, 1.007 and 0.959. The full curve is obtained with h = 0.2 and the dashed one with h = 1. In Fig.6 there are only differences for the elastic cross-section because of the interference between Coulomb and nuclear amplitudes. There are already important differences for the polarization (Fig. 7) because the step is very large

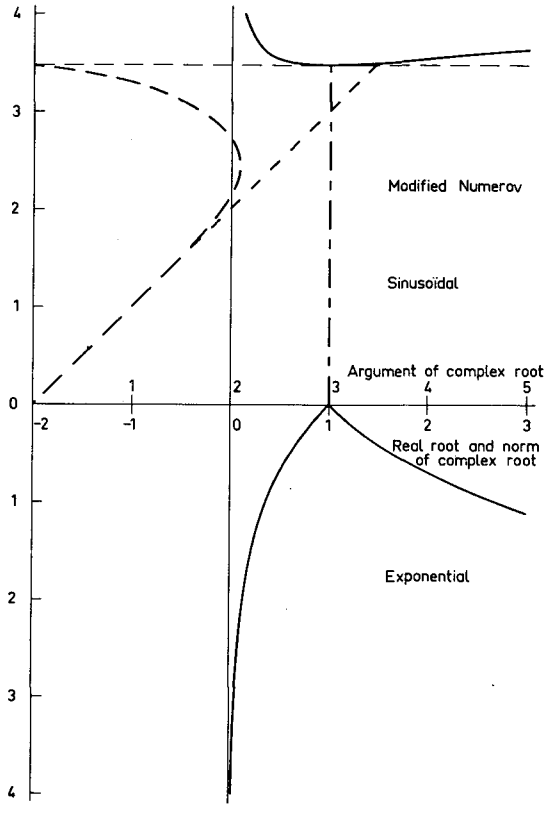


FIG. 3. Asymptotic behaviour of the modified Numerov method. Same notations as for Fig. 1.

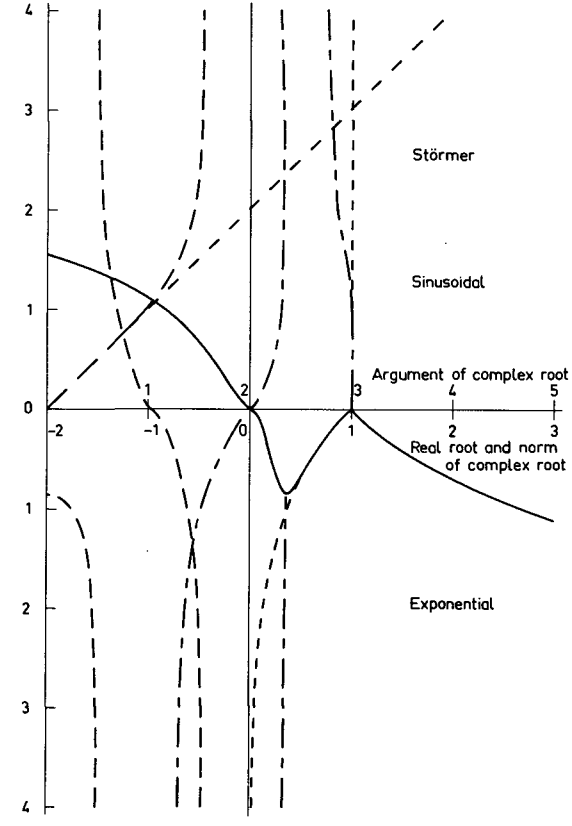


Fig. 4. Asymptotic behaviour of the Störmer method. Same notations as for Fig. 1. The norm of the complex roots is plotted negative in the exponential case to avoid a discontinuity for their argument.

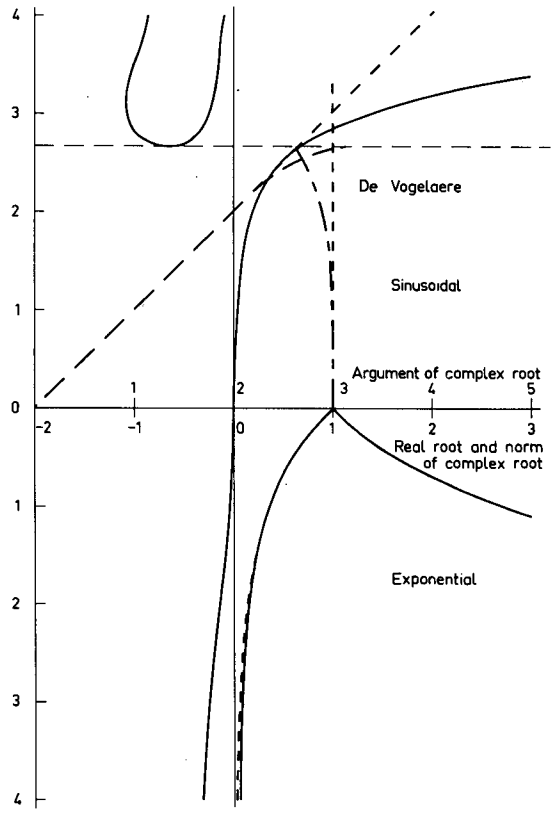


FIG. 5. Asymptotic behaviour of the De Vogelaere method. Same notations as for Fig. 1.

compared to the width of the spin-orbit form factor, even smeared out by the rotational model. The last curve was obtained with h = 1.2, which is a too large step even for the cross-sections.

Figures 8 and 9 are cross-sections obtained with the same program for the scattering of 104 MeV α-particles by ^{58}Ni in the vibrational model. The potential is a volume potential with V = 90 MeV, W = 22 MeV, R = 1.35 fermi and a = 0.55 fermi. The 2^+ state is coupled to the ground state and the 4^+ to the 2^+ by a vibration with β = 0.22. The values of k are, respectively, 4.173, 4.142 and 4.119. The elastic cross-section is shown in Fig. 8. The curves are obtained with steps of 0.05, 0.3, 0.375 and 0.4; differences between steps 0.05 and 0.3 can be seen only after 60°; the step of 0.375 gives more differences in the same region and different values at the minima in the forward angles; with steps of 0.4, the curve is completely changed. Note that the step size of the sinus in the asymptotic region is about $\pi/2$ for h = 0.375 and 1.67 for h = 0.4, and also that these steps are smaller than the diffuseness of the form factors of the potentials. For the

TABLE II. ADMIXTURE OF THE IRREGULAR FUNCTION WHEN INTEGRATING THE RADIAL EQUATION WITHOUT POTENTIAL UP TO kr = 60

		Numerov	Modified Numerov	Störmer	De Vogelaere
h = 0.1	L = 0	-0.124 (-4)	0.831 (-5)	0.337 (-5)	0.186 (-4)
	L = 10	-0.699 (-5)	0.458 (-5)	0.308 (-5)	0.104 (-4)
	L = 20	-0.340 (-5)	0.224 (-5)	0.165 (-5)	0.508 (-5)
	L = 40	-0.357 (-6)	0.253 (-6)	0.362 (-6)	0.549 (-6)
h = 0.2	L = 0	-0.199 (-3)	0.133 (-3)	0.208 (-3)	0.297 (-3)
	L = 10	-0.112 (-3)	0.736 (-4)	0.151 (-3)	0.166 (-3)
	L = 20	-0.544 (-4)	0.359 (-4)	0.743 (-4)	0.808 (-4)
	L = 40	-0.572 (-5)	0.406 (-5)	0.126 (-4)	0.867 (-5)
h = 0.5	L = 0	-0.785 (-2)	0.538 (-2)	0.456 (-1)	0.113 (-1)
	L = 10	-0.440 (-2)	0.296 (-2)	0.269 (-1)	0.629 (-2)
	L = 20	-0.213 (-2)	0.143 (-2)	0.121 (-1)	0.307 (-2)
	L = 40	-0.226 (-3)	0.161 (-3)	0.140 (-2)	0.325 (-3)
h = 1.0	L = 0	-0.130 (+0)	0.969 (-1)	-0.191 (+2)	0.167 (+0)
	L = 10	-0.726 (-1)	0.521 (-1)	0.137 (+1)	0.895 (-1)
	L = 20	-0.345 (-1)	0.246 (-1)	0.466 (+0)	0.447 (-1)
	L = 40	-0.381 (-2)	0.276 (-2)	0.418 (-1)	0.489 (-2)

inelastic cross-sections (Fig. 9), there are very small differences for the 2^+ when the step 0.375 is used and the 4^+ angular distribution changes only when the step size is increased from 0.375 up to 0.4.

From these two examples, one can conclude that the stepsize cannot be too much larger than the diffuseness of the potential and that a product kh of the order of $\pi/2$ can still yield interesting results.

3. ECIS AND DWBA

The resolution of systems of coupled equations, as it had been described above, needs very long time. The step-by-step integration methods cannot be applied to integro-differential equations.

The Distorted-Wave Born Approximation can be used when the coupling potentials are small. If we consider a multiplicative parameter in front of these potentials, the DWBA is the linear term in the expansion of the solution as a function of this parameter; the ECIS method is the computation of the following terms.

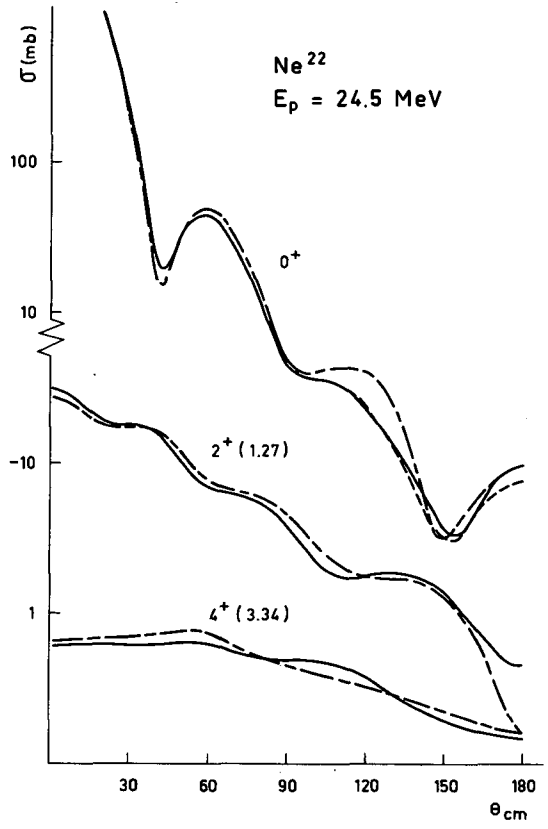


FIG. 6. Cross-sections of 24.5 MeV protons on ^{22}Ne in the rotational model. The full curve was obtained with h = 0.2 fermis, the dashed one with h = 1.0 and the mixed curve with h = 1.2.

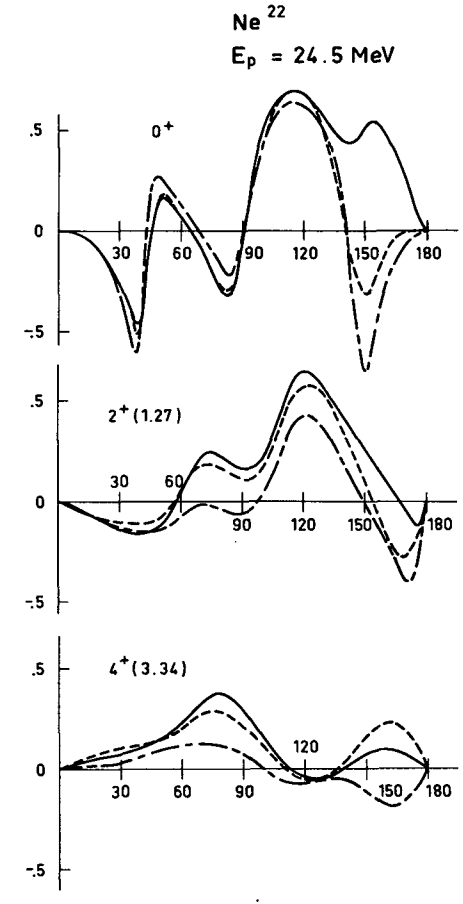


FIG. 7. Analysing powers in the same calculation as for Fig. 6.

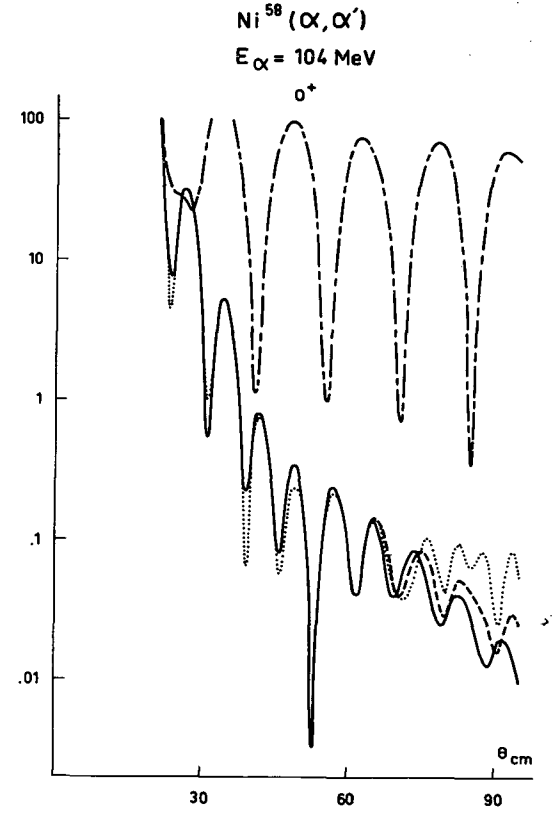


FIG. 8. Elastic cross-section of 104 MeV α-particles on ^{58}Ni in the vibrational model. The full curve was obtained with h = 0.05, the dashed one with h = 0.3, the dotted one with h = 0.375 and the mixed curve with h = 0.4.

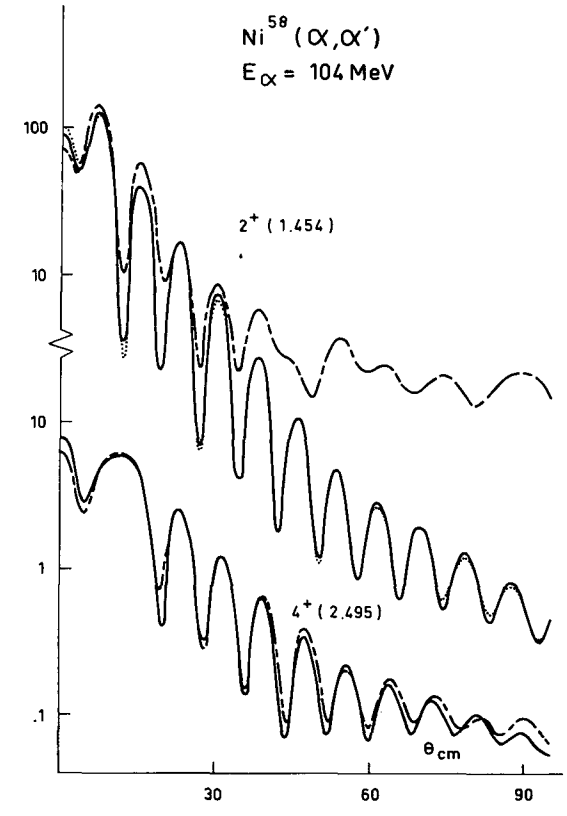


FIG. 9. Inelastic cross-section for the 2^+ and the 4^+ in the same calculation as for Fig. 8.

The first step of these approximations is the separation of the total Hamiltonian into an "unperturbed Hamiltonian" and a "residual interaction". The unperturbed Hamiltonian is diagonal: usually the optical model which gives the best fit to the elastic scattering. We choose

$$\left\{ -\frac{\hbar^2}{2m_i}\left[\frac{d^2}{dr^2} - \frac{\ell_i(\ell_1+1)}{r^2} + k_i^2\right] - V_i^{opt} + V_{ii}(r)\right\} f_i(r)$$

$$= -\lambda \sum_{j\neq i} V_{ij}(r)\, f_j(r) - \lambda \sum_j \int K_{ij}(r, r')\, f_j(r')\, dr' \qquad (27)$$

and we want the solution for $\lambda = 1$.

3.1. The Green function

The left-hand sides of these equations are not coupled. Each of these uncoupled equations has two linearly independent solutions:

1) the regular function f_i^0 completely defined as vanishing at the origin and by its asymptotic value

$$f_i^0(r) = F_{\ell_i}(\eta_i, k_i r) + C_i^0 \left\{ G_{\ell_i}(\eta_i, k_i r) + i\, F_{\ell_i}(\eta_i, k_i r)\right\}$$

which is the usual optical-model wave function;

2) the irregular solution g_i^0 completely defined by its asymptotic value

$$g_i^0(r) = G_{\ell_i}(\eta_i, k_i r) + i\, F_{\ell_i}(\eta_i, k_i r)$$

which can be obtained by a backwards integration from the matching radius to the origin with the asymptotic form as starting values; near the origin, it increases as $r^{-\ell_i}$.

The Wronskian relation

$$g_i^0(r)\, f_i^{0\prime}(r) - f_i^0(r)\, g_i^{0\prime}(r) = k_i$$

holds between these two solutions and allows us to write the solution of the coupled system of equations in an integral form.

Let us denote by $W_i(r)$ the right-hand side of the i-th equation. The general solution is

$$f_i(r) = a\, f_i^0(r) + b\, g_i^0(r) + \int \mathcal{G}_i(r, r')\, W_i(r')\, dr'$$

where $\mathcal{G}_i(r, r')$ is the Green's function of the homogeneous equation, defined by

$$\left\{ -\frac{\hbar^2}{2m_i}\left[\frac{d^2}{dr^2} - \frac{\ell_i(\ell_i+1)}{r^2} + k_i^2\right] - V_i^{opt} + V_{ii}\right\} \mathcal{G}_i(r, r') = \delta(r-r')$$

The definition of the Green's function is not unique. Two conditions must be chosen to define it. The Green's function which is regular at the origin and has an outgoing asymptotic form is

$$\mathcal{G}_i(r, r') = \frac{2m_i}{\hbar^2 k_i} f^0(r_<)\, g^0(r_>)$$

where $r_<$ is the smaller of the two lengths r and r' and $r_>$ is the larger one.

The solution of the coupled system of equations with an ingoing wave for the equation k is given by

$$f_i^k(r) = f_i^0(r)\,\delta_{ik} - \frac{2m_i\lambda}{\hbar^2 k_i}\left[g_i^0(r)\int_0^r \left\{ \sum_j V_{ij}(r')\, f_j^k(r') + \sum_j \int_0^R K_{ij}(r', r'')\, f_j^k(r'')\, dr'' \right\} f_i^0(r')\, dr' + f_i^0(r)\int_r^\infty \left\{ \sum_j V_{ij}(r')\, f_j^k(r') + \sum_j \int_0^R K_{ij}(r', r'')\, f_j^k(r'')\, dr'' \right\} g_i^0(r')\, dr' \right]$$

The coefficient of the outgoing wave in the asymptotic part is given by

$$C_i^k = C_i^0\,\delta_{ik} - \frac{2m_i\lambda}{\hbar^2 k_i}\left\{ \sum_j \int_0^R V_{ij}(r')\, f_j^k(r')\, f_i^0(r')\, dr' + \sum_j \int_0^R \int_0^R K_{ij}(r', r'')\, f_j^k(r'')\, f_i^0(r')\, dr''\, dr' \right\}$$

This is the integral equation for C which involves the solution of the coupled system and the optical solutions of the uncoupled equations.

3.2. The Distorted-Wave Born Approximation

The solution $f_i^k(r)$ can be expanded in powers of λ:

$$f_i^k(r) = \sum_n f_i^{(n)k}(r)\,\lambda^n$$

This expansion can be introduced into the integral equation. Using the fact that the coefficient of each power of λ vanishes, we obtain

for n = 0:

$$f_i^{(0)k}(r) = f_i^0(r)\,\delta_{ik}$$

for n = 1:

$$f_i^{(1)k}(r) = -\frac{2m_i}{\hbar^2 k_i}\left[g_i^0(r)\int_0^r\left\{V_{ik}(r')\,f_k^0(r') + \int_0^R K_{ik}(r', r'')\,f_k^0(r'')\,dr''\right\} f_i^0(r')\,dr' + f_i^0(r)\int_r^\infty\left\{V_{ik}(r')\,f_k^0(r') + \int_0^R K_{ik}(r', r'')\,f_k^0(r'')\,dr''\right\} g_i^0(r')\,dr''\right]$$

for any other value of n:

$$f_i^{(n)k}(r) = -\frac{2m_i}{\hbar^2 k_i}\left[g_i^0(r)\int_0^r\left\{\sum_j V_{ij}(r')\,f_j^{(n-1)k}(r') + \sum_j\int_0^R K_{ij}(r', r'')\times f_j^{(n-1)k}(r'')dr''\right\} f_i^0(r')\,dr' + f_i^0(r)\int_r^\infty\sum_j V_{ij}(r')\,f_j^{(n-1)k}(r')\,dr' + \sum_j\int_0^R K_{ij}(r', r'')\,f_j^{(n-1)k}(r'')\,dr''\right\} g_i^0(r')\,dr'\Bigg]$$

The DWBA is not defined for the wave function but only for the phase shifts. It is the first order in the expansion

$$C_i^k = \sum_n C_i^{(n)k}\lambda^n$$

The $C_i^{(n)k}$ are the coefficients of $g_i^0(r)$ in $f_i^{(n)k}(r)$ when r is greater than the matching radius. The DWBA result is

$$C_i^{(1)k} = -\frac{2m_i}{\hbar^2 k_i}\left[\int_0^R\left\{V_{ik}(r)\,f_k^0(r) + \int_0^R K_{ik}(r, r')\,f_k^0(r')\,dr'\right\} f_i^0(r)dr\right]$$

It involves only the optical wave functions of the ingoing and the outgoing channel. For this reason, the DWBA is widely used, but always to the first order.

The reciprocity relation exists for DWBA but unitarity is lost. The value of the C is proportional to λ and the cross-section to λ^2. In many cases, λ is a parameter of the model obtained by fitting the calculation to

the experiments: the best example is the determination of the deformations in the vibrational model.

The main advantage of the DWBA is the simplicity of its use. Many problems, relevant of the coupled-channel method, are solved with it because the time of computation is much smaller and the results are often quite the same. Almost all the problems involving coupled integro-differential equations are solved by DWBA: an example is the microscopic description of nuclear reactions with finite-range forces and antisymmetrization.

The main defects of DWBA are:

1) the impossibility of solving problems of double excitation. If there is no direct coupling between the equations i and k, but both of them are coupled to the equation j, the DWBA result vanishes. The second-order term is obtained only if $f_j^{(1)k}(r)$ is known everywhere, and the computation of some unperturbed irregular functions g^0 is necessary for that. An example of this situation is the 4^+ in ^{58}Ni shown in Fig. 9.

2) the limitation to the first order introduces doubts as to the results because it is always interesting to know the higher-order effects.

3.3. ECIS

The ECIS method (sequential iteration for coupled equations) is designed in order to solve sets of coupled integro-differential equations when the coupling terms are not too strong. It has only been applied to the macroscopic models of inelastic scattering, with non-diagonal potentials including first-derivative terms. There can be some cases for which this method fails: for example [16], the inelastic scattering of 100 MeV α particles on ^{238}U described in the rotational model with a deformation of 0.4 (deformation too large for this nucleus).

The usual method of solving coupled equations needs too much computing time, because we obtain as many solutions as there are equations, although only one is needed. The ECIS method is a search for this solution by iteration. Its advantage in the usual coupled-equation methods, from the point of view of computational time, is proportional to the number of equations and inversely proportional to the number of iterations needed.

The ECIS method supposes some ordering of the channels: first, the ground state and, after, the state which is most strongly coupled to the ground state; any channel must be coupled to some preceding one.

Each iteration is not related to the corresponding power of the parameter λ and presents some properties which are:

1) the first gives a better result than the DWBA, because there is already an effect in a double-excitation problem;

2) the second often gives the same results as the coupled channels;

3) the last, defined by a convergence test on the phase shifts, must give the same result as coupled equations.

The storage required by this method is larger than for usual coupled equations. There is an integral and a differential way of doing the iterations. The differential one has been used first but we shall present the integral method first.

The equations are ordered: 0 for the ground state (we suppose that there is only one equation for this state), 1, 2, ... for the others states. The zeroth-order approximation is

$$f_0^{(0)}(r) = f_0^0(r) \qquad f_i^{(0)}(r) = 0$$

where i stands for 1, 2, ... Let us suppose that there is no integral term. For the first iteration, we consider the equation with i = 1 and obtain

$$f_1^{(1)}(r) = -\frac{2m_1\lambda}{\hbar^2 k_1}\int_0^R \mathcal{G}_1(r, r')\, V_{10}(r')\, f_0^{(0)}(r')\, dr'$$

then, the equation with i = 2:

$$f_2^{(1)}(r) = -\frac{2m_2\lambda}{\hbar^2 k_2}\int_0^R \mathcal{G}_2(r, r')\left\{V_{20}(r')\, f_0^{(0)}(r') + V_{21}(r')\, f_1^{(1)}(r')\right\} dr'$$

and, in general,

$$f_i^{(1)}(r) = -\frac{2m_i\lambda}{\hbar^2 k_i}\int_0^R \mathcal{G}_i(r, r')\left\{V_{i0}(r')\, f_0^{(0)}(r') + \sum_{j=1}^{i-1} V_{ij}(r')\, f_j^{(1)}(r')\right\} dr'$$

When these operations have been completed on all the equations, the main part of the first iteration is done. As far as the phase-shifts are concerned, $C_1^{(1)}$ is exactly the DWBA result but $C_2^{(1)}$ includes already second-order corrections due to $f_1^{(1)}(r)$. The second part of the first iteration is the computation of

$$f_0^{(1)}(r) = f_0^0(r) - \frac{2m_0\lambda}{\hbar^2 k_0}\int_0^R \mathcal{G}_0(r, r')\sum_{j=1}^{N} V_{0j}(r')\, f_j^{(1)}(r')\, dr'$$

The n-th step of the iteration includes the computation of

$$f_i^{(n)}(r) = -\frac{2m_i\lambda}{\hbar^2 k_i}\int_0^R \mathcal{G}_i(r, r')\left\{V_{i0}(r')\, f_0^{(n-1)}(r') + \sum_{j=1}^{i-1} V_{ij}(r')\, f_j^{(n)}(r') + \sum_{j=i+1}^{N} V_{ij}(r')\, f_j^{(n-1)}(r')\right\} dr'$$

and also

$$f_0^{(n)}(r) = f_0^0(r) - \frac{2m_0\lambda}{\hbar^2 k_0}\int_0^R \mathcal{G}_0(r, r')\left\{\sum_{j=1}^{N} V_{0j}(r')\, f_j^{(n)}(r')\right\} dr'$$

The result of each iteration depends on the specific order chosen for the equations. For each channel, the equations can be ordered with increasing angular momentum, to obtain decreasing couplings, as it was recommended for the channels themselves.

If there is more than one equation related to the ground state, the whole process must be done again for all of them: the efficiency of the ECIS method is proportional to the ratio of the total number of equations to the number of those related to the ground state.

3.4. Integral involving a Green's function

We have discussed fast and precise methods for the integration of differential equations. The computation of

$$f(r) = \frac{1}{k} \int_0^\infty \mathcal{G}(r, r')\ g(r')\, dr'$$

may seem to be trivial, but it is worthwhile to indicate how to perform this operation with the same precision. The usual methods do not apply because the Green's function does not have continuous derivatives.

The integration methods for the differential equation give the function at equidistant points with a precision on the phase shifts of the order h^4. To obtain a similar precision on the integral, let us consider the trapeze method, written as follows:

$$\int_{nh}^{(n+1)h} f(r)\, dr = \frac{1}{2} h\, [\, f(nh) + f((n+1)h)\,] + \frac{h^2}{12} [\, f'(nh) - f'((n+1)h)\,]$$

with an error of the order h^5. Taking into account the expression of the Green's function, we obtain

$$f(nh) = \frac{2mh}{\hbar^2 k} \left\{ g^0(nh) \sum_{i=1}^{n} g(ih)\ f^0(ih) + f^0(nh) \sum_{i=n+1}^{\infty} g(ih)\ g^0(ih) \right\} - \frac{2m}{\hbar^2} \frac{h^2}{12} g(nh)$$

We have taken into account that the function g(r) vanishes at the origin and at the infinity and that the first derivative of the integrand vanishes also at the two ends of the interval. Consequently, in order to obtain a precise value of the integral, the best is to compute:

$$F(nh) = \frac{2mh}{\hbar^2 k} \sum_{i=1}^{n} g(ih)\ f^0(ih)$$

$$G(nh) = \frac{2mh}{\hbar^2 k} \sum_{i=n+1}^{\infty} g(ih)\ g^0(ih)$$

and, after:

$$f(nh) = g^0(nh)\ F(nh) + f^0(nh)\ G(nh) - \frac{2m}{\hbar^2} \frac{h^2}{12} g(nh)$$

With this correcting term, the trapeze method is as precise as the integration methods. Without such a correction term, any method introduces errors of the order h^2.

Such a procedure was suggested by R. Schaeffer [17] when writing a DWBA program for the microscopic description of inelastic scattering. The multipoles of a Yukawa potential, as used in this program for the two-body interaction, behave as a Green's function. A previous method, which used G(nh) defined like F(nh) and the difference G(Nh)-G(nh) instead of G(nh) in the expression of f(nh), gave too important round-off errors when the range of the Yukawa potential was small. The change was introduced when the spin-orbit was added to this program [4]. The spin-orbit interaction is a sum of terms with geometrical coefficients such that the corrections cancel out. Comparison of matrix elements between harmonic-oscillator wave functions without nodes showed discrepancies with values obtained with Moshinsky brackets of the order of 0.1% when the range is equal to the step h and some per cents when the range is half of the step.

3.5. The differential method

The ECIS method can be applied to differential equations, giving at each step exactly the same result as for the integral method already described. Let us write the system

$$f_i''(r) + V_{ii}(r)\, f_i(r) = -\sum_{i \neq j} V_{ij}(r)\, f_j(r)$$

First, the unperturbed regular solution $f_i^0(r)$ must be obtained for each equation. The zeroth order solution vanishes for all the equations not related to the ground-state and is $f_0^0(r)$ for the ground state.

For the first iteration, to begin with, there is the equation i = 1:

$$f_1''(r) + V_{11}(r)\, f_1(r) = - V_{10}(r)\, f_0^{(0)}(r)$$

of which we need a solution which is regular at the origin and purely outgoing beyond the range of the potentials. A numerical solution $z_1(r)$ is obtained with the starting values

$$z_1(0) = 0 \qquad\qquad z_1(h) = 0$$

The solution which we are seeking is of the form $z_1(r) + a f_1^0(r)$ where a is defined by the two conditions:

$$z_1(R-h) + a\, f_1^0(R-h) = C_1^{(1)} \{G_{\ell_1}(R-h) + i\, F_{\ell_1}(R-h)\}$$

$$z_1(R+h) + a\, f_1^0(R+h) = C_1^{(1)} \{G_{\ell_1}(R+h) + i\, F_{\ell_1}(R+h)\}$$

which also give the value of the phase-shift $C_1^{(1)}$. When a is known, the function $f_1^{(1)}(r)$ is formed by adding $af_1^0(r)$ to the numerical solution $z_1(r)$. Then, the i = 2 equation is solved with $f_0^{(0)}(r)$ and $f_1^{(1)}(r)$ in the right-hand side and the process is continued to the last equation.

The second part of the first iteration is to obtain a solution of the equation

$$f_0''(r) + V_{00}(r)\, f_0(r) = -\sum_i V_{0i}(r)\, f_i^{(1)}(r)$$

with an incident wave plus an outgoing wave beyond the range of the potentials. A numerical solution $z_0(r)$ is obtained with the same starting values. The solution needed is also of the form $z_0(r) + af_0^0(r)$ but a is defined by the two conditions:

$$z_0(R-h) + a\, f_0^0(R-h) = F_{\ell_0}(R-h) + C_0^{(1)}(G_{\ell_0}(R-h) + i\, F_{\ell_0}(R-h)$$

$$z_0(R+h) + a\, f_0^0(R+h) = F_{\ell_0}(R+h) + C_0^{(1)}(G_{\ell_0}(R+h) + i\, F_{\ell_0}(R+h))$$

which give also $C_0^{(1)}$.

For the following iterations, all the already known functions are used on the right-hand side.

The storage requirement is the same for the integral and the differential approach: we need the values in all points of the iterated solution and of the functions $f_i^0(r)$, plus the values of the functions $g_i^0(r)$ in the integral approach or the values of the diagonal potentials $V_{ii}(r)$ in the differential one. The integral approach has been introduced recently in a new program ECIS71 and seems to be quicker than the differential approach; the longest part of the calculation is the computation of the right-hand sides which is the same in the two approaches.

3.6. Convergence and Padé approximation

Figures 10 to 13 show results obtained in the vibrational model for inelastic scattering of 16.5 MeV protons on ^{60}Ni. The optical parameters are: V = 49.15 MeV, R = 1.25 fermi and a = 0.65 fermi for the real potential, W = 11.30 MeV, R = 1.25 fermi and a = 0.47 fermi for a surface imaginary potential and V = 5.4 MeV, R = 1.12 fermi and a = 0.47 fermi for the spin-orbit interaction. The energy of the 2^+ is 1.333 MeV. For the ground state, the curves presented as results of first, second, ... order are, in fact, the zeroth, first, ... order.

With a deformation 0.231 which is the normal value, the second iteration practically gives the same results as the coupled-channel method, as is shown in Fig.10 for the cross-sections and in Fig.11 for the polarizations. If the deformation is increased up to 0.4, the fourth iteration is necessary to give the true result (Fig.12 and 13). The difference between the zeroth-order and the first-order result in the elastic cross-section (labelled as first and second order on Fig.12) is characteristic of an increase of the

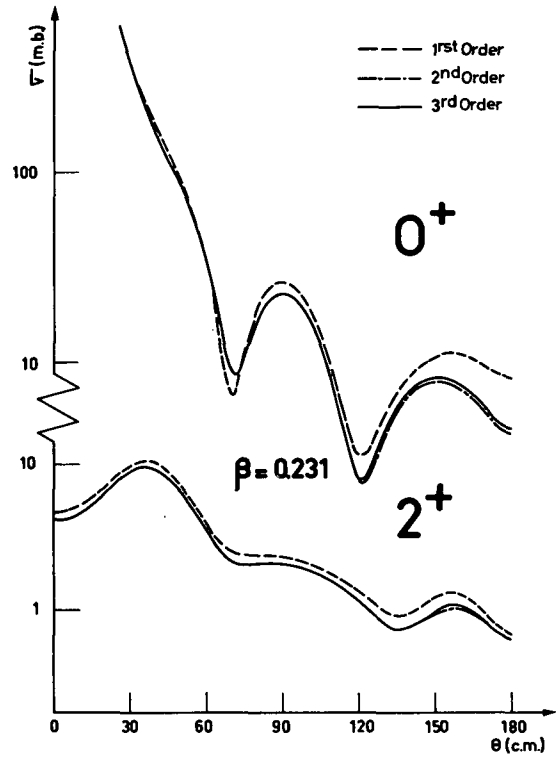


FIG. 10. Results of the successive iterations for the cross-sections of 16.5 MeV protons on ^{60}Ni in the vibrational model with a physical deformation.

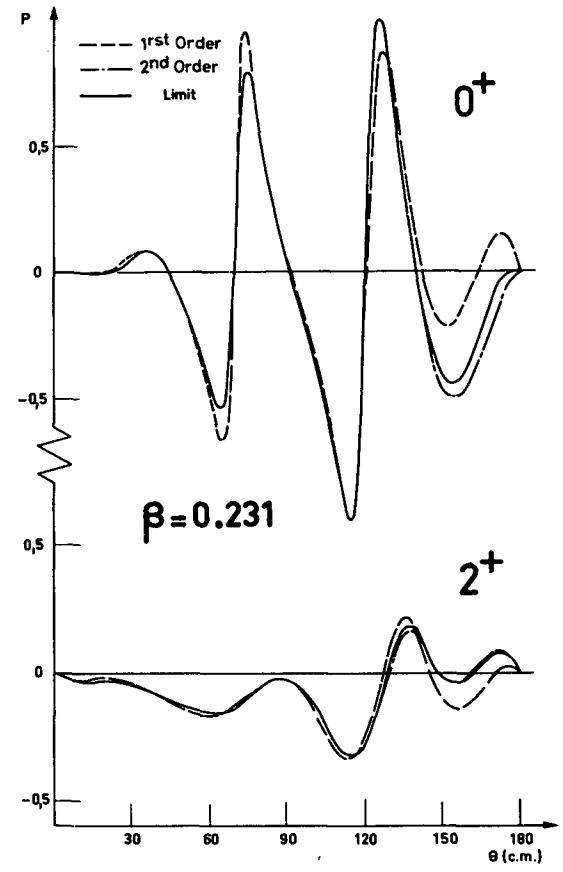


FIG. 11. Analysing powers in the same calculation as for Fig. 10.

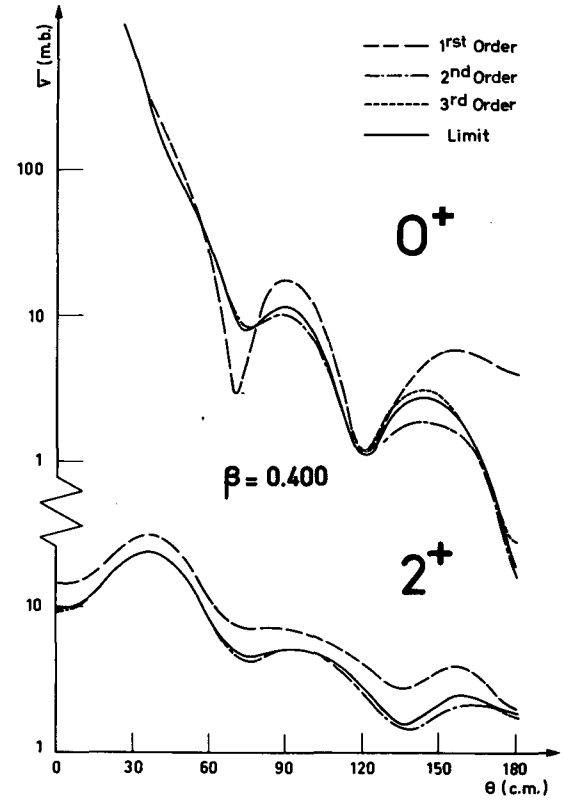


FIG. 12. Same calculation as for Fig. 10, with an increased deformation.

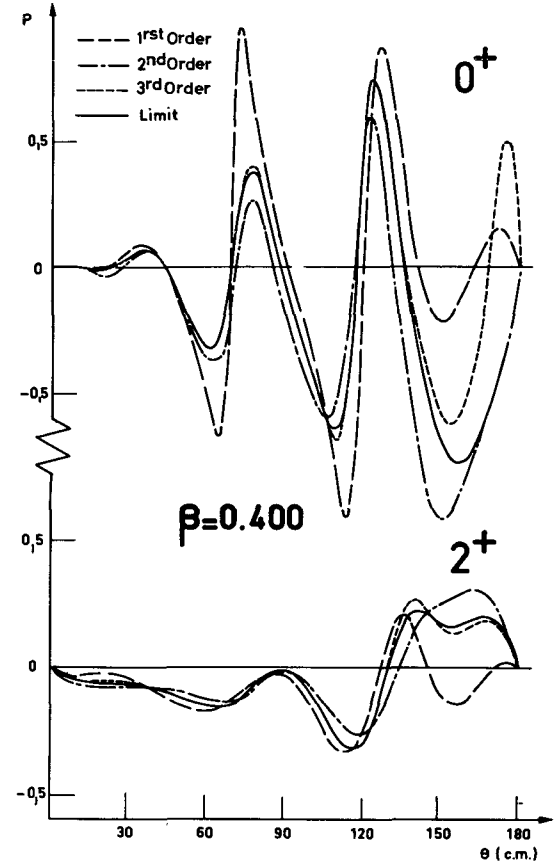


FIG. 13. Analysing powers in the same calculation as for Fig. 12.

depth of the imaginary part of the potential. As the optical parameters were chosen by fitting the zeroth-order result to the elastic experimental data, this example emphasizes the necessity of decreasing the imaginary part when the coupling is strong.

The method is not used with so small a number of iterations. Only, a maximum number of iterations is given (20 for example). For a given value of total angular momentum and parity, two iterations are performed. If for all the equations the differences $C_i^{(2)} - C_i^{(1)}$ are smaller than a given number (0.0001 for example), the system is solved; if not, another iteration is done. The process is stopped at the n-th iteration if the differences between the results of the n-th and the (n-1)-th one are less than this number. Usually, the number of necessary iterations decreases for large angular momenta. When this number is only two, one can assume that the DWBA is sufficient and one can limit the resolution for higher angular momenta to only one iteration.

This method of iteration can give divergent results. The convergence can be accelerated and the possibility of divergence almost eliminated by Padé approximants [18]. The result of the n-th iteration can be written:

$$C_i^{(n)} = \sum_{p=1}^{n} a_p$$

i.e. the sum of the n first terms of the Taylor expansion of some function for the value unity of the variable. The Padé approximant replaces this Taylor expansion by the ratio of two polynomials with the same number of coefficients. An accurate result can be obtained further than the poles of the function which limit the circle of convergence of the Taylor expansion.

As an example of the Padé approximants, let us consider the following equations:

$$y'' + y + V z = 0 \qquad V = A \text{ if } r < \pi$$

with

$$z'' + z + V y = 0 \qquad V = 0 \text{ if } r > \pi$$

and look for the solution which becomes

$$y = \sin r + C_1(A) \exp(ir)$$

$$z = C_2(A) \exp(ir)$$

for r larger than π. $C_1(A)$ is an even function of A and $C_2(A)$ is an odd one. In fact, this is a square well problem because $C_1(A) \pm C_2(A)$ is the phase shift of a single equation $y'' + y \pm Vy = 0$. The convergence radius of the series is around 0.6; but, for A = 3, the Padé approximant gives the good result to 10^{-4} with 9 iterations.

At the fourth iteration, we compute the Padé approximants obtained with the three first iterations and with all of them. If the difference is less than a given value we keep the Padé result. Similarly, the Padé approximant of the n-th iteration is compared to that of the (n-1)-th iteration.

Up to now, we have only found one case for which this method fails: the inelastic scattering of 100 MeV α-particles on ^{238}U. The divergence was perhaps a numerical instability because it did not happen for the same angular momenta in the integral approach and in the differential one; for some angular momenta, the Padé approximants oscillated between different values.

3.7. The ECIS70 code

The Buck and Hill's code for coupled channels was the starting point of the ECIS procedure: only the integration method and the matching was to change. At the beginning, there was the possibility of using usual coupled-channel methods for a part of the channels and iteration for the others. Figure 14 shows results obtained for the elastic scattering of 18.5 MeV protons on ^{58}Ni in the vibrational model and the inelastic scattering on a 2^+ state described by a phonon with β=0.22 and a 4^+ state described by two phonons. The full curve is the coupled-channel result, the dashed curve is the first approximation of ECIS, which is already good for the 4^+; the dotted curve is the result of a coupled-channel calculation between the 0^+ and the 2^+, completed by a first iteration for the 4^+. This possibility was dropped because it was too complicated and did not seem necessary.

The codes were almost completely rewritten when the Blair and Sherif deformed spin orbit was introduced. The first derivatives introduced in the non-diagonal potentials by the deformed spin orbit are obtained by

$$f'(r) = \frac{3}{4h}\left[f(r+h) - f(r-h)\right] - \frac{3}{20h}\left[f(r+2h) - f(r-2h)\right] + \frac{1}{60h}\left[f(r+3h) - f(r-3h)\right]$$

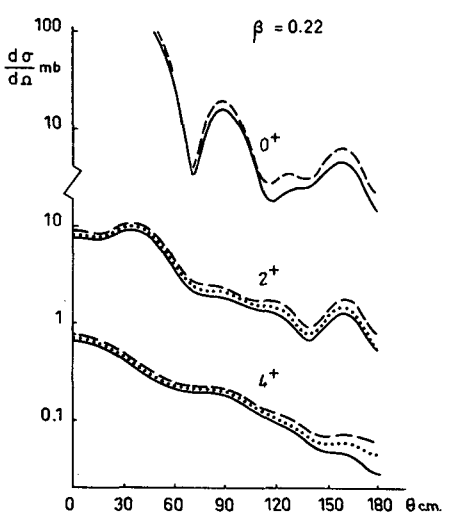


FIG. 14. Inelastic scattering of 18.5 MeV protons on ^{58}Ni in the vibrational model with a two phonons 4^+ state. The full curve is the coupled-channel result. The dashed curve is the first ECIS iteration. The dotted curve is the result of a $0^+ - 2^+$ coupled-channel calculation followed by a DWBA calculation for the 4^+ state.

in the middle of the range of integration, set equal to zero for the three last points and computed by simpler formulas for the two first points. The usual coupled-equation integration method (modified Numerov) is included in the code but cannot be used for deformed spin-orbit potential.

As it is difficult to know the storage requirements in advance, the MAIN defines a large array which is cut into parts by the program itself which defines in it all the arrays needed as soon as their dimensions are known. In this way, there is no limitation, at all, except for the size of the computer. To solve a large problem, there is only a MAIN of five cards to compile in order to change this array. Very recently, Renardy suggested a procedure which defines this array as all the region left free by the program.

There is no change from the original Hill's program for the vibrational model. In the rotational model, the form factors are obtained by Gauss integration. The asymmetric rotational model is added: the form factors are calculated by integration on the sphere with 36 points, the weights of which were obtained once for all by the inversion of the matrix of rotation matrix elements at these points (this procedure means that the potential is supposed to be expanded only with 36 rotation matrix elements, the coefficients of which are obtained by solving a set of 36 linear equations). The number of multipoles is limited to 15 (i.e. L = 8, whereas 36 is L = 14).

For the usual coupled equations, the V_{ij} are computed in as many points at a time as is allowed by the storage. For the iterations, all the potentials can be computed before or at each step of the iteration. There is no difference in the vibrational model at its first order because there is only one form factor by V_{ij}, but for the rotational model the time can be divided by two if the V_{ij} are calculated beforehand. This is a storage problem. It is useless to compute the V_{ij} and to store them on a scratch tape from the point of view of time, at least with our machine.

The helicity formalism is used for amplitudes. Consequently, rotation matrix elements are used instead of Legendre polynomials. They are computed by upwards recurrence relations (which are not the best for large values).

The Coulomb functions [19] are obtained by subroutines written at the Computer Department of Saclay. The geometry coefficients [20] are also written there; they are even too precise for the problem and take a non-negligible amount of time.

4. AUTOMATIC SEARCH FOR PARAMETERS

A comparison of the fits to the experimental data obtained with different sets of parameters may be given quantitatively by introducing the quantity

$$\chi^2 = \sum_i \left[\left(\sigma(\theta_i) - \sigma^x(\theta_i)\right) / \Delta\sigma^x(\theta_i) \right]^2$$

where $\sigma^x(\theta_i)$ is experimentally given at the angle θ_i, and $\Delta\sigma^x(\theta_i)$ is the experimental error. Separate χ^2 can be defined for the cross-section and the polarization in each channel. The total χ^2 is their sum, possibly with some weight. The χ^2 can be corrected in order to take into account the

detector width by using an averaged calculated value and angular error by increasing the experimental error when the slope of the theoretical curve is large. When the absolute normalization λ^x of the experimental data is not known, the criterion defines a normalization λ as minimizing:

$$\chi^2(\lambda) = \sum_i [(\lambda\sigma(\theta_i) - \sigma^x(\theta_i))/\Delta\sigma^x(\theta_i)]^2 + [(\lambda-\lambda^x)/\Delta\lambda^x]^2$$

where λ^x and $\Delta\lambda^x$ are, respectively, an estimated value and the error reducing the range of variation of λ.

4.1. Chi square minimization

To find the minimum of the χ^2, starting from some point $\vec{p}_0$ in the parameter space, some information is needed on the behaviour of this function around the point $\vec{p}_0$. When the number N of parameters under investigation is large, the best information is given by

$$\chi^2(\vec{p}_0 + \vec{\Delta}_p) = \sum_i [(\sigma_i(\vec{p}_0) + \vec{\nabla}\sigma_i(\vec{p}_0)\cdot\vec{\Delta}_p - \sigma_i^x)/\Delta\sigma_i^x]^2$$

$$= A + 2\vec{B}\cdot\vec{\Delta}_p + \vec{\Delta}_p\cdot C\cdot\vec{\Delta}_p$$

where A is the χ^2 itself, and the vector $\vec{B}$ is half the gradient of the χ^2 with respect to the parameters:

$$B_j = \sum_i \left(\frac{d}{dp_j}\sigma_i\right)(\sigma_i - \sigma^x)/(\Delta\sigma_i^x)^2$$

and the matrix C is an approximation of the second derivatives of the χ^2:

$$C_{jk} = \sum_i \left(\frac{d}{dp_j}\sigma_i\right)\left(\frac{d}{dp_k}\sigma_i\right)/(\Delta\sigma_i^x)^2$$

This information is obtained by N + 1 calculations, including a central run and the change of each parameter – one at a time – or only a central run including the computation of N derivatives. The complete computation of the second derivatives would require (1/2)(N +1)(N + 2) calculations.

This approximate quadratic expansion of the χ^2 is positive-definite and there is a minimum at some point $\vec{p}$. If the distance $|\vec{p}_0 - \vec{p}|$ is small in a metric defined by some unit chosen for each parameter, the next run will be at the point $\vec{p}$. If $|\vec{p}_0 - \vec{p}|$ is too large, the next run will be at some maximum distance from $\vec{p}_0$, in the direction of $\vec{p}$. After this, there are various possibilities. If the computation of derivatives does not multiply the computing time by N + 1, the same process is started again. One can do a single calculation without derivatives at the new point and an other one twice further if the χ^2 decreased or in the middle if the χ^2 increased: a parabolic

approximation [21] between $\vec{p}_0$ and the two new points gives a new starting point for the whole process, if it is not too far. There are more elaborate methods [22] which correct the matrix C by computing only one derivative; they suppose that the χ^2 curvature does not change too much from a point to another; at the end, the complete matrix of second derivatives is obtained.

The optical-model codes of Oak Ridge use the parabolic approximation. The optical-model code MAGALI can use the parabolic approximation or not, and the coupled-channel code ECIS71 computes always the derivatives; these two programs obtain the derivatives in about 40% of the time needed for a central calculation. In these programs, the maximum distance between two runs is a multiple (twice, for example) of the distance between the two preceding points; this procedure increases the speed of the search. If the χ^2 increases, the change of parameters is divided (by 10, for example). If the χ^2 does not decrease by, at least, 0.01% at two successive points or if the change of parameters is less than a given value, the search is stopped.

The derivative of C with respect to the parameter p is

$$\frac{d}{dp} C_m^n = \frac{1}{k_m} \sum_{ij} \int \frac{d}{dp} V_{ij} f_i^n f_j^m \, dr \qquad (28)$$

For a single equation, f_i^n as well as f_i^m are the optical solutions. For a coupled system, the derivatives are needed only for n related to the ground state. If m is not also related to the ground state, the ECIS method gives the functions f_i^n but not the functions f_i^m. Consequently, this formula can only be used for the optical model without further assumptions.

4.2. The MAGALI code [23]

The conclusion of Ref. [1] was that an optical-model code was still to be written. The UCLA code SCAT4 was modified into SEEK [24] (and MERCY at Berkeley). The MAGALI code was written in 1966 with the possibilities allowed at that time by the IBM 7094 or CDC 3600 machines. In fact, there are three programs in one (for spin 0, 1/2 and 1) with common arrays in equivalence. It used overlays with two stages (for compatibility with the CDC overlay structure). At the first level of this structure, there is input of data or computation and output. The second branch includes grid, automatic search, Coulomb functions, computation of potential and the second level of overlays which can be six different parts of the program: for computation or output, for spin 0, 1/2 or 1. If I = 2s + 1, the different segments can be called on a CDC machine by:

```
CALL SEGMENT (2*I - 1)  for computation
CALL SEGMENT (2*I)      for output.
```

The number of overlays can be reduced on IBM 7094 machines. This previous version can be used on a CDC machine without changes except more precise Coulomb functions. The size is around $140\,000_8$ and is reduced below 100000_8 by the overlay structure. A different version was written for the IBM-360 with simple precision for potentials, double precision for integration algorithms, matching and amplitudes. Its size is 240K without overlays.

All the sixteen optical-model parameters can be set constant, equal to another, in grid, in automatic search, or all the possible combinations only by giving a grid step or not, a search unit or the number of a preceding parameter to which it must be equal. This mixture of grid and automatic search allows the best fit as a function of a parameter to be found. The grid and search procedure is accelerated by starting from the parameters of the last fit for the second point of a grid and extrapolating the last two results for the following ones. The grid is followed forward and backward in order to do the next calculation with the smallest difference of parameters as possible. The reaction cross-section can also be included into the grid.

The matching radius is determined by a given relative decrease of the potentials and can be changed during the search if the potentials are pushed outside. The Coulomb functions are computed at the beginning, but computed again if the matching point is changed. The number of partial waves is fixed by the requirement that the last C should be less than a given value; they can be only 40 ℓ values for spin 1, and 60 for spins 0 and 1/2.

The modified Numerov method is used with the variable kr. There can be only 400 integration steps and the step size is increased if needed. The derivatives of the phase shifts are computed with only one half of the integration points. The form factors are calculated with the minimum number of exponentials, and also their derivatives, when required. A surface form factor must have a diffuseness larger than the step ($a \geq h/k$) and there is no search on the radius or the diffuseness of a form factor the diffuseness of which is less than two steps. There have been no studies about these limitations, but they are quite natural. At very low energy, the step h must be decreased in order to be smaller than the wave-length inside the potential; the limitations mentioned above stop the computation when measures of caution of this kind are not used.

Time-requirement measurements were made in Ref. [1]. They show that, although the integration time is the main one, the computation of cross-section is quite long. The computation of the derivatives of phase shifts requires computation of the square of the wave function which is 25 to 30% as much work as an integration step and two and a half more than a derivative. As it is done for a step out of two and not from the beginning of the integration for large angular momenta, there is a 15% increase of time as soon as a derivative is required and 75% for ten derivatives. But the derivatives of cross-sections or polarizations need more than 50% of the time necessary for the cross-sections or the polarizations themselves.

Lately, the imaginary spin-orbit potential, which is almost never used, has been replaced, in a variant of the IBM-360 version, by a tensor potential for spin-1 particles. The form factor is a second derivative of the Saxon-Woods form factor normalized to a maximum value unity. With this potential, the radial equations for $j, \ell = j - 1$ and $j, \ell = j + 1$ are coupled. The derivatives of the phase-shifts are calculated by Eq. (28).

4.3. The ECIS71 code

The ECIS71 code is, in principle, the ECIS70 code with automatic search. The integral version of the ECIS method has been added.

There is no clear indication which is the best way of obtaining derivatives with a sufficient accuracy for an efficient search. Many of them are available in this code and can be chosen by some control read in the data. Their

relative efficiency can change with the case. When applied with test cases similar to ^{22}Ne described by Fig. 6 and 7, all of them are equivalent (at least, the χ^2 decreases with all of them).

A first approach is the use of the ECIS method itself. The parameters are slightly changed and the iteration is started with the last solution, with the hope of very fast convergence. However, in the test case, a change of only 1 MeV for the depth of the real potential needs almost as many iterations as the first calculation. A first choice is to perform the iterations up to convergence. A second choice is to use only one iteration, the criterium being not the precision of the derivatives, but that the minimum of the χ^2 is reached in less time. A third choice is the use of DWBA: in the ECIS method, even with only one iteration, we compute the function of equation i = 1, then use it for equation 2, and so on; in DWBA, we do not include the correction of the function of equation 1 in equation 2; except for the computation of right-hand sides, which is the same in the two cases, the number of operations is three times less in the DWBA approximation.

The second approach is the use of Eq. (28). However, only the solution with an ingoing wave in the initial channel is known. Strictly speaking, Eq. (28) can be used only for the variation of the elastic scattering. However, in many cases, the solution with an ingoing wave in the exit channel can be replaced by the uncoupled optical solution. The derivatives of the potential are obtained by difference of the potentials for two slightly different sets of parameters. When the parameters are varied one at a time, many derivatives vanish, and the computation is speeded up by skipping them. This method is the fastest.

The geometrical coefficients are independent of the parameters. They can be calculated only at the first run and stored on a scratch tape. The rotation matrix elements for the experimental angles can be stored on the same tape. The advantages of this storage depends upon the computer. On the IBM 360-91 at Saclay the time requirements for storage are such that the geometrical coefficients can be calculated in about the same time.

The wave functions of the last run are read from a scratch tape and the new functions are written on another scratch tape. These tapes are exchanged when the χ^2 diminishes. If the storage possibilities are sufficient, all the derivatives are calculated at the same time, for each set of equations; if not, they are calculated one after the other. When the derivatives are calculated one after the other, all the wave functions must be read and the time of computation is increased by this operation.

So, the ECIS71 code includes computation of derivatives, using methods ranging from almost the same calculation as for the central run to very fast approximations. In the test cases studied up to now, the χ^2 decreases almost as well with any of these methods, but there is no guarantee that this will always happen.

With overlays, this code needs less than 100 K (one third of which for system subroutines and basic functions as square root or exponential). This figure does not include the main storage for which 80 K is sufficient in almost all the problems.

5. CONCLUSIONS

The most efficient methods of computation are not the first a physicist would think of. Sometimes they differ from the mathematical presentation

of the problem. Our optical-model approach is very different from the one of the coupled-channel calculation.

For the optical-model calculation, we use a numerical method which does not use the wave function directly; other methods which are two or three times slower give the wave function.

For the coupled-channel problem, with the ECIS method, we obtain the wave function itself, which is not given directly by the usual methods. The difference of computation time is drastic, in the case of many equations. Moreover, each step of the ECIS method can be schematized by diagrams; this method is closer to the language of physicists who need some visualization.

The storage requirement of the ECIS method is greater than the one of the other methods. It is a general feature of quick codes. But the storage is limited by the use of large steps of integration which is allowed by the integration method.

REFERENCES

[1] MELKANOFF, M.A., RAYNAL, J., SAWADA, T., Methods Comput. Phys. 6, Academic Press, New York (1966).

[2] TAMURA, T., Rev. mod. Phys. 37 (1965) 679.

[3] RAYNAL, J., Equations couplées et DWBA, Aussois (1968) Report LYCEN, 6804.

[4] RAYNAL, J., Symposium sur les Mécanismes des Réactions Nucléaires et Phénomènes de Polarisation, Université Laval Quebec (1969).
RAYNAL, J., Spin-orbit interaction in the inelastic nucleon scattering, Nuclear Theory Course, Trieste (1971).

[5] BEINER, M., FABRE de la RIPELLE, M., Report IPNO/TH 184, May 1970.

[6] ZAKHARIEV, B.N., PUSTOVALOV, V.V., EFROS, V.D., Sov. Journ. Nucl. Phys. 8 (1969) 234.
REVAI, J., RAYNAL, J., in a communication to the Symposium on the Three-Body Problem and Related Topics, Budapest (July 1971).

[7] NEVIERE, M., CERUTTI-MAORI, G., CADILHAC, M., Optics Communications 3 (1971) 48.

[8] SHERIF, H., BLAIR, J.S., Phys. Letts 26B (1968) 489.
SHERIF, H., Spin dependent effects in proton inelastic scattering - Thesis University of Washington (1968).

[9] JACOB, M., WICK, G.C., Anns of Physics 7 (1959) 404.

[10] COWELL, P.H., CROMMELIN, A.C.D., Appendix to Greenwich Observations for 1909, Edinburg (1910) 84.
MANNING, M.F., MILLMAN, J., Phys. Rev. 53 (1938) 673.

[11] NUMEROV, B.V., Monthly Notices Roy. Astr. Soc. 84 (1924) 592; Publ. Observ. Astrophys. Central Russie 2 (1923) 188.
COLLATZ, L., The Numerical Treatment of Differential Equations, Springer-Verlag (Berlin, 1960) 167.

[12] BLATT, J.M., J. Comp. Phys. 1 (1967) 382 and the majority of "Numerical Analysis" books.

[13] RAYNAL, J., MELKANOFF, M.A., SAWADA, T., Nucl. Phys. A101 (1967) 369.

[14] STORMER, C., Arch. Sci. Phys. et Nat. (1907) 63; COLLATZ, L., ibid.125.

[15] DE VOGELAERE, R., J. Res. Nat. Bur. Std. B54 (1955) 119.

[16] SCHEIMER, G.R., private communication. I am indebted to Dr. Schweimer for a serious check of ECIS70.

[17] SCHAEFFER, R., Un Modèle Microscopique pour la Diffusion Inélastique de Protons à Basse et Moyenne Energie, Thesis Orsay (1970) Report CEA-R-4000.

[18] PADE, H., Sur la représentation approchée d'une fonction par des fractions rationnelles, Ann. Sci. Ec. Norm. Sup. Paris 9 (1892) 1 and 16 (1899) 395.
WALL, H.S., Continued Fractions, Van Nostrand, New York (1948).

[19] BARDIN, C., DANDEU, Y., GAUTIER, L., GUILLERMIN, J., LENA, T., PERNET, J.M., Note CEA-N-906 (1968).

[20] LAFON, R., Report DCE-Saclay No. 326 (1967).

[21] MADDISON, R.N., Proc. Phys. Soc. (London) 79 (1962) 264.

[22] POWELL, M.J.D., The Computer Journal 7 (1964) 303.

[23] RAYNAL, J., A Fortran-IV Program for Automatic Searches with the Nuclear Optical Model for Spin 0, 1/2 and 1-Particles, DPhT/69-42 (1969).

[24] MELKANOFF, M.A., RAYNAL, J., SAWADA, T., SEEK, a Fortran Program for Automatic Searches in Elastic Scattering Analyses with the Nuclear Optical Model - UCLA Report 66-10 (1966).

COMPUTER SIMULATION IN SOLID-STATE PHYSICS

R. BULLOUGH
Theoretical Physics Division,
Atomic Energy Research Establishment,
Harwell, Berks,
United Kingdom

Abstract

COMPUTER SIMULATION IN SOLID-STATE PHYSICS.

Various applications of computer simulation methods applied to the study of defects in crystalline solids are surveyed. The discussion emphasizes the physical significance of the various applications rather than the computational techniques involved.

It is not possible here to adequately survey all the aspects of solid-state physics to which computer simulation methods have been applied. I shall therefore restrict the discussion to the defect solid state; this is not a completely arbitrary restriction since I believe that computer simulation methods have proved particularly valuable in increasing our understanding of defect properties.

It is useful to identify five separate defect fields in which computer simulation has proved valuable:

1. Fast neutron cascade and ion penetration studies
2. Shock waves in crystalline solids
3. Thermal motion and atom transport
4. Static defects – their atomic configurations, self and interaction energies
5. Computer simulation of electron microscope diffraction contrast defect images

Each of these fields involves distinct simulation procedures, and to convey the overall scope of computer simulation to the defect solid state, each field is briefly discussed. Those fields that have provided particularly fruitful areas for computer simulation are discussed in more detail.

1. Fast neutron cascade and ion penetration studies

A knowledge of the numbers and distribution of point defects produced by either fast neutrons or high-energy charged particles interacting with solids is an important aspect of radiation damage. For example, such information is required for understanding the technological problem of swelling of structural components in the core of fast reactors [1].

When a fast neutron or high-energy charged particle interacts with an atom in a solid, the atom can be displaced with a kinetic energy which can exceed 100 keV. This primary knock-on atom (PKA) interacts with

its neighbours in the solid and causes a cascade of displaced atoms to develop until the maximum kinetic energy falls below the displacement threshold for the solid [2]. The energy dissipation in the cascade is achieved by a combination of elastic and inelastic processes, the latter being of major importance for high-energy collisions. The essential damage function we wish to compute is $\nu(E)$, which is the number of atom displacements (number of interstitials and vacancies) resulting from a PKA of energy E. Previous analytic theories of such cascade formation have been developed for amorphous solids and yield expressions for $\nu(E)$ of the form

$$\nu(E) = KE/2E_D$$

where E_D is the displacement threshold energy (typically ~ 25 eV for copper) and K is referred to as the displacement efficiency (K = 1 for a hard-sphere model with no anelastic energy loss [3]).

Such cascades were first simulated by Beeler and Besco [4] in crystalline solids in which both thermal vibrations and anelastic losses were ignored. More recent calculations by Torrens and Robinson [5] are more sophisticated and have culminated in a cascade program of great generality named "MARLOWE" which is being continuously developed by the theoretical groups at Oak Ridge, Saclay and Harwell. The impetus for this work stems, of course, from interest in the fast reactor design problems [1].

The simulation is achieved by a relatively simple two-body collision model. Such a two-body collision model is clearly appropriate at high energies; however, the model is also satisfactory even at low energies if a suitable choice of anisotropy in the displacement energy is made.

In Fig. 1, a schematic representation of the collision cascade is given. The deflection and energy transfer in each binary collision depend on the form of the repulsive interatomic potential and the magnitude of the impact parameter; in general, the result of the collision requires a relatively simple quadrature. In the MARLOWE program the collision event is preceded by a search routine which checks whether simultaneous collisions should be included. In such cases the simple binary collision model is replaced by a more complex multiple collision model [5]. At the end of each collision the electron excitation energy is removed from the kinetic energy of the projectile atom. Since the entire cascade is produced in a total time less than a lattice vibration period the effects of temperature on the function $\nu(E)$ can be incorporated using a static model. In fact, in MARLOWE, the Debye model is used and each atom is given a suitable small thermal displacement from its lattice site and then considered stationary for the duration of the collision. The damage function $\nu(E)$ is then constructed by simulating for each PKA energy E a total of ten cascades with the direction of the PKA randomly chosen. To give an idea of the magnitude of the computations: on an IBM 360 91 machine, a 10 keV cascade requires about 2 seconds machine time and a 100 keV cascade requires about 70 seconds.

An interesting and important result of the somewhat preliminary calculations using MARLOWE is that the early Kinchin-Pease result [3] is quite good if the PKA energy is simply reduced by the total inelastic energy loss. This clearly indicates, in complete contrast to the first simulations of Beeler and Besco [4], that channelling processes (very

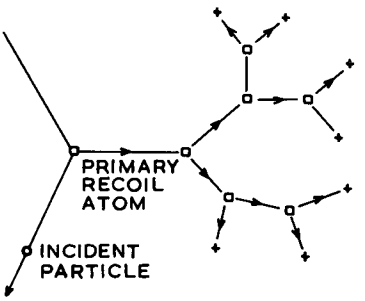


FIG. 1. Schematic representation of the collision cascade.
+ displaced atom; □ vacant site.

structure-dependent) are not important in the overall energy loss process. However, it should be emphasized that much more work is required, especially concerning the effects at low energies, before too much confidence can be put in the results. The annealing characteristics are particularly important, since we really wish to know the net number of point defects that can contribute to the gross swelling process and not just the instantaneous number produced in each cascade. Such annealing studies can be made by the simulation procedures [6] used to study actual thermal motion and atom transport discussed below.

The ion penetration or channelling simulations have also been accomplished using a binary collision model. The basic observation is that a beam of energetic ions incident upon a crystalline target close to one of its principal crystallographic axes or planes exhibits unusually small rates of energy loss and penetrates very deeply into the target. It is worth emphasizing that such channelling phenomena were actually first observed in a computer simulation experiment by Robinson and Oen [7] and then subsequently found in many real systems. Such trajectories can be made highly regular and provide a powerful technique for studying the forces between the ion and the atoms of the target at separations of about 1 Å, which is a separation very difficult to examine by other methods [8].

Concerning the actual simulation, it will suffice to say that because of the high velocities of the channelled ions and the large impact parameters associated with the individual collisions the conditions for the application of the classical impulse approximation are met. The ions suffer only a slight deflection in each collision and in these circumstances the discrete atomic nature of the planes bordering the channel can be ignored and an equivalent planar continuum can be used.

2. Shock waves in crystalline solids

I shall discuss this subject only very briefly; I have included it because the simulation work that has been done, though rather preliminary in nature, does clearly show that the usual continuum understanding of shock propagation is inadequate. To the best of my knowledge, no truly three-dimensional

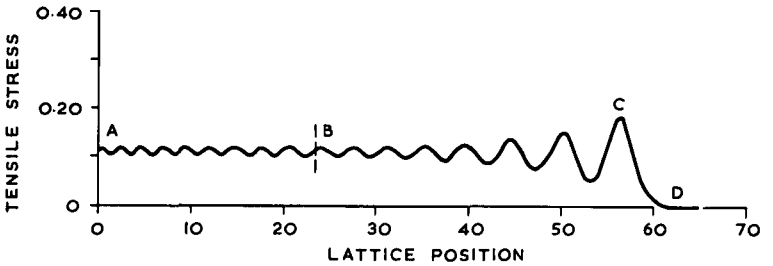


FIG. 2. Stress profile for one-dimensional face-centred cubic lattice [9]. Note the oscillations behind the shock front C – such oscillations are not present in the usual continuum model.

studies have yet been made. Tsai and Beckett [9] and Verlet [10] have made one- and two-dimensional studies. The two-dimensional lattice is held in equilibrium under a simple Morse potential extending to fourth neighbour, and such lattice models have been used to study the actual dynamics of shock propagation, the lattice energies and the value of the Gruneisen parameter (a measure of the partition of energy) behind the shock front. The shock was introduced into the stationary semi-infinite lattice by allowing a similar semi-infinite lattice, travelling with a uniform velocity, to impinge on its surface. The subsequent atom motions were followed by an iterative procedure which involved finding the positions of all the lattice points at each instant of time, so that the equations of motion for all the atoms affected by the shock wave were satisfied within a prescribed tolerance. From such atom positions, the density and stress profiles were obtained, and the propagation velocity of the shock front was then obtained from the positions of the shock front at different instants of time. A typical stress profile behind the shock front is shown in Fig. 2. For precise details of the lattice model, with its various constrained degrees of freedom, the paper by Tsai and Beckett [9] should be consulted.

3. Thermal motion and atom transport

One of the most interesting simulations of a vibrating lattice in which various intrinsic diffusion processes were studied has been made by Tsai, Bullough and Perrin [6]. These authors used a small crystallite of body-centred cubic iron that was maintained in equilibrium at the correct lattice spacing with a central force potential. The potential, due to Chang [11], was constructed from a set of quartic polynomials so that the response of the lattice was correct to second and third order. A temperature was assigned to the lattice by ensuring a Maxwellian distribution to the velocity components of the lattice points in random order. Subsequent motion of the lattice points and any defect (vacancies or interstitials) were calculated by solving the classical equations of motion for the atoms.

Such a dynamic simulation requires a great deal of repeated relaxation, and the largest block of lattice (with periodic boundary conditions) that could be studied was only $10 \times 10 \times 10$ unit cells (about 2000 atoms). About 30 complete relaxations per period of atomic vibration were required to ensure

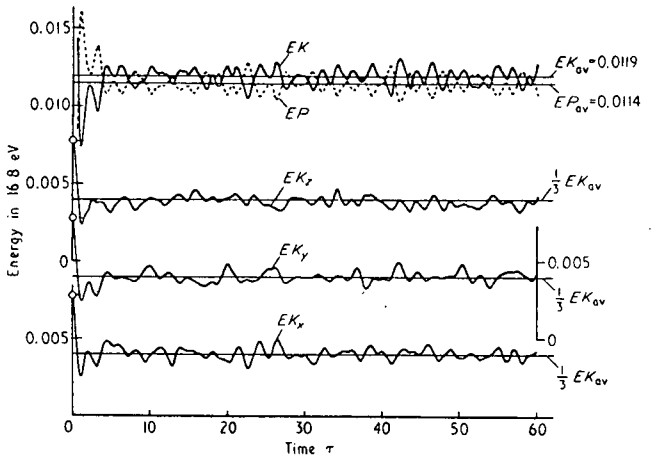


FIG. 3. Time development of the energies in the b.c.c. lattice model (250 atoms and 1 interstitial atom) at a simulated temperature of $T/\Theta = 4.08$. EP and EK are respectively the potential and kinetic energies per atom in units of 16.8 eV. EK_x, EK_y and EK_z are the kinetic energies in the x, y and z degrees of freedom. The curves EK_x and EK_y have been displaced downward from the curve EK_z in order to show all the curves more clearly. The starting points, marked by circles, show the kinetic energy to be 0.0078 in each degree of freedom. The averaged values $EP_{av} = 0.0114$ and $EK_{av} = 0.0119$ were obtained over a period from $\tau = 10$ to 200. The difference between EP_{av} and EK_{av} shows the lack of equipartition of the potential and kinetic energies due to the anharmonicity of the interaction potential. (From Ref. [6]).

sufficient accuracy throughout the oscillation. The precise temperature was defined by assigning to each (perfect) lattice point, in random order, velocity components with a Maxwellian distribution and a mean kinetic energy of kT in each degree of freedom. Since the kinetic energy is a maximum and the potential energy a minimum, this is an allowable though highly improbable configuration. However, as shown in Fig. 3, as the lattice points are allowed to move from their static equilibrium sites, the total kinetic energy 3 kT very quickly becomes equally distributed between kinetic and potential energy; thus in only one period of atomic vibration we have approximate equipartition of energy, and at the same time the atomic displacements change from the zero initial displacements to an approximate Maxwellian distribution. The deviation from precise equipartition, due to the anharmonicity in the potential, is also indicated in the figure.

This model was used to study the diffusion of vacancies and interstitials and yielded interesting activation energies and entropies. It is clear that this simulation procedure, which enables actual annealing studies to be made on the computer and also allows the precise jump frequencies to be observed, is of great potential value. This is particularly true when the speed of arithmetical operations is increased (with present machine speeds such complete dynamic simulations are only just feasible) and when more physically reliable interatomic potentials are available.

4. Static defects – their atomic configuration, self and interaction energies

The two simulation studies that I shall use to briefly illustrate the static calculations are from Bullough, Perrin and Englert. Bullough and Perrin [12] studied the aggregation of interstitials in a body-centred cubic lattice (iron), and Perrin, Englert and Bullough [13] used the simulation procedure to make an extensive study of point and line defects in a face-centred cubic lattice (copper).

When certain b. c. c. metals, including α-iron, are irradiated with neutrons the displaced interstitial atoms are observed to aggregate in the form of planar circular platelets or edge dislocation loops orientated on {111} planes [14, 15]. To understand why the aggregation occurs with this particular morphology involves a knowledge of the mode of nucleation of the aggregate; thus we wish to know: what is the form of the interstitial aggregate nuclei and at what stage in its growth and by what process does it become the observed dislocation loop?

This nucleation problem was investigated by a static simulation procedure using an atomic parallelepiped containing almost 6000 atoms. The atoms were held in equilibrium, at the correct body-centred cubic α-iron lattice spacing, under a pair potential constructed by Johnson [16]. To construct the potential V(r), Johnson [16] fitted it to the short-range elastic moduli (the long-range electronic contributions were removed by the method of Fuchs using a free electron model). The potential was represented by three splines such that V(r) was set to zero with zero slope and curvature midway between second and third neighbours and was matched at short range to the radiation damage potential of Erginsoy, Vineyard and Englert [17]. A dynamic relaxation method was used in which the finite difference form of the classical equations of motion was solved by periodic "quenching" as the total kinetic energy passed through a maximum value [12, 18]. This relaxation method ensures very rapid convergence to the absolute minimum in total potential energy and has the distinct advantage that metastable configurations can usually be avoided and the true stable configuration found. To avoid any spurious effects of possible boundary constraints on the defect configuration, the equilibrium configuration was first obtained with the surface atoms held rigid in their perfect crystal positions and then forces were imposed on these atoms to simulate the surrounding infinite crystal. The whole assembly, including the surface atoms, was then allowed to further relax and the forces on the latter were raised in direct proportion to their subsequent displacements. In this way the internal atomic configuration was not prejudiced.

The procedure was to successively insert interstitials near the centre of the assembly, relaxing completely between each addition, and to allow the aggregation to proceed. We found [12], in agreement with Johnson's original work on the single interstitial [16], that the isolated interstitial adopted a split dumb-bell configuration along a [110] axis. When subsequent interstitials were put in the vicinity of this interstitial they were found to aggregate with all their axes parallel to the same [110] axes. At this stage we were able to identify the nucleation plane as the (110) plane and it was apparent that the "two sides" of the nucleus were beginning to shear over to try to remove the high energy stacking fault across the nucleus. The transition to the observed morphology actually occurred when the nucleus

had incorporated sixteen interstitials. At this stage the (110) fault completely sheared over and the rhombus-shaped aggregate became a glissile dislocation with a [111] Burgers vector. This loop then lowered its self-energy by slipping on its {110} faced glide prism into a pure edge orientation such that it lay on the observed (111) plane. A schematic representation of the single interstitial in α-iron is shown in Fig. 4 where we recognize the split dumb-bell configuration with its axis along a $\langle 110 \rangle$ direction; a projection of the actual computer output is given in Fig. 5 where the dumb-bell form is confirmed. It is particularly interesting to note the extensive relaxations along the $\langle 111 \rangle$ close-packed direction associated with this

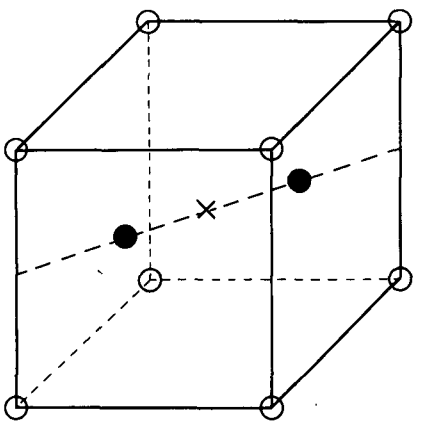


FIG. 4. Split dumb-bell configuration (schematic) of a single interstitial in the body-centred cubic lattice.

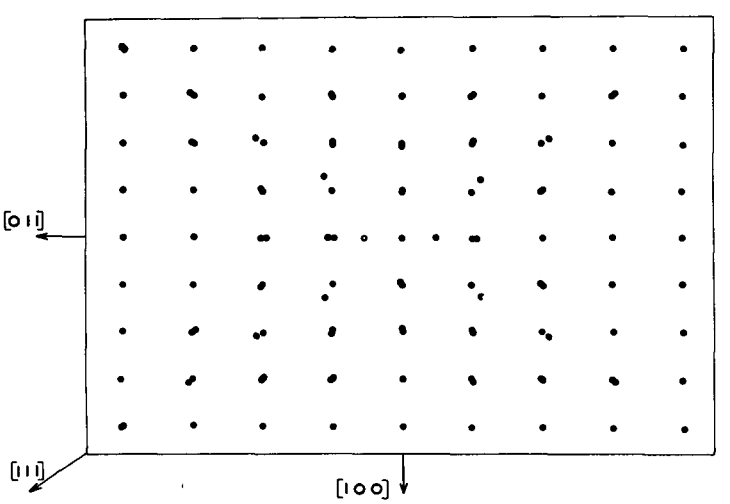


FIG. 5. Projection on to the $(1\bar{1}0)$ plane of the relaxed atomic configuration around the split dumb-bell single interstitial. The extra interstitial atom is marked o.

point defect. In Fig. 6, the relaxed configuration associated with four such interstitials is given, and the final sixteen-interstitial glissile platelet is shown in Fig. 7. The simulation study thus provides a consistent and detailed explanation of the origin of the interstitial {111} dislocation loops observed in irradiated α-iron.

The direct simulation method is particularly useful for the study of dislocations in crystalline solids since the complex topological features of such defects preclude the use of the somewhat simpler lattice statics methods [19, 20, 21] that have proved so valuable in the study of point defects in metals and ionic solids. To obtain the atomic configuration associated with a straight dislocation in copper, it is again necessary to set up a parallelepiped of discrete atoms such that the atoms form a perfect face-centred cubic lattice with the copper lattice spacing, subject to an appropriate interatomic potential V(r). The potential used by Perrin, Englert and Bullough [13] was originally constructed from a set of ten spline functions by Englert and Tompa [22] by fitting to the following physical data: the elastic constants, the force constant data [19] which provided the best fit to the phonon dispersion data of Sinha [23], the best value of vacancy formation energy (1.09 eV), the experimentally observed intrinsic stacking fault energy (70 erg/cm^2), and the Born-Mayer radiation damage potential determined by Gibson et al. [18] for interatomic distances less than the first-neighbour separation. Finally, it was truncated at the third neighbour with zero slope and curvature and together with the volume-dependent term it was constrained to hold the copper lattice in equilibrium at the correct lattice parameter (3.608 Å). A particular feature of this potential is its oscillatory form between the second and third neighbour. The presence of such an oscillation is consistent with the long-range potentials deduced by second-order perturbation theory with a pseudopotential

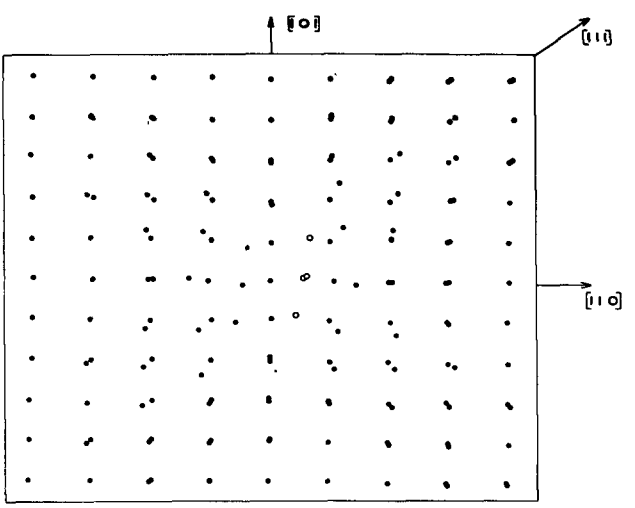


FIG. 6. Projection on to the $(1\bar{1}0)$ plane of the relaxed atomic configuration around a group of four interstitials. The axes of the component split interstitials are all parallel to each other but a small shear has occurred across the two sides of the platelet.

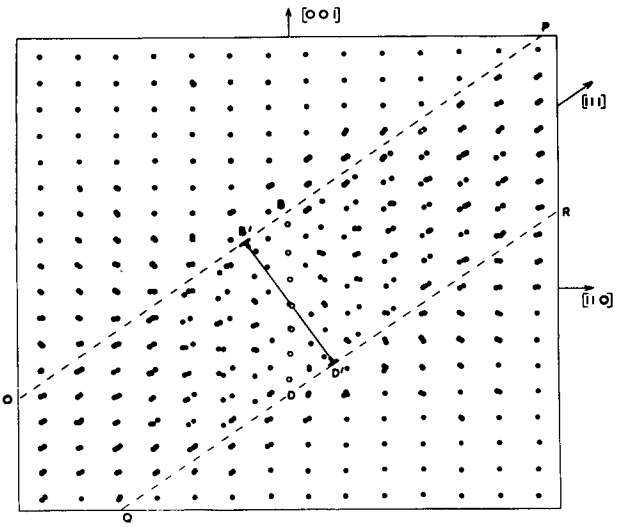


FIG. 7. Sixteen-interstitial platelet. The fault has disappeared and the platelet has become a glissile dislocation loop which lowers its energy by gliding into a {111} orientation B' D'.

[24]; it appears in the present empirical construction as a direct result of the constraint imposed by the fit to the stacking fault energy. The faces of the parallelepiped were appropriate {110}, {111} and {112} crystallographic planes and the dislocation was arranged to lie through the centre of the assembly and orthogonal to the two {112} faces. In the direction of the dislocation line, specifically the $[11\bar{2}]$ direction, the assembly was only six lattice planes thick, and periodic boundary conditions were imposed across these two $(11\bar{2})$ faces; the dislocation was thus automatically long and straight. To accommodate the long-range strain field of the dislocation, the assembly was made as extensive as possible in the two directions orthogonal to the dislocation line. About 900 atoms were actually relaxed and the required dislocation configuration was imposed on the initially perfect lattice by giving the boundary layer of atoms the appropriate, anisotropic elastic displacements.

Several important dislocations and their interactions with point defects have been studied. It will suffice to describe briefly some of the results for the dissociated pure edge dislocation. This is one of the common glissile dislocations in an f. c. c. metal like copper [25] and clearly an understanding of its properties will provide useful insight into the overall deformation properties of copper. It lies along the $[11\bar{2}]$ direction with a total Burgers vector $\vec{b} = a/2\,[\bar{1}10]$ and can dissociate into a pair of Shockley partial dislocations by the reaction [25]

$$\frac{a}{2}[\bar{1}10] = \frac{a}{6}[\bar{1}2\bar{1}] + \frac{a}{6}[\bar{2}11]$$

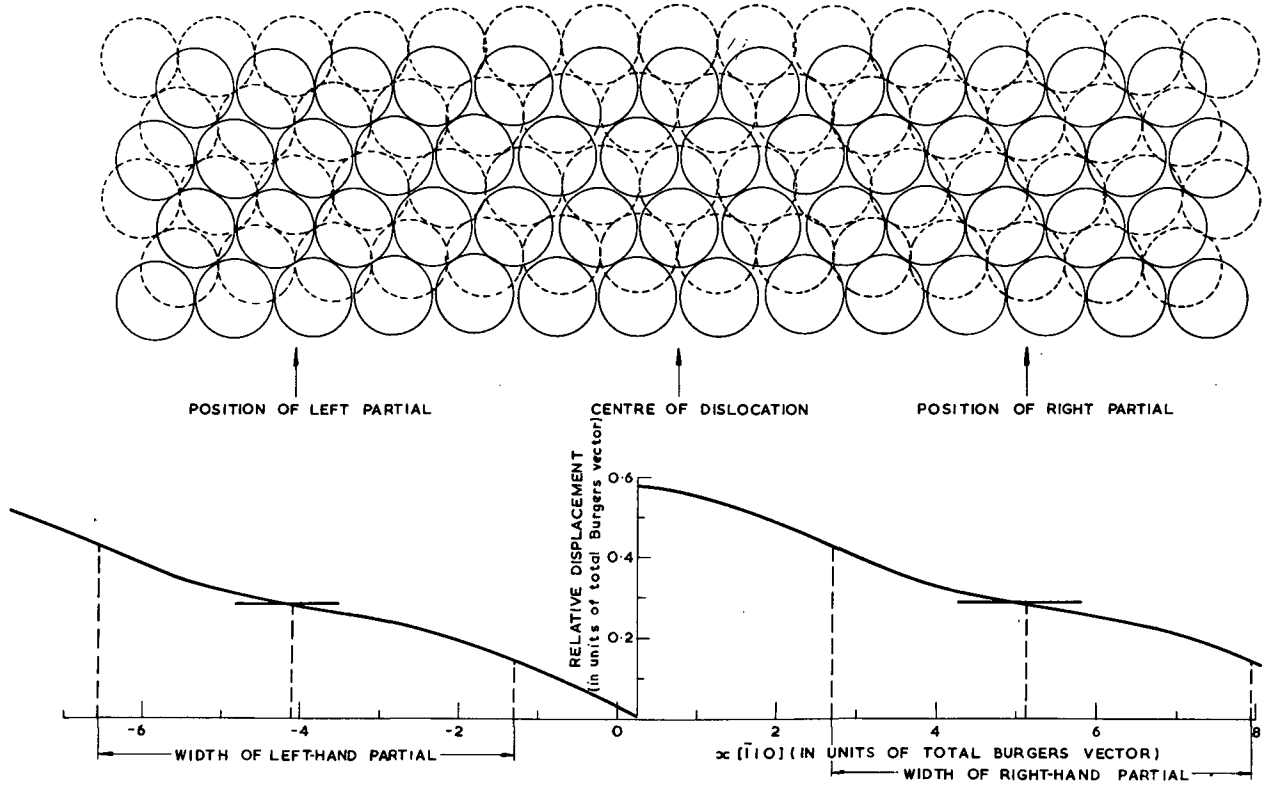


FIG. 8. (111) projection of the atoms above and below the slip plane for the relaxed dissociated edge dislocation and the variation of relative displacement across this slip plane. The relative displacement curves were obtained by calculating the deviation of the atoms in the upper (111) slip plane from the centroids of the corresponding triangles of atoms in the lower (111) slip plane. The precise positions and widths of the two partials are indicated by the arrows. – – – upper (111) plane; ——— lower (111) plane.

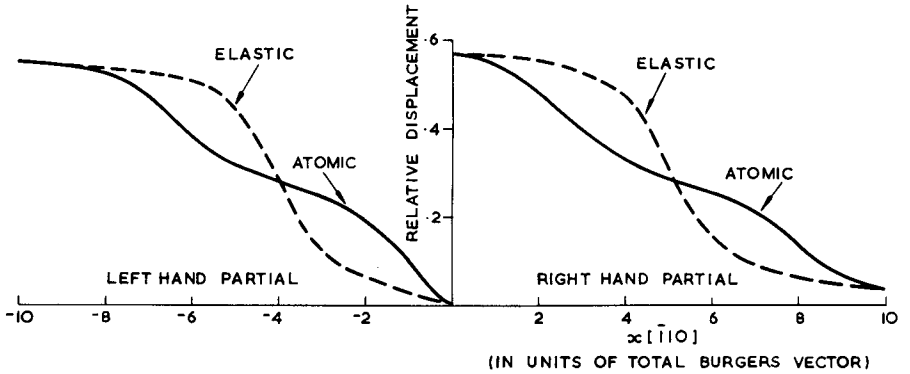


FIG. 9. Comparison between the (anisotropic) elastic and fully relaxed relative displacements across the slip plane of the dissociated edge dislocation. In both cases the partials are separated by 9.2 b. The elastic partials have a width of about 2 b whereas the relaxed partials have each increased their widths to just over 5 b.

The equilibrium separation of these partials arises from a balance between the elastic repulsion between the partials and the attraction from the stacking fault separating them. In the present atomic model this separation was obtained by a self-consistent method which involved repeatedly recalculating the boundary displacements as the partials moved towards their equilibrium separation. It is essential, because of the unavoidable restrictions on the size of the assembly, that the boundary setting should be exactly consistent with the configuration near the centre of the dislocation. With an initial assumed separation of 8 b the iterative process converged to give the final partial separation of 9.2 b.

A (111) projection of the atoms above and below the slip plane of the dislocation is shown in Fig. 8. The positions of the two partials are indicated and the stacking fault arrangement at the centre of the dislocation is apparent. The two partials are very wide, as indicated in detail in Fig. 9, where we see that the widths have increased from about 2 b to just over 5 b in going from the elastic continuum solution to the relaxed atomic configuration. This result is in striking agreement with the known ductility of copper; the critical shear stress to move a dislocation is an exponentially decreasing function of its width [25]. Also a careful study of the tensile strains in the slip plane of the dislocation shows that, in contrast to the elastic model, the strains have an oscillatory form; this result is in beautiful agreement with Parson's recent high-resolution electron microscopy observations of dislocation cores in such metals [26].

Finally, the model has been used to calculate the interaction energy between intrinsic point defects (vacancies and interstitials) and dislocations. To do this it was necessary to extend the parallelepiped in the dislocation direction and then drop the periodic boundary conditions. The boundary atoms of a smaller three-dimensional subassembly were held in their previous relaxed dislocation positions and the point defect was placed near the centre of the subassembly; the internal atoms were then relaxed and the interaction energy deduced. The position of the point defect relative to the dislocation was then varied by simply changing the location of this

atomic subassembly relative to the dislocation itself. These calculations have shown that interstitials have a much larger interaction with dislocations than do vacancies. This is a not unexpected result but nonetheless gratifying since, for example, such a preferential interstitial attraction was a fundamental hypothesis of our theoretical explanation of void growth in metals irradiated to a high dose with fast neutrons [1]. Of particular interest to dislocation theory was the observation that at very short range the vacancies are bound to the dislocation by the second-order inhomogeneity interaction, whereas at distances of only one neighbour separation from the slip plane the first-order size effect appears to dominate and the vacancies are repelled from the dilated regions and attracted to the compressed regions [27]. Furthermore, it was found that the point defects interacted strongly with the entire faulted region and not just with the partials themselves. The interstitial interaction was dominated completely by the size effect interaction with strong attraction to the dilated regions.

5. Computer simulation of electron microscope diffraction contrast defect images

The results of theoretical electron microscope image calculations have usually been presented as profiles giving the variation of transmitted or diffracted intensity along a line in a plane normal to the electron beam whereas the corresponding experimental results are normally in the form of a micrograph. It therefore followed that computed intensities displayed as a simulated micrograph might ultimately be a more useful representation of the theoretical result since an easy and immediate comparison with the experimental micrograph would be possible. The simulation of such micrographs, first achieved by Head [28] using a computer line printer, has now been developed on a Stromberg-Carlson 4060 microfilm recorder which is inherently more versatile and capable of much better definitions [29, 30]. The S-C 4060 plotter has the capability of generating a set of 116 alphanumeric and special characters on a grid of 4096 × 3072 addressable positions. The output is normally in the form of a 35 mm positive. Images are formed by constructing a square mesh of about 30 000 closely spaced

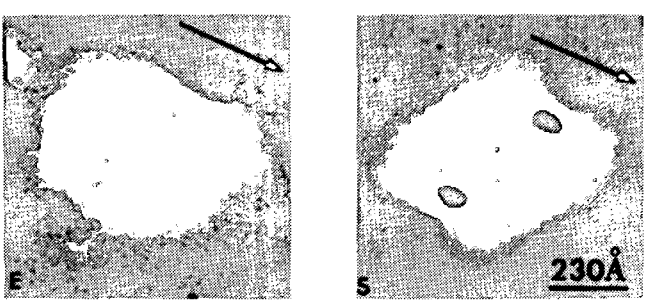


FIG. 10. Image comparison for a perfect dislocation loop in a molybdenum crystal. The diffracting vector is [121]. The plane of the micrograph is (012) and the loop radius is 230 Å. E – experimental micrograph; S – computer-simulated micrograph.

small dots (typical spacing ~ 0.1 mm on 35 mm film) whose intensities and sizes are varied by repeated plotting. Repeated overprinting of the dots gives a set of levels of grey which is used to represent the range of theoretical intensities that arise in the computation of the image of, for example, a lattice defect in a crystal.

By careful calibration the optimum density of points and number of overprint can be obtained in order to match precisely the simulated and real electron microscope images. The theoretical intensities are calculated by numerically integrating the wave equation for the transmission of electrons through the defect crystal [31]. To illustrate the results of this procedure, a comparison between an experimental and simulated image of an edge dislocation loop in molybdenum is given in Fig. 10. The details are given in the figure caption. It is at once apparent that the simulated image is quite as good as (if not better than) the actual micrograph. This image simulation technique clearly has very wide applications and should provide a very useful aid to the experimentalist in the interpretation of electron micrographs of crystal defects.

REFERENCES

[1] BULLOUGH, R., PERRIN, R.C., Proc. Albany Conf. Radiation Induced Voids in Metals (CORBETT, J.W., Ed.), Albany, N.Y., June 1971, CONF-710601.

[2] THOMPSON, M.W., Defects and Radiation Damage in Metals, Cambridge University Press, Cambridge (1969) 143.

[3] KINCHIN, G.H., PEASE, R.S., Rep. Prog. Phys. 18 (1955) 1.

[4] BEELER, J.R., BESCO, D.G., J. appl. Phys. 34 (1963) 2873.

[5] TORRENS, I.M., ROBINSON, M.T., Interatomic Potentials and the Simulation of Lattice Defects (BEELER, J.R., GEHLEN, P.C., Eds), Battelle Colloq., Harrison Hot Springs, B.C. (1971).

[6] TSAI, D.H., BULLOUGH, R., PERRIN, R.C., J. Phys. C, Solid St. Phys. 3 (1970) 2022.

[7] ROBINSON, M.T., OEN, O.S., Appl. Phys. Lett. 2 (1963) 30.

[8] ROBINSON, M.T., Interatomic Potentials and the Simulation of Lattice Defects (BEELER, J.R., GEHLEN, P.C., Eds), Battelle Colloq., Harrison Hot Springs, B.C. (1971).

[9] TSAI, D.H., BECKETT, C.W., J. geophys. Res. 71 (1966) 2601; Proc. Behaviour of Dense Media under High Dynamic Pressure, IUTAM, Paris, Sep. 1967, Gordon and Breach, New York (1967) 99.

[10] VERLET, L., Phys. Rev. 159 (1967) 98.

[11] CHANG, R., Phil. Mag. 16 (1968) 1021; Proc. Conf. Calculation of the Properties of Vacancies and Interstitials, Misc. Publ. 287, National Bureau of Standards, Washington, D.C. (1966).

[12] BULLOUGH, R., PERRIN, R.C., Proc. R. Soc. A305 (1968) 541.

[13] PERRIN, R.C., ENGLERT, A., BULLOUGH, R., Interatomic Potentials and the Simulation of Lattice Defects (BEELER, J.R., GEHLEN, P.C., Eds), Battelle Colloq., Harrison Hot Springs, B.C. (1971).

[14] EYRE, B.L., BARTLETT, A.F., Phil. Mag. 12 (1965) 261.

[15] MASTERS, B.C., C.E.G.B. Rep. (B), N245 (1964).

[16] JOHNSON, R.A., Phys. Rev. 134 (1964) A1329; 145 (1966) 423.

[17] ERGINSOY, C., VINEYARD, G.H., ENGLERT, A., Phys. Rev. 133 (1964) A595.

[18] GIBSON, J.B., GOLAND, A.N., MILGRAM, M., VINEYARD, G.H., Phys. Rev. 120 (1960) 1229.

[19] BULLOUGH, R., HARDY, J.R., Phil. Mag. 17 (1968) 833.

[20] FLOCKEN, J.W., HARDY, J.R., Phys. Rev. 175 (1968) 919.

[21] KANZAKI, H., J. Phys. Chem. Solids 2 (1957) 24.

[22] ENGLERT, A., TOMPA, H., Tech. Rep. E.R.A., Ref. 16/99, Union Carbide Corp. (1969).

[23] SINHA, S.K., Phys. Rev. 143 (1966) 422.

[24] HARRISON, W., Interatomic Potentials and the Simulation of Lattice Defects (BEELER, J.R., GEHLEN, P.C., Eds), Battelle Colloq., Harrison Hot Springs, B.C. (1971).

[25] COTTRELL, A.H., Dislocations and Plastic Flow in Crystals, Clarendon Press, Oxford (1953).
[26] PARSON, J.R., Interatomic Potentials and the Simulation of Lattice Defects (BEELER, J.R., GEHLEN, P.C., Eds), Battelle Colloq., Harrison Hot Springs, B.C. (1971).
[27] BULLOUGH, R., NEWMAN, R.C., Rep. Prog. Phys. 33 2 (1970) 101.
[28] HEAD, A.K., Aust. J. Phys. 20 (1967) 557.
[29] MAHER, D.M., PERRIN, R.C., BULLOUGH, R., Phys. Status Solidi (b) 43 (1971) 707.
[30] BULLOUGH, R., MAHER, D.M., PERRIN, R.C., Phys. Status Solidi (b) 43 (1971) 689.
[31] HOWIE, A., WHELAN, M.J., Proc. R. Soc. A263 (1961) 217; A267 (1962) 206.

THE QUANTUM COMPUTATIONAL PHYSICS OF ATOMS AND MOLECULES

R. K. NESBET
IBM Research Laboratory,
San José, Calif.,
United States of America

Abstract

THE QUANTUM COMPUTATIONAL PHYSICS OF ATOMS AND MOLECULES.

This paper surveys computational methods currently used in applications of quantum theory to the structure and physical properties of atoms and molecules. Emphasis is placed on applications that require large-scale computation or that involve a major effort of organization or programming. Specific applications and algorithms are discussed for three major computational areas: systems of integro-differential equations, matrix methods, and symmetry algebra. To illustrate the criteria used in development of new methods for large-scale computation, a case study is given of a computer program for electron-atom scattering calculations.

1. INTRODUCTION

Purpose of atomic and molecular calculations

In atomic and molecular physics, well-known and understood theory is being applied to complex systems involving many interacting electrons and nuclei. The validity of the underlying Schrödinger or Dirac quantum mechanics is not in doubt. However, the precise experimental implications of the theory can be very unclear, requiring a logical chain of approximative assumptions. The ability to do quantitative or even qualitatively correct calculations can be very helpful in such circumstances. In practice, both qualitative and quantitative calculations are of value. Some general purposes served by this work, with an illustrative example in each case, will be given here.

a. Qualitative or model calculations

Any proposed new refinement of theory or of computational method must be tested on simple but realistic problems before it can be used with confidence. For example, low-energy electron scattering by the hydrogen atom is currently being used as a test of formal methods, because results can be compared with the very accurate variational calculations of Schwartz and others [1].

Theoretical calculations can be very helpful in testing the qualitative validity of proposed explanations of new or unexpected experimental observations. Recently, the experimental technique and formal analysis of ion-atom scattering data has advanced to the point that perturbations appearing in the data can be attributed to specific crossings of adiabatic potential surfaces [2]. Ab-initio calculations of these potential surfaces, while not yet of quantitative accuracy comparable to the experimental data,

are nevertheless sufficient to verify the proposed qualitative assignment of perturbations to crossings between specific molecular energy states.

Before definitive experiments can be carried out, or in some cases, even designed, predictions of new phenomena can be tested by qualitative calculations. A recent example is the proposal [3] that high-temperature superconduction might occur in long-chain molecules with a specific structure. This hypothesis is being tested by semi-empirical calculations, which will either indicate the unlikelihood of the proposed phenomenon, or suggest specific molecular systems to be synthesized for experiment [4].

b. Quantitative calculations

Experimental data of intrinsically very high precision may be subject to misidentification. This happens particularly in spectroscopy, where many different transitions can occur with similar intensity in a given frequency range. Unless enough data are available to provide internal cross-checks, the assignment of a given transition can be ambiguous. In the case of the very common and thoroughly studied molecule CO, systematic calculations of valence and Rydberg electronic excited states indicated that a particular state had been misidentified [5]. When transitions to this state were subsequently observed under higher resolution than had previously been possible, the proposed new identification was confirmed [6].

It is often impossible, for detailed experimental reasons, to make direct measurements of data of fundamental scientific or technological interest. Calculations of high quantitative accuracy are required to obtain such data for atoms and molecules. For example, direct measurements of nuclear quadrupole moments are not possible, because laboratory fields of quadrupolar symmetry are too weak to compete with the extremely strong internal electrostatic fields of atoms and molecules. Internal field strengths are of the order of volts per ångström, or 10^8 V per centimetre. Only the product of a nuclear quadrupole moment with the electric field gradient due to its electronic environment can be measured directly. To make use of these data, accurate calculations of electric field gradients in atoms and molecules are required. Recent developments of theory and computational procedure indicate that calculations of sufficient accuracy have become possible for light atoms [7].

Chemical and physical phenomena affecting atoms and molecules can be so complex that simplifying assumptions must be introduced in any attempt to interpret experimental data. Such assumptions can often be tested by quantitative calculations on simple molecules or atoms, when a direct experimental test is not possible. A situation of this kind occurs in the theory of the condensed phases of rare-gas systems. A parametrized two-body potential energy function must be deduced from physical and thermodynamic properties of condensed phases and mixed gases. Perturbation theory can be used to indicate the functional form of the long-range attractive potential, but the short-range repulsive potential can only be obtained by ab-initio diatomic molecule calculations of adequate accuracy to avoid spurious results. The range of choices for parametrized functional forms was greatly reduced when such calculations on mixed rare-gas diatomic systems showed that the repulsive potential was of nearly pure exponential form over several logarithmic decades of magnitude [8].

Applications of quantum theory to electronic properties of large molecules can be made only if approximations are introduced. For this purpose it is necessary to have systematic and accurate data for the simplest systems, atoms and diatomic molecules. These data can then be combined in "semi-empirical" calculations on large molecules. Electronic correlation energies, defined as energy corrections to the Hartree-Fock or independent-electron approximation, have been found to be very useful in this context. Since the correlation energy is a purely theoretical concept, not subject to experimental measurement, such systematic data can only be obtained by computation. Data of this kind were obtained by systematic Hartree-Fock calculations on atoms and ions of the first third of the periodic table [9]. More recently, direct calculations of electron pair-correlation energies for the ground states of atoms up to argon have been carried out [10].

Computer usage and requirements

Computations in atomic and molecular physics make use of the full range of facilities of modern computer systems. The specific requirements will be discussed separately here for two aspects of this work: development of algorithms and production calculations. An effective hardware and software system must meet these often complementary sets of requirements concurrently, without degrading either usage, so that new methods can be developed while production work on specific applications goes forward.

a. Development of algorithms

Access to a specialized application program package is essential. The structure of computational procedures is often very complex, involving several different stages of calculation, each of a different mathematical character. It is important to make as much use as possible of an existing package of modular programs when testing a new method that affects only one major stage of an overall computation. Otherwise, the task of rewriting an entire set of procedures, debugging each step, could easily be an insuperable obstacle to any serious innovation.

To implement this requirement, there must be maintained on-line (at least, as seen by the user) a library of modular specialized application programs. A simple updating procedure with temporary update capability is required, so that tested production programs can be used freely with new or temporarily updated program modules for testing changes and innovations.

Rapid turnaround is also essential. This requirement must be satisfied for work that makes use of a specialized on-line program package, as described above, subject of course to the obvious upper limits on central processing unit (CPU) time that distinguish test and debugging runs from production work. A computer system is inadequate if it can provide rapid turnaround only for trivial work, but cannot at the same time provide for updating of the complex program packages needed in large-scale production calculations. Why should any test run turnaround time (from input to test results available for examination) be greater than the sum of real time requirements for reading input, library access, CPU utilization, and

writing output? Any great excess over this sum must be attributed to an unbalanced, overloaded, or inadequately designed or managed hardware and software system.

Failure to achieve efficient turnaround becomes a powerful incentive against innovative work, since the level of concentration needed to create or modify complex methods and programs is very high. The practical effect of this failure must be to degrade the overall efficiency of a computer installation, since without flexible testing and innovation, errors and inefficient procedures in production programs can too easily be perpetuated.

b. Production calculations

A very important requirement for production calculations in atomic and molecular physics is the ability to handle scratch data lists of sometimes enormous size. For example, eigenvalues of matrices with several million independent elements are required in practical calculations. Such data lists cannot be contained in the high-speed memory of any existing or foreseeable computer. As hardware memory capacity has increased, the requirements for such scratch data storage have increased much more rapidly. Hence a fundamental requirement is for large on-line auxiliary memory capacity together with adequate data transfer rate (concurrent with CPU processing) to keep the CPU functioning efficiently.

Atomic and molecular computations can make use of the largest available high-speed memories. If adequate scratch storage is available, with adequate data transfer rate, it is often possible to devise computational algorithms in which the data flow efficiently through a buffer system. This can greatly reduce the maximum demand for high-speed memory capacity, since the latter is used primarily as a buffer, and not for passive storage of entire data lists. Unless such a computation is to become hopelessly input/output (I/O) bound, the software system must provide for concurrent I/O scheduling by the application program.

Large-scale computations clearly benefit from increased CPU speed. Some of the work considered here has only become feasible within the last decade because of advances in computer technology. However, because of continuing refinements of methodology and computing technique, the magnitude of problem undertaken has tended to grow even faster than the computer hardware capability.

For obvious practical reasons, computational efficiency is an extremely important criterion for methods and programs used for large-scale computations. Since compiled programs in modular packages tend to be used over and over again in different production runs, the only important criterion for compilers in this context is the efficiency of the compiled code. This requirement is exactly complementary to that for compilers in test runs, where the important criterion is turnaround time and not efficient execution of the compiled test code.

Criteria for effective methods

Progress toward developing methods capable of obtaining valid and useful results in atomic and molecular computations has required innovations in pure theory, in applied mathematics, and in the organization of computer programs. The most effective methods have been designed with

the help of all three of these disciplines. To be of use, computational methods must satisfy criteria of efficient and accurate pure computation, of efficient data handling, and of ability to produce physically meaningful results.

2. SURVEY OF APPLICATIONS

Despite the wide variety of computational techniques used for different specific problems in this field, the fundamental problem in each case is to obtain an approximate solution of the many-particle Schrödinger equation. For stationary states, this is a linear eigenvalue equation of the form

$$H\Phi_\alpha = E_\alpha \Phi_\alpha \tag{1}$$

where H is a differential operator depending on space and spin co-ordinates of all nuclei and electrons in a given atom or molecule. For bound states, Φ_α is normalizable (quadratically integrable) over an infinite spatial volume in all co-ordinates, but in scattering theory, boundary conditions require a specific oscillatory asymptotic functional form, and the wave function is not normalizable. To consider the interaction with radiation, the time-dependent Schrödinger equation must be used

$$H\Psi = i\hbar \frac{\partial}{\partial t} \Psi \tag{2}$$

where the wave function is expanded as a superposition of stationary states

$$\Psi(t) = \sum_\alpha \Phi_\alpha c_\alpha(t) \exp(E_\alpha t/i\hbar) \tag{3}$$

The slowly-varying coefficients $c_\alpha(t)$ are obtained by standard time-dependent perturbation theory, and the principal computational problem is the evaluation of the eigenfunctions Φ_α.

The most important areas of application will be described briefly here, with an indication of the computational methods used in each case. Three main classes of computational problems can be abstracted from this survey. These are, respectively, systems of integro-differential equations, matrix equations, and symmetry algebra. Details of the most effective algorithms used for these three classes of problems will be given in following sections.

Nuclear co-ordinates

a. Rotation

The rotational wave functions for molecules can be expressed in terms of the irreducible representation matrix elements of the three-dimensional rotation group, given as functions of the Euler angles that parametrize

a rotation. Electronic and nuclear spin can be taken into account by standard vector-coupling methods, which are applications of the symmetry algebra of the rotation group. Matrix elements required for the computation of rotational energy levels or of transition probabilities can be expressed in terms of the n-j symbols of angular momentum theory [11].

b. Vibration

Analysis of the vibrational wave functions of molecules is greatly simplified by the use of normal modes, which diagonalize the quadratic form of internuclear potential energy expanded about an equilibrium nuclear conformation. This analysis can be simplified in conformations possessing point-group symmetry, by use of group representation theory. In its simplest form, for a single normal mode or for a diatomic molecule, the vibrational Schrödinger equation is an ordinary differential eigenvalue equation. This presents no important computational problem.

To provide simple parametric formulas for vibrational energy levels, special functional forms, such as the Morse potential, are often introduced as approximations to vibrational potential functions. The semi-classical approximation (WKB method) can be used, because of the large effective nuclear mass, to derive very accurate potential functions from observed values of vibrational energy levels [12].

c. Scattering and reactions

The quantum theory of atomic or molecular collisions requires continuum solutions of the effective Schrödinger equation for the nuclear co-ordinates of the combined system formed by collision. Inelastic scattering occurs by rotational or vibrational excitation of the colliding species, and chemical reactions occur when structural rearrangements are possible during the collision. An added complication, of great experimental interest, occurs when the internuclear potential energy hypersurfaces corresponding to different electronic states become degenerate for some nuclear conformation. Then, because of this curve-crossing, changes of electronic state as well as vibrational or rotational excitation can contribute to inelastic scattering and to chemical reactions.

For scattering on a single potential surface, the relatively large nuclear mass makes it possible to use semi-classical approximations, associating a Lagrangian phase integral with each possible classical trajectory, and then considering the interference between an ensemble of trajectory wave functions. This approach justifies such simplifications as considering only the dynamics of a classical reaction path, along which the potential function is stationary with respect to all but one normal mode of nuclear motion. Curve-crossing can be taken into account by the Landau-Zener formula [13], an application of the semi-classical approximation.

A full quantum-mechanical treatment of inelastic or reactive scattering requires a very great computational effort. Even for scattering at thermal energies, rotational excitation of the colliding species must be taken into account. The problem is inherently a many-channel problem, requiring the solution of a system of coupled differential equations [14].

Electronic wave functions

a. Hartree-Fock calculations

The simplest N-particle function that can have the same symmetry properties as a true electronic stationary state wave function is a Slater determinant, or antisymmetrized product of one-electron wave functions. The latter are usually referred to as orbital wave functions or simply as orbitals. If a stationary state wave function is approximated by a single Slater determinant, the Hartree-Fock variational equations for the orbitals are a system of coupled non-linear integro-differential equations. For atoms, because of spherical symmetry, only a single independent variable occurs in each equation, and the system of equations can be integrated by methods to be described below [15]. For molecules, each equation contains two or more independent variables, and no direct method of solution of such systems of coupled partial differential equations is known. Instead, molecular orbitals are expanded as linear combinations of basis functions centred on the various atoms, and a system of matrix equations is solved [16]. As the basis orbital set is extended to completeness, the matrix method becomes formally equivalent to integration of the corresponding Hartree-Fock equations.

Various generalizations are possible. If the variational trial function is taken to be a linear combination of Slater determinants, the resulting equations for the orbitals are similar in form to the Hartree-Fock equations, but of more complex structure. A similar remark holds for relativistic equations (Hartree-Fock-Dirac), which will not be considered in detail here.

b. Electronic correlation

The Hartree-Fock equations correspond to a very useful approximation to a solution of the many-electron Schrödinger equation, but the approximation is incapable of describing accurately the detailed interaction between electrons. In the Hartree-Fock approximation each electron interacts only with an average "self-consistent" field. The residual effect of the electronic Coulomb repulsion, known as electronic correlation, can be very important in specific applications, but requires calculations going beyond the limitations of the Hartree-Fock approximation.

The formal many-body perturbation theory derived from quantum field theory, making use of Feynman diagrams and the so-called "linked-cluster" expansion, has been successfully adapted to the electronic correlation problem for atoms [17]. The computational techniques required are similar to those for the atomic Hartree-Fock equations, since, in principle, a complete set of continuum orbitals must be generated. The specialized methods used to generate and evaluate linked-cluster diagrams have been described by Kelly [17].

A more straightforward approach to the electronic correlation problem is the matrix method known as "superposition of configurations". The wave function Φ_α of Eq.(1) is expanded as a linear combination of Slater determinants, and the coefficients in this expansion are obtained as an eigenvector of the many-electron Hamiltonian matrix in the Slater determinant basis. A complete orthonormal set of basis orbitals generates a

complete orthonormal set of Slater determinants, so this method is capable, in principle, of obtaining an arbitrarily accurate stationary-state many-electron wave function. In practice, the dimensionality of the many-electron Hamiltonian matrix increases so rapidly with the number of electrons and with the size of the orbital basis set actually used (always finite), that the method cannot be used except with simplifying procedures such as matrix perturbation theory. In this method, matrix elements must be evaluated as definite integrals, but no systems of differential equations are solved. Hence the method is formally the same for atoms and molecules, but the task of evaluating the necessary integrals is significantly more difficult for molecules.

A systematic procedure has been developed that makes use of the matrix technique of superposition of configurations, but shares with perturbation theory the property of being a step-by-step procedure that eventually converges to exact results. This method makes use of the formal mathematical structure of a lattice decomposition of the many-electron Hilbert space, represented in a basis of Slater determinants. The method, which will be described in more detail in a following section, represents a generalization of the independent-pair model introduced by Brueckner in nuclear many-body theory [18]. A hierarchy of variational equations is used, equivalent at the two-particle level to the Bethe-Goldstone equation of nuclear theory, adapted to take advantage of the ordinary Hartree-Fock approximation as the zeroth level of the hierarchy [19].

c. Electron scattering

The wave function for electron scattering by an N-electron atom or molecule has the general form

$$\Psi = \sum_{p} \mathscr{A}\, \Theta_p \psi_p + \sum_{\mu} \Phi_\mu\, c_\mu \qquad (4)$$

Here Θ_p is the N-electron wave function for a stationary state of the target atom or molecule; ψ_p is the continuum orbital for an electron in the open scattering channel defined by Θ_p; $\mathscr{A}$ is an antisymmetrizing operator; and the functions Φ_μ are normalizable N+1-electron functions that in principle form a complete orthonormal set. For low-energy scattering, the asymptotic potential function, except for Coulomb and centrifugal terms, is dominated by a polarization potential due to electron correlation. Both of the two types of terms in Eq.(4) are needed to describe this effect. Hence electronic correlation must be taken into account from the outset in scattering theory.

In the close-coupling method [20], the second expansion in Eq.(4) is truncated to a very small number of terms, written in the form of the first expansion, but with normalizable "closed-channel" orbitals ψ_q, antisymmetrized into "pseudostate" polarization functions Θ_q. The variational equations for the functions ψ_q and the open-channel orbitals ψ_p are then solved as a system of coupled integro-differential equations. This method is especially suitable for electron-hydrogen scattering, since the

target eigenfunctions Θ_p are known explicitly, but the method has also been applied to heavier atoms with appropriate approximations.

In an attempt to extend accurate scattering calculations to heavier atoms, and eventually to molecules, linear expansion methods have recently been applied to the electron-atom scattering problem. As in bound state calculations, systems of matrix equations must be solved, but the principal computational step is the evaluation of matrix elements of the electronic Hamiltonian operator, expressed as definite integrals. The integrands contain continuum orbitals and the integrals are considerably more difficult to compute than in the case of bound states. A case study of the development and implementation of this method will be given in the section below.

Interaction with perturbing fields

a. Static fields

The interaction of an atom or molecule with external perturbing fields can always be treated by perturbation theory. First-order interactions are governed by static multipole moments, computed as mean values of the appropriate operators in a statistical ensemble of stationary states, usually dominated by the electronic ground state. Such results are obtained as a byproduct of calculations of electronic stationary states.

Second-order interactions are governed by generalized polarizabilities. Computations require evaluation of the first-order perturbed wave function for a given perturbing field, as well as of ground or low-lying excited electronic stationary states. The specialized computational procedures available for polarizability calculations are similar to those used for stationary states, and will not be described in detail here.

The non-Coulombic interaction with nuclear magnetic and electric quadrupole moments is responsible for hyperfine structure in atomic and molecular spectra. Computations are similar to those for static multipole moments.

b. Dynamic fields – radiation

Relatively little accurate work has been done in the field of radiative transition rates. An accurate representation, including correlation effects, of both initial and final state is needed. Calculations require essentially the same techniques as stationary state calculations. Some recent work in this area is given in Ref.[21].

The cross-section for coherent scattering of radiation (Rayleigh scattering) is governed by a frequency-dependent dynamic polarizability, which is computed in a manner similar to the static polarizability. The mixed electric and magnetic dipole dynamic polarizability governs optical activity. Accurate calculations of optical activity have not yet been carried out because of the difficulties of such calculations for molecules.

The transition rate or cross-section for incoherent scattering of radiation (Raman scattering) also requires computation of a dynamic polarizability depending on two different frequencies, for the incident and scattered radiation. Few ab-initio calculations of Raman intensities have been carried out, again because of the difficulty of molecular calculations.

In the high-energy limit of scattering, either of radiation or of electrons, the Born approximation becomes valid, and elastic scattering cross-sections can be expressed in terms of matrix elements involving only the electronic ground state. Inelastic scattering requires a sum of second-order matrix elements, analogous to that occurring in polarizability calculations. If this sum is simplified by a closure approximation, the total scattering cross-section can be expressed in terms of correlation properties of the electronic ground-state wave function [22]. Few accurate calculations of X-ray scattering or of high-energy electron diffraction have been carried out, and this work will not be discussed in detail here.

3. SYSTEMS OF INTEGRO-DIFFERENTIAL EQUATIONS

Such systems of equations arise in variational approximations to the solution of the stationary state equation

$$(H-E)\,\Psi = 0 \tag{5}$$

For bound states, this equation follows from requiring the variational functional

$$I = (\Psi|H-E|\Psi) \tag{6}$$

to be stationary. Then, since H is Hermitian,

$$(\delta\Psi|H-E|\Psi) = 0 \tag{7}$$

which implies Eq.(5) if $\delta\Psi$ and Ψ vanish for infinite values of the particle co-ordinates.

The Hamiltonian operator is of the form

$$H = H_A(a_1\ldots) + H_B(b_1\ldots) + H_{AB}(a_1\ldots;\ b_1\ldots) \tag{8}$$

where the subsets of co-ordinates are chosen differently for different purposes.

a. Born-Oppenheimer expansion

If the co-ordinates {b} in Eq.(8) are taken to be the nuclear co-ordinates of a molecule, and {a} to be the electronic co-ordinates, a stationary-state molecular wave function can be expressed as

$$\Psi_\beta = \sum_\alpha \Psi_{A\alpha}(b;a)\,\Psi_{B\alpha,\beta}(b) \tag{9}$$

where for each nuclear conformation there is a complete orthonormal set of electronic wave functions satisfying equations

$$\{H_A + H_{AB} - E_\alpha(b)\}\ \Psi_{A\alpha}(b;a) = 0 \tag{10}$$

The parametric dependence of the electronic wave function for energy level α on the nuclear co-ordinates is indicated. The energy functions $E_\alpha(b)$ serve as potential energy hypersurfaces for the nuclear motion.

The nuclear wave functions $\Psi_{B\alpha,\beta}$ are determined by a coupled system of differential equations. These equations follow from the variational principle, Eq.(7), in the form

$$(\delta_B \Psi_\beta | H - E_\beta | \Psi_\beta) = 0$$

$$\Rightarrow (\Psi_{A\alpha} | H - E_\beta | \Psi_\beta) = 0, \text{ all } \alpha \tag{11}$$

$$\Rightarrow \{H_B(b) + X_{\alpha\alpha}(b) + E_\alpha(b) - E_\beta\} \Psi_{B\alpha,\beta}(b) = -\sum_{\alpha' \neq \alpha} X_{\alpha\alpha'}(b) \Psi_{B\alpha',\beta}(b)$$

where

$$X_{\alpha\alpha'}(b) = (\Psi_{A\alpha} | H_B \Psi_{A\alpha'} - \Psi_{A\alpha'} H_B)_a = (\Psi_{A\alpha} | [T_B, \Psi_{A\alpha'}])_a \tag{12}$$

The operator T_B is the nuclear kinetic energy operator, which acts on the parametric nuclear co-ordinates in the electronic wave function as indicated here. Since the ratio of electronic to nuclear masses is very small, the functions $X_{\alpha\alpha'}(b)$ can often be neglected, leading to the widely used adiabatic approximation [23]. The significance of curve-crossing is clear from the structure of Eqs (11), since the coupling terms $X_{\alpha\alpha'}$ cannot be neglected when two potential hypersurfaces E_α and $E_{\alpha'}$ intersect. This can invalidate the adiabatic approximation for atom-atom or molecular scattering.

Even in the adiabatic approximation, when Eq.(11) reduces to a differential equation in the nuclear co-ordinates, a further separation of rotational and vibrational co-ordinates leads to a system of coupled differential equations. An example of this is discussed, in detail, in the paper by Lester in these Proceedings [14] who considered inelastic scattering of Li^+ by the H_2 molecule.

In the electronic and vibrational ground state of H_2, rotational excitation is possible, and an expansion in rotational wave functions of the molecule leads to coupled equations in the two remaining co-ordinates: r for the displacement of Li^+ from the centre of mass of H_2, and the angle ψ between vector $\vec{r}$ and the H_2 molecular axis. The energy surface in electronic state α is expanded in spherical harmonics

$$E_\alpha(R, r, \psi) = \sum_\ell v_\ell(R, r) P_\ell(\cos\psi) \tag{13}$$

where R is the H_2 interatomic distance. The coefficients $v_\ell(R, r)$ are replaced by mean values in the H_2 vibrational ground state

$$v_\ell(r) = (0 | v_\ell(R, r) | 0) \cong v_\ell(R_e, r) \tag{14}$$

and the wave function for nuclear motion is approximated by an expansion of the form

$$\Psi_{B\alpha} \cong \sum_{\gamma} u_{\gamma}(r)\, Y_{\gamma}(\theta, \phi) \tag{15}$$

Here θ, ϕ specify the orientation of the H_2 axis in a space-fixed frame of reference, and γ symbolizes the angular quantum numbers. With these approximations and expansions,

$$(\gamma' \,|E_{A\alpha}(b) - E\,|\gamma) = \sum_{\ell} (\gamma' \,|\ell\,|\gamma)\, u_{\gamma}(r) - \tfrac{1}{2}k^2 \delta_{\gamma',\gamma} \tag{16}$$

and the variational equations reduce to the form

$$(T - \tfrac{1}{2}k_{\gamma}^2)\, u_{\gamma}(r) = -\sum_{\ell} (\gamma' \,|\ell\,|\gamma)\, u_{\gamma'}(r) \tag{17}$$

where T is an effective kinetic energy operator.

b. Close-coupling equations

Equations (17) are typical of close-coupling equations, in this example a system of differential equations. For open scattering channels the boundary conditions are

$$r u_{\gamma}(r) \to 0, \quad r \to 0$$

$$u_{\gamma}(r) \sim \alpha_{0\gamma} \sin(k_{\gamma} r) + \alpha_{1\gamma} \cos(k_{\gamma} r), \quad r \to \infty \tag{18}$$

Elastic and inelastic cross-sections are determined by the asymptotic coefficients α_0, α_1.

The method used by Lester [14] to integrate the close-coupling equations is that proposed by de Vogelaere [24], which will not be discussed in detail here. In a recent review of numerical methods applicable to such systems of equations, Allison [25] advocates the use of an iterative matrix version of the Numerov method, which will be described in more detail below.

A new method of great promise has recently been developed by Gordon [26]. In scattering solutions of the Schrödinger equation, a relatively smooth potential function leads characteristically to an oscillatory wave function. Standard numerical methods attempt to construct a smooth fit to the wave function, point by point. For a rapidly oscillating function, many points are required. Gordon proposes to construct a smooth fit to the potential by a piecewise linear function. Then the wave function is given as an exact solution over each piecewise interval, expressed in terms

of Airy functions. This method can lead to a very great reduction in the number of integration points required for given accuracy. The method can be applied to coupled equations as well as to a single equation. The analysis needed to match Airy functions at boundary points has been given by Gordon [26].

These methods can all be applied to the bound-state eigenvalue problem. A computer program using the Numerov method and an iterative method to match inward and outward integration and determine the eigenvalue has been distributed through SHARE by Cooley [27].

c. The Numerov method

The most widely used method for differential equations of the Schrödinger form for y(x),

$$y'' + fy = g \tag{19}$$

with no explicit first derivative, is the method of Numerov [28]. Expansion of the differential equation by central differences gives the recurrence formula for interval $\Delta x = h$,

$$\left(1 + \frac{1}{12} h^2 f_{n+1}\right) y_{n+1} - \left(2 - \frac{10}{12} h^2 f_n\right) y_n + \left(1 + \frac{1}{12} h^2 f_{n-1}\right) y_{n-1}$$

$$= \frac{1}{12} h^2 (g_{n+1} + 10 g_n + g_{n-1}) + \text{error term} \tag{20}$$

The error term is

$$- \frac{1}{240} \delta^6 y_n + \ldots \tag{21}$$

In the Numerov method, Eq.(20) is solved for y_{n+1} for a step-by-step solution. If Eq.(20) is a matrix equation, this requires inversion of the matrix $[1 + (h^2/12) f_{n+1}]$ at each step. The iterative procedure of Allison [25] is designed to simplify this inversion. Since Eq.(20) is of the form of a recurrence relation, it can be extended as a tridiagonal matrix equation coupling a sequence of integration points. Direct solution of the resulting matrix equations, by methods such as Gauss elimination, is used in practical applications to avoid numerical instability when y(x) is an exponentially decreasing function [29]. The Numerov method requires separate procedures to start the integration from a boundary point and to change mesh size. The de Vogelaere method has a larger error term but avoids these special procedures [24].

When the Numerov method is used in an iterative procedure, as in eigenvalue problems or solution of Hartree-Fock equations, the error term can be estimated from the previous iteration and used to increase the accuracy of the integration [30]. For this purpose, the error term is used in the form

$$h^2 \left(- \frac{1}{240} \delta^4 + \frac{31}{60\,480} \delta^6 - \ldots\right) y_n'' \tag{22}$$

Since y'' is given as g-fy from Eq.(19), this requires only the central fourth-difference of the product of the integrated function and the given function f.

d. Hartree-Fock equations

In the simplest case, valid for closed-shell ground states, the N-electron Hartree-Fock wave function is a single normalized Slater determinant

$$\Phi_0 = \det \phi_1 \ldots \phi_N$$

$$= (N!)^{-\frac{1}{2}} \begin{vmatrix} \phi_1(1) & \ldots & \\ \ldots & \phi_i(j) & \ldots \\ & \ldots & \phi_N(N) \end{vmatrix} \tag{23}$$

The variational Eq.(7) in this case is

$$(\delta\ \Phi_0 | H-E | \Phi_0) = 0 \tag{24}$$

which implies a system of integro-differential equations with the canonical form

$$\mathscr{H}_0 \phi_i = \epsilon_i \phi_i \qquad i = 1, \ldots, N \tag{25}$$

For a many-electron Hamiltonian operator of the form

$$H = \sum_i K(i) + \sum_{ij} Q(ij) \tag{26}$$

the effective one-electron Hamiltonian operator is

$$\mathscr{H}_0 = K + \sum_{i=1}^{N} (i|Q-QP|i) \tag{27}$$

summed over occupied orbitals of Φ_0, where

$$\begin{aligned} (i|Q|i)\,\phi_j &= \int \phi_i^*(1)\,\phi_i(1)\,r_{12}^{-1}\,d\tau_1\,\phi_j(2) \\ (i|QP|i)\,\phi_j &= \int \phi_i^*(1)\,\phi_j(1)\,r_{12}^{-1}\,d\tau_1\,\phi_i(2) \end{aligned} \tag{28}$$

The second equation here defines an exchange potential in terms of a linear integral operator. Exchange is treated as an inhomogeneous term in integrating the Hartree-Fock equations.

For general electronic configurations, the system of Hartree-Fock equations cannot be reduced to the uncoupled form indicated in Eq.(25), since a matrix of Lagrange multipliers ϵ_{ij} couples the equations for different orbitals. For an atom, in the nonrelativistic approximation, angular integrations can be carried out explicitly to give a coupled system of radial equations of the form

$$\left[\frac{d^2}{dr^2} + \frac{2}{r} Y(n\ell; r) - \epsilon_{n\ell, n\ell} - \frac{\ell(\ell+1)}{r^2}\right] P_{n\ell}(r)$$

$$= X(n\ell; r) + \sum_{n' \neq n} \epsilon_{n\ell, n'\ell} P_{n'\ell}(r) \qquad (29)$$

The function $P_{n\ell}(r)$ is r times the radial factor of an occupied orbital. The angular factor is a spherical harmonic with angular momentum quantum numbers ℓ, m_ℓ. Index n is the usual principal quantum number. Electron spin is included formally as a factor in the orbital function, with spin quantum numbers $s = \frac{1}{2}$, m_s.

The methods used to integrate Eqs (29) have been reviewed in detail by Hartree [31] who emphasizes methods used before the introduction of modern digital computers. More recent work, adapted to computers, is described by Froese-Fischer [29], who has published a description of a program using the Numerov method [32].

The functions Y and X in Eqs (29) are sums of potential energy terms derived from Eqs (28) by integrating over angular and spin variables. The interelectronic Coulomb potential is expanded in the well-known series

$$\frac{1}{r_{12}} = \sum_k \frac{4\pi}{2k+1} \frac{r_<^k}{r_>^{k+1}} \sum_{s=-k}^{+k} Y_k^{s*}(\theta_1, \phi_1) Y_k^s(\theta_2, \phi_2) \qquad (30)$$

where $r_>$, $r_<$ denote, respectively, the greater or lesser of r_1, r_2. All contributions to the potential energy terms are of the form

$$Y_k(n\ell, n'\ell'; r) = \int_0^\infty r\, U_k(r, s)\, P_{n\ell}(s)\, P_{n'\ell'}(s)\, ds \qquad (31)$$

where

$$rU_k = (s/r)^k, \quad s < r \qquad (32)$$

$$= (r/s)^{k+1}, \quad s > r \qquad (33)$$

To evaluate Eq.(31), it is convenient to define the function

$$Z_k(n\ell, n'\ell'; r) = \int_0^r (s/r)^k P_{n\ell}(s) P_{n'\ell'}(s)\, ds \qquad (34)$$

First Z_k is obtained by outward integration of the differential equation

$$\frac{d}{dr} Z_k = -\frac{k}{r} Z_k + P_{n\ell}(r) P_{n'\ell'}(r) \tag{35}$$

starting from $Z_k(0) = 0$. Then Y_k is obtained by inward integration of

$$\frac{d}{dr} Y_k = \frac{k+1}{r} Y_k - \frac{2k+1}{r} Z_k \tag{36}$$

starting from

$$Y_k - Z_k \sim 0 \text{ as } r \to \infty \tag{37}$$

This two-step procedure is more stable numerically than direct integration of the equivalent equation

$$\frac{d^2}{dr^2} Y_k = \frac{k(k+1)}{r^2} Y_k - \frac{2k+1}{r} P_{n\ell}(r) P_{n'\ell'}(r) \tag{38}$$

In a computer program for Hartree-Fock equations [29, 32], changes of the integration step size can be avoided by using $\rho = \ln r$ as independent variable. Then an equation of the form

$$\left\{ \frac{d^2}{dr^2} + \left[2v - \epsilon - \frac{\ell(\ell+1)}{r^2} \right] \right\} P = X, \quad 0 < r < \infty \tag{39}$$

becomes

$$\left\{ \frac{d^2}{d\rho^2} + \left[2r^2 v - \epsilon r^2 - \left(\ell + \frac{1}{2} \right)^2 \right] \right\} (r^{-\frac{1}{2}} P) = r^{\frac{3}{2}} X, \quad -\infty < \rho < \infty \tag{40}$$

The matrix Numerov method is used to integrate the close-coupling equations for electron-atom scattering [20, 33], similar in form to the Hartree-Fock equations, but including continuum orbitals for open scattering channels.

4. MATRIX METHODS

If an electronic bound-state wave function is expanded in the form

$$\Psi = \sum_{\mu} \Phi_{\mu} c_{\mu} \tag{41}$$

i. e. in a linear combination of Slater determinants

$$\Phi_{\mu} = \det(\phi_{1\mu} \ldots \phi_{i\mu} \ldots \phi_{N\mu}) \tag{42}$$

the usual variational principle leads to matrix eigenvalue equations for the expansion coefficients,

$$\sum_{\nu} (H_{\mu\nu} - E\,\delta_{\mu\nu})\, c_{\nu} = 0, \quad \text{all } \mu \tag{43}$$

If the many-electron Hamiltonian operator is given by Eq. (26), the matrix elements in Eqs (43) are of the form

$$H_{\mu\nu} = \text{l.c.}\ \{(p|K|q)\ ;\ (pq|Q|rs)\} \tag{44}$$

where l.c. denotes "linear combination".

The essential computational problem is to evaluate the matrix elements $H_{\mu\nu}$ as linear combinations of definite integrals, and then to obtain eigenvalues and eigenvectors of the Hamiltonian matrix. For molecules, the orbitals $\phi_{i\mu}$ are expressed as linear combinations of basis orbitals chosen to simplify the multi-dimensional quadrature problem involved in evaluating the "electrostatic integrals" $(pq|Q|rs)$. For atoms, the orbitals $\phi_{i\mu}$ can be taken to be some convenient complete orthonormal set, or, alternatively, can be expanded as linear combinations of simple basis functions.

a. Matrix Hartree-Fock method

Since the Hartree-Fock equations for a molecule are partial differential equations, they cannot be integrated by any known numerical method. For this reason, molecular Hartree-Fock orbitals are expressed as linear combinations of basis orbitals η_p,

$$\phi_i = \sum_{p} \eta_p\, c_{pi} \tag{45}$$

and the Hartree-Fock equations are solved in matrix form [34]. The same technique can be used for atoms [35]. It is widely used to provide approximate atomic Hartree-Fock orbitals for use in molecular calculations or in calculations of atomic physical properties and electronic correlation effects.

For a closed-shell ground state (single-determinant wave function), the variational equation, Eq. (24), leads to the matrix Hartree-Fock equations,

$$\sum_{q} [(p|\mathscr{H}_0|q) - \epsilon_i\,(p|q)]\, c_{qi} = 0\ , \quad \text{all } p \tag{46}$$

The matrix elements required here are

$$(p|\mathscr{H}_0|q) = (p|K|q) + \sum_{i(\text{occ})} (pi|Q\text{-}QP|qi) \tag{47}$$

$$= (p|K|q) + \sum_r \sum_s \sum_i c_{ri}\, c_{si}\, \{[pq|rs] - [ps|qr]\} \qquad (48)$$

where the electrostatic integrals over basis orbitals are

$$[pq|rs] = \int\int \eta_p^*(1)\ \eta_q(1)\ r_{12}^{-1}\ \eta_r(2)\ \eta_s^*(2)\ d\tau_1\, d\tau_2 \qquad (49)$$

The same integrals occur in both Coulomb and exchange terms in the matrix method. The operator $\mathscr{H}_0$ is defined by Eq. (47). Equation (46) is solved by iteration of matrix eigenvalue calculations.

If N_b linearly independent basis orbitals are used in a calculation, the number of electrostatic integrals $[pq|rs]$ is proportional to N_b^4. All of the possible simplifications of Eq. (49), using symmetry algebra, spin reduction, and special co-ordinate systems to simplify the six-dimensional spatial integration, serve only to reduce the coefficient of N_b^4 in the computing time for a given application. For molecules, the number of integrals required for a reasonably accurate expansion of Hartree-Fock orbitals is such that evaluation of the electrostatic integrals becomes the rate-determining step of any computation. It is necessary to devise methods of evaluation of these integrals that lend themselves to mass production without sacrificing numerical accuracy. Several of these methods will be discussed below.

b. Superposition of configurations

A complete orthonormal set of orbital functions generates a complete orthonormal set of N-electron Slater determinants, formed from all N-fold subsets of the basis orbitals. Starting from reference Slater determinant Φ_0, the N-electron Hilbert space is most easily described in terms of virtual excitations of this reference state determinant. Given the basis orbital set

$$\{\phi_i, \phi_a\}, \quad 1 \le i \le N < a, \qquad (50)$$

where orbitals ϕ_i are occupied in Φ_0, and orbitals ϕ_a are unoccupied, a typical virtual excitation is described by replacing n occupied orbitals i, j, ... by unoccupied orbitals a, b, ... This produces a Slater determinant

$$\Phi_{ij\ldots}^{ab\ldots} = \det \phi_1 \ldots \phi_a \ldots \phi_b \ldots \phi_N \qquad (51)$$

where occupied orbitals are replaced subject to

$$i < j < \ldots \le N < a < b < \ldots \qquad (52)$$

The N-electron Hilbert space $[\Psi]$ can be expressed as a direct sum of disjoint spaces such as (ij), obtained by including all possible virtual excitations of the indicated subset of occupied orbitals. This direct sum is

$$[\Psi] = (0) + \sum_i (i) + \sum_{ij} (ij) + \ldots + (1 \ldots N) \qquad (53)$$

In the notation used here, a disjoint space of virtual excitations of n specified occupied orbitals is denoted by

$$(ij\ldots) = \left\{ \Phi_{ij\ldots}^{ab\ldots} \right\} \tag{54}$$

where i, j, ... are fixed and a, b, ... take on all possible values subject to Eq. (52). In many-body perturbation theory, the virtual excitation $\Phi_{ij\ldots}^{ab\ldots}$ is denoted by a diagram with backward-directed (hole) lines labelled i, j, ... and forward-directed (particle) lines labelled a, b, ...

If N_b independent orbitals are used to describe an N-electron system, the total dimension of all disjoint virtual excitation spaces of order n (n indices) is the product of binomial coefficients

$$\binom{N}{n}\binom{N_b-N}{n} \sim (NN_b)^n \tag{55}$$

Although the effective coefficient can be decreased by use of symmetry algebra, this rapid increase of dimensionality with n is the fundamental practical limitation of the method of superposition of configurations.

Given an orbital basis of dimension N_b, the Hamiltonian matrix $H_{\mu\nu}$ of Eq. (43) is of linear dimension proportional to $(NN_b)^N$ if all virtual excitations are included in the variational wave function. Even for atoms, if N_b is sufficiently large to give meaningful results, calculations become impractical for N greater than four or so. A systematic way of avoiding (or, at least, of deferring) this rapid growth of dimensionality can be based on the general concept of a decomposition of the Hilbert space [Ψ] into nested subspaces,

$$[ij\ldots] = (0) + \sum_{i\subset X}(i) + \sum_{ij\subset X}(ij) + \ldots + (ij\ldots) \tag{56}$$

where the summation indices have values restricted to subsets of those present in the symbol X = [ij ...]. In words, the variational subspace [ij ...] is the direct sum of all disjoint virtual excitation subspaces whose indices form a subset of the given list ij ... . The variational subspace so defined contains Φ_0 [denoted by (0)] and all virtual excitations affecting the specified occupied orbitals ϕ_i, ϕ_{ij} ... in any combination.

The variational spaces form ordered sequences of nested subspaces. The whole subspace structure constitutes a lattice decomposition of the Hilbert space [Ψ] as indicated in Fig. 1.

As indicated in the figure, it is convenient to define

$$[0] = (0) = \{\Phi_0\} \tag{57}$$

In general, [0] is the linear space of a multi-determinant Hartree-Fock calculation that defines the zeroth level of virtual excitation.

Each entry on such a lattice diagram corresponds to an independent variational calculation. Each calculation is equivalent to solution of a

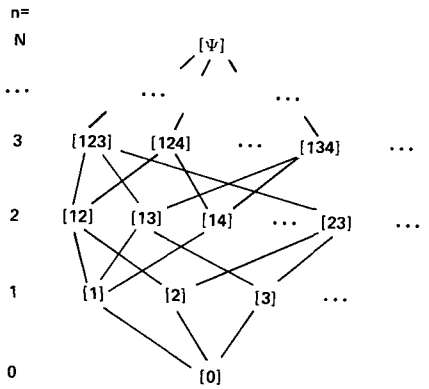


FIG. 1. Lattice of nested sub-spaces of the Hilbert space [Ψ].

generalized n-particle Bethe-Goldstone equation, and the lattice of variational subspaces corresponds to solution of a hierarchy of Bethe-Goldstone equations, proceeding through the lattice diagram in order of increasing order n. Calculations of atomic hyperfine structure and of electronic correlation energies have been carried out by this method [19].

Each variational subspace [ij ...] corresponds to a variational wave function $\Psi_{ij\ldots}$, expanded in Slater determinants as in Eq. (41). For the ground state, the coefficients in this expansion are obtained as the eigenvector corresponding to the lowest energy eigenvalue of the Hamiltonian matrix $H_{\mu\nu}$ defined by the Hilbert space [ij ...]. The gross increment of any physical property, defined as a mean value of some operator F, is defined for normalized $\Psi_{ij\ldots}$ by

$$\Delta F(ij\ldots) = (\Psi_{ij\ldots} \,|\, F \,|\, \Psi_{ij\ldots}) - F_{00} \tag{58}$$

where index zero refers to the reference state Φ_0. Then a net increment of this physical property is defined inductively as the difference between the directly computed gross increment and the sum of all net increments that correspond to proper subspaces of [ij ...] on the lattice diagram. By this definition, the net increment $f_{ij\ldots}$ is

$$f_{ij\ldots} = \Delta F(ij\ldots) - f_0 - \sum_{i \subset X} f_i - \sum_{ij \subset X} f_{ij} - \ldots \tag{59}$$

where the summations are restricted as in Eq. (56). The net increment f_0 is included in this definition to allow for a correction to F_{00} when space [0] represents a configuration, containing Slater determinants in addition to Φ_0.

These definitions of gross and net increments establish a bookkeeping procedure with the property that, in the limit of completeness of the orbital basis set, the exact mean value $\langle F \rangle$ is given by

$$\langle F \rangle - F_{00} = f_0 + \sum_i f_i + \sum_{ij} f_{ij} + \ldots + f_{1\ldots N} \qquad (60)$$

This is just the sum of all possible net increments. Clearly this result holds for any lattice decomposition of the Hilbert space [Ψ], giving a hierarchy of variational calculations that must converge to the exact result for any mean value physical property in a finite number of steps. In practice, calculations of electronic correlation energies can be truncated after n = 2 and calculations of hyperfine interactions after n = 3 [19]. Calculations at the level n = 2 correspond to the "independent pair model" introduced by Brueckner in nuclear many-body theory [18, 36] and characterized by solution of the two-particle equation of Bethe and Goldstone [37].

Eigenvalues and eigenvectors of the very large matrices that occur in this work are obtained by a rapidly converging iterative algorithm, in a form adapted to efficient handling of symmetric matrices [38].

Formal many-body perturbation theory, in the form derived by Brueckner and Goldstone [39], has been applied to the calculation of electronic correlation effects by Kelly [17]. The correlation energy is expressed as a sum to infinite order of "linked-cluster" diagrams, each equivalent to a specific formula involving effective one-electron energy parameters and a product of two-electron matrix elements. The order of the diagram is the number of matrix elements in this product. For example, the second-order diagram, including exchange, written as indicated in Fig. 2, is evaluated as

$$- \frac{(ab|Q\text{-}QP|ij)^2}{\epsilon_a + \epsilon_b - \epsilon_i - \epsilon_j} \qquad (61)$$

The parameters ϵ are one-electron energies, diagonal matrix elements of an effective one-electron operator as in Eq. (25). If this operator is not the Hartree-Fock operator, one-electron potential terms must be included in the numerators of the perturbation diagram formulas.

The electronic correlation energy is given by the sum to infinite order of all linked-cluster energy diagrams. In practical calculations, approximate summation techniques are used to sum certain classes of diagrams to infinite order, but the general summation is truncated at second or third order [17].

Physical properties defined as mean values are given by a similar infinite sum of diagrams, but each diagram includes a special vertex, symbolizing a matrix element $(p|F|q)$ of the operator whose mean value is required. It can be shown [19] that each net increment $f_{ij\ldots}$, as defined by Eq. (59) in the formalism based on a lattice decomposition of the N-electron Hilbert space or hierarchy of generalized Bethe-Goldstone equations, is a sum to infinite order of all linked F-diagrams in which hole lines occur with all of the indicated indices ij ... . Thus the generalized

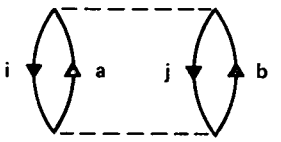


FIG. 2. Second-order energy diagram.

Bethe-Goldstone method can be considered within the context of perturbation theory to be simply a systematic procedure of computing partial summations of perturbation diagrams.

Although continuum functions have been used for the required complete set of orbitals in perturbation theory [17], discrete complete sets could equally well be used, as in applications of superposition of configurations. If this were done, evaluation of matrix elements would involve the same computational procedures in perturbation theory as in the generalized Bethe-Goldstone method.

The essential convergence problem in expanding an electronic wave function as a linear combination of Slater determinants is that a two-particle interaction requires a wave function expressed in relative co-ordinates, not as a sum of products of one-electron orbital functions. A systematic procedure for including correlation factors, explicit functions of relative co-ordinates, in the many-electron wave function has recently been proposed by Boys and Handy [40]. Calculations by this method use techniques similar to those of matrix methods considered here.

c. Molecular integrals

A number of specialized methods have been developed to evaluate the electrostatic integrals given by Eq. (48). Because Hartree-Fock orbitals for a given atom A can be represented to high accuracy as linear combinations of exponential orbital functions,

$$\eta_a(\vec{r}_A) = N_a r_A^{n_a-1} e^{-\zeta_a r_A} Y_{\ell_a}^{m_a}(\theta_A, \phi_A) \tag{62}$$

these functions are used in molecular calculations wherever possible. The efficient evaluation of three- and four-centre electrostatic integrals is a very difficult problem. Because of this, an expansion of molecular orbitals in Gaussian basis functions, with quadratic exponential factors $\exp(-\alpha r_A^2)$, is used for calculations on polyatomic molecules. The general electrostatic integral for Gaussian basis orbitals can be evaluated in closed form [41]. An intermediate method, which takes advantage of the more rapid convergence of expansions in exponential functions, but represents the latter as transforms of Gaussian orbitals in order to evaluate multicentre electrostatic integrals, has been developed by Shavitt [42].

Methods for evaluation of multicentre integrals for exponential basis orbitals have been reviewed by Harris and Michels [43]. In addition to the electrostatic integrals, one-electron integrals of the following forms are required in variational calculations:

$$(a|b),\ (a|T|b),\ (a|V_c|b) \tag{63}$$

These are, respectively, overlap, kinetic energy, and nuclear attraction integrals. The overlap, kinetic energy, and two-centre nuclear attraction integrals can be evaluated in closed form. Special methods for three-centre nuclear attraction integrals are described by Harris and Michels [43].

One approach to multicentre electrostatic integrals is based on the expansion of $1/r_{12}$ given by Eq.(30). This can be expressed in the more explicit form

$$\frac{1}{r_{12}} = \sum_{\ell=0}^{+\ell} \sum_{m=-\ell} \frac{(\ell - m)!}{(\ell + m)!} \frac{r_<^{\ell}}{r_>^{\ell+1}} P_\ell^m(1)\, P_\ell^m(2)\, e^{im(\phi_2 - \phi_1)} \tag{64}$$

Each product of basis orbitals defines a charge density that can be expanded in spherical co-ordinates about a fixed point,

$$\rho_{ab} = \eta_a^*\, \eta_b = \sum_{\ell m} f_{\ell m}^{ab}(r_1)\, P_\ell(\cos\theta_1)\, e^{im\phi_1} \tag{65}$$

Then the general electrostatic integral has the form

$$[\,ab|\,cd\,] = \sum_{\mu\nu} \left(\frac{4\pi}{2\mu+1}\right)^2 \frac{(\mu+|\nu|)!}{(\mu-|\nu|)!} \int_0^\infty r_1^2\, dr_1 \int_0^\infty r_2^2\, dr_2\, \frac{r_<^{\mu}}{r_>^{\mu+1}}\, f_{\mu\nu}^{ab}(1)\, f_{\mu\nu}^{cd}(2) \tag{66}$$

On integrating by parts, this can be put into the more symmetrical form

$$[\,ab|\,cd\,] = \sum_{\mu\nu} \frac{16\pi^2}{2\mu+1} \frac{(\mu+|\nu|)!}{(\mu-|\nu|)!} \int_0^\infty \frac{dx}{x^{2\mu+2}} \left[\int_0^x f_{\mu\nu}^{ab}(r_1)\, r_1^{\mu+2}\, dr_1\right]\left[\int_0^x f_{\mu\nu}^{cd}(r_2) r_2^{\mu+2}\, dr_2\right] \tag{67}$$

The transformation from Eq. (66) to Eq. (67) is of fundamental practical importance in facilitating computation of the very large number of electrostatic integrals required in a typical molecular calculation. In Eq.(67), the outer integration is done by modified Gauss quadrature, requiring evaluation of the integrand at a specified set of points $\{x_i\}$, corresponding to weights W_i in the quadrature formula. For a given value of μ, index ν has only a limited range of values (often only one, depending on molecular symmetry). Then the integral is of the general form, truncated in $(\mu\nu)$ as the series converges,

$$[\,ab|\,cd\,] \cong \sum_{\mu\nu} \sum_i W_i\, F_{\mu\nu}^{ab}(i)\, F_{\mu\nu}^{cd}(i) \tag{68}$$

While the number of integrals is proportional to N_b^4 (where N_b is the number of basis orbitals), only the double summation indicated in Eq. (68) is performed for each final integral. The much more complex computations

required to obtain the terms $F_{\mu\nu}^{ab}(i)$ are weighted only by the number of charge distributions ρ_{ab}, hence by N_b^2. By use of data-handling methods to store and retrieve tables of $F_{\mu\nu}^{ab}(i)$, this density weighting can greatly reduce the total computational effort when N_b is even moderately large (~ 30).

The one-centre expansion of general multicentre integrals leads to an infinite series in index μ. In many cases, this series converges rather slowly, and special methods are needed to extrapolate a finite number of series terms to convergence. Evaluation of the coefficients $f_{\ell m}^{ab}(r)$ in Eq. (65) is also a source of difficulty in the method.

The electrostatic interaction $1/r_{12}$ can also be represented by a two-centre expansion in spheroidal co-ordinates. This leads to a method that has been used extensively for calculations on diatomic molecules [44]. The expansion, due to Neumann, is expressed in terms of regular (P) and irregular (Q) associated Legendre functions,

$$\frac{1}{r_{12}} = \frac{2}{R}\sum_{\ell=0}^{\infty}\sum_{m=-\ell}^{\ell}(-1)^m(2\ell+1)\left[\frac{(\ell-|m|)!}{(\ell+|m|)!}\right]^2 P_\ell^m(\xi_<)Q_\ell^m(\xi_>)P_\ell^m(\eta_1)P_\ell^m(\eta_2)e^{im(\phi_2-\phi_1)} \tag{69}$$

where the plane spheroidal co-ordinates (elliptical co-ordinates) are

$$\xi = \frac{1}{R}(r_A + r_B)\,; \quad \eta = \frac{1}{R}(r_A - r_B) \tag{70}$$

and R is the separation of atoms A and B. The integral [ab| cd] is proportional to

$$I = (-1)^{\nu+1}\int_1^\infty dx\left[\frac{d}{dx}\frac{Q_\mu^\nu(x)}{P_\mu^\nu(x)}\right]\int_1^x \phi_{\mu\nu}^{ab}(\xi_1)\,d\xi_1\int_1^x \phi_{\mu\nu}^{cd}(\xi_2)\,d\xi_2 \tag{71}$$

where

$$\phi_{\mu\nu}^{ab}(\xi) = f^{ab}(\xi)\,(\xi^2-1)^{|\nu|/2}\,P_\mu^\nu(\xi) \tag{72}$$

with f^{ab} the coefficient of a two-centre expansion of ρ_{ab}, similar to Eq. (65). As shown by Ruedenberg [44], the Wronskian of the associated Legendre differential equation can be used to eliminate the irregular functions Q by the formula

$$(-1)^{\nu+1}\frac{d}{dx}\left(\frac{Q_\mu^\nu}{P_\mu^\nu}\right) = \frac{(\mu+|\nu|)!}{(\mu-|\nu|)!}\,\frac{1}{(x^2-1)\,(P_\mu^\nu(x))^2} \tag{73}$$

When this is substituted in Eq. (71), and modified Gauss quadrature is used for the outer integration, the electrostatic integrals reduce to the charge-density weighted form of Eq. (68).

From Eq. (69), the integrations over η_1 and η_2 in $[\,ab|\,cd]$ are already in factorized form, and have not been included explicitly above. The required η-integrals are

$$I_{ab}^{\mu\nu} = \frac{(\mu-|\nu|)!}{(\mu+|\nu|)!} \int_{-1}^{+1} d\eta\ g^{ab}(\eta)\ P_\mu^\nu(\eta)\ (1-\eta^2)^{|\nu|/2} \tag{74}$$

where the charge-density factor g^{ab} is a linear combination of terms of the form $\eta^n \exp(-\zeta\eta)$. These integrals can be evaluated by recurrence formulas [45].

Another method for evaluating multicentre electrostatic integrals is based on properties of three-dimensional Fourier transforms and on the relationship between convolution integrals and products of Fourier transforms. This method was originally proposed by Geller [46], and has been further developed by Silverstone [47] and by Harris [48]. In this method, an electrostatic integral is expressed as a Fourier transform,

$$[\,ab|\,cd] = \int\int \rho_{ab}(\vec{r}_{1A})\ r_{12}^{-1}\ \rho_{cd}(\vec{r}_{2B})\ d^3\vec{r}_1\ d^3\vec{r}_2$$

$$= \frac{(-1)^{\ell_2}}{(2\pi)^3} \int e^{-i\vec{k}\cdot\vec{R}}\ \rho_{ab}^T(\vec{k})\ V^T(\vec{k})\ \rho_{cd}^T(\vec{k})\ d^3\vec{k} \tag{75}$$

Here ρ_{ab} is a charge density expanded about centre A, ρ_{cd} a charge density expanded about B, $\vec{R}$ is the vector distance from A to B, and superscript T denotes a three-dimensional Fourier transform.

It is convenient to use the following notation:

$$(n\ell m\,\zeta) = r^{n-1} \exp(-\zeta r)\ Y_\ell^m(\theta,\phi) \tag{76}$$

and to assume that the charge densities ρ_{ab} and ρ_{cd} are expanded as linear combinations of terms $(n\ell m\,\zeta)$ about centres A and B, respectively. The required Fourier transforms are of the form

$$V^T(\vec{k}) = 4\pi/k^2 \tag{77}$$

$$\phi_A^T(\vec{k}) = (n\ell m\,\zeta)^T = 4\pi i^\ell\,(n\ell\zeta)^T\ P_\ell^m(\cos\theta_k)\ e^{im\phi_k} \tag{78}$$

where, in terms of spherical Bessel functions $j_\ell(kr)$,

$$(n\ell\zeta)^T = \int_0^\infty r^{n+1}\, j_\ell(kr)\, e^{-\zeta r}\ dr = G(\ell, \ell+n+1\,|\,k,\ \zeta) \tag{79}$$

Efficient methods for computation of the integrals over spherical Bessel functions denoted by G here have been developed for work in electron-atom scattering theory [49].

The final form of the electrostatic integrals is a linear combination of terms of the form

$$c^L(\ell,\ell')\int_0^\infty j_L(kR)\,(n\ell\zeta)_A^T\,(n\ell\zeta)_B^T\,dk \tag{80}$$

where $c^L(\ell,\ell')$ is an angular momentum coupling coefficient. When modified Gauss quadrature is used for the outer integral over k, the general electrostatic integral reduces to a charge-density weighted expansion

$$[ab|cd] \cong \sum_{L\ell\ell'}\sum_i W_i\, K^{ab}_{L\ell}(i)\, K^{cd}_{L\ell'}(i) \tag{81}$$

A one-centre charge distribution constructed from exponential basis orbitals can be expressed as a finite sum of terms $(n\ell m\zeta)$. The sum over $(L\ell\ell')$ in Eq. (81) reduces to a finite sum for the two-centre "Coulomb" integral between two one-centre charge densities. Efficient formulas for evaluation of such integrals have been derived from recurrence relations for the integrals of Eq. (80) [48].

5. SYMMETRY ALGEBRA

Group representation theory can be used to take full advantage of symmetry in simplifying atomic and molecular calculations. Effective use of such symmetry algebra can require very complex and sophisticated program organization. Computations of symmetry algebra tend not to be very time-consuming, despite the complexity of the formalism, but such computations are important when they can reduce the most heavily weighted steps of an overall computation by a significant factor.

Basic concepts

Given a group $\mathscr{G}$ of symmetry operations $\{G\}$, and a set of functions $\{\psi\}$ such that the transformed functions $G\psi$ are defined and can be expanded in the linear space spanned by $\{\psi\}$, symmetry-adapted functions $\psi^\mu_{\alpha\lambda}$ are defined by

$$G\psi^\mu_{\alpha\lambda} = \sum_{\mu'} \psi^{\mu'}_{\alpha\lambda}\,\Gamma^\lambda_{\mu',\mu}(G), \quad \text{all } G \in \mathscr{G} \tag{82}$$

The expansion coefficients Γ are matrix elements of an irreducible representation [50] of group $\mathscr{G}$, characterized by symmetry species index λ and by subspecies indices μ,μ'. In general, the functions ψ are complex

and the matrices Γ unitary. In words, a symmetry-adapted function ψ^{μ}_{λ} transforms as a basis function for irreducible representation Γ^{λ}, corresponding to row μ of the representation. The index α here is used to distinguish essentially different functions that share the same transformation properties.

In its most general sense, the term configuration can be used to denote a linear space $\{\psi\}$ closed under operations of a given transformation group, in the sense that $G\psi$ lies in the linear space if ψ does. A function that can be expressed as a linear combination of the functions $\{\psi\}$, and hence lies in the linear space, is said to belong to the configuration. For basis functions ψ_α of a configuration, the transformation law is

$$G\psi_\alpha = \sum_\beta \psi_\beta \, \Gamma_{\beta\alpha}(G), \text{ all } G \in \mathcal{G} \tag{83}$$

If the functions $\{\psi_\alpha\}$ are orthonormal, a unitary transformation can be found that puts Eq. (83) into the form of Eq. (82). This transformation converts the matrix $\Gamma_{\beta\alpha}$ into a direct sum of irreducible representation matrices, which occur as disconnected square matrix blocks strung along the diagonal of the transform $U\Gamma U^{-1}$ of $\Gamma_{\beta\alpha}$. This operation is called complete reduction of the matrix $\Gamma_{\beta\alpha}$, and the original matrix is said to be reducible if more than one block can occur in the direct sum.

For finite groups, matrices of a given reducible representation can be combined linearly to give a projection operator $^{\mu}_{\lambda}\mathcal{O}$, which produces an unnormalized symmetry-adapted function with the given symmetry species indices when applied to an arbitrary basis function of the given representation [50]. When the same irreducible representation occurs more than once in the reducible representation generated by a given configuration, Schmidt orthonormalization of the projected functions $^{\mu}_{\lambda}\mathcal{O}\,\psi_\alpha$ can be used to produce an orthonormal set of symmetry-adapted functions, spanning the configuration.

For continuous groups, in particular for the three-dimensional rotation group, less direct methods must be used, as described below. In any case the construction of orthonormal sets of symmetry-adapted functions from a given configuration is an important step in the practical use of symmetry algebra, since such functions are assumed in specific applications of group representation theory.

A linear operator Ω which acts on the functions ψ transforms as $G\Omega G^{-1}$ under symmetry transformations, if the functions ψ transform as $G\psi$. Corresponding to the definition of symmetry-adapted function, an irreducible tensor operator $\Omega_{\Lambda M}$ is defined by the linear transformation law,

$$G\Omega_{\Lambda M}G^{-1} = \sum_{M'} \Omega_{\Lambda M'} \Gamma^{\Lambda}_{M'M}(G), \text{ all } G \in \mathcal{G} \tag{84}$$

Many important applications of symmetry algebra make use of the Wigner-Eckart theorem [11, 50, 51] for matrix elements of irreducible tensor operators:

$$(\alpha\lambda\mu \,|\Omega_{\Lambda M}|\, \alpha'\lambda'\mu') = (\lambda\mu\,|\Omega M\,|\,\lambda'\mu')\ (\alpha\lambda\,||\Omega_\Lambda||\,\alpha'\lambda') \tag{85}$$

The great simplification achieved by this formula is due to the fact that the subspecies indices (μ, M, μ') occur only in the factor $(\lambda\mu\,|\Lambda M\,|\lambda'\mu')$, a generalized Clebsch-Gordan coefficient which depends only on transformation properties. The reduced matrix element $(\alpha\lambda\,||\Omega_\Lambda||\alpha'\lambda')$ contains all information relevant to the specific operator Ω and functions $\psi_\alpha, \psi'_\alpha$. In the special case of an invariant operator,

$$(\lambda\mu\,|00|\,\lambda'\mu') = \delta_{\lambda\lambda'}\,\delta_{\mu\mu'} \tag{86}$$

Thus matrix elements of such an operator vanish unless $\lambda = \lambda'$ and $\mu = \mu'$, and are otherwise independent of the subspecies index μ.

It is often required to evaluate matrix elements of tensor operators constructed as symmetry-adapted linear combinations of products of simpler operators. Such matrix elements can be expressed in terms of generalized coupling coefficients (n-j symbols), and of reduced matrix elements of the simple operators. For example, with reference to the rotation group and angular momentum eigenfunctions and quantum numbers, the matrix element of a tensorial scalar product of two commuting tensor operators is

$$(\gamma j_1' j_2' J' M\,|\,\vec{T}(k)\cdot\vec{U}(k)\,|\gamma j_1 j_2 J M)$$

$$= (-1)^{j_1+j_2'+J}\;\delta_{J'J}\,\delta_{M'M}\begin{Bmatrix} J & j_2' & j_1' \\ k & j_1 & j_2 \end{Bmatrix}\sum_{\gamma''}(\gamma' j_1'||\,T(k)\,||\gamma'' j_1)\,(\gamma'' j_2'||\,U(k)\,||\gamma j_2) \tag{87}$$

The coupling coefficient indicated here in curly brackets is a 6-j symbol, equal except for a phase factor to a Racah W-coefficient [11].

To make use of the Wigner-Eckart theorem, it is necessary to construct symmetry-adapted wave functions or basis functions for linear expansion and to compute the necessary generalized coupling coefficients. A program that computes general n-j symbols for the rotation group has recently been published [52]. When large numbers of such coefficients are required for repetitive use, it would be desirable, in principle, to prepare tables for access during a particular calculation. The number of indices in typical coupling coefficients can lead to very large tables, and to a difficult data retrieval problem on a computer.

Application to the N-electron problem

With respect to a symmetry group $\mathcal{G}$ and its irreducible representations (λ), a configuration can be defined for an N-electron system as the linear space spanned by Slater determinants whose occupied orbitals have indices represented by

$$(n_1\lambda_1\mu_1)\;(n_2\lambda_2\mu_2)\;\ldots\;(n_N\lambda_N\mu_N) \tag{88}$$

for all possible values of the set of subspecies indices $\{\mu_i\}$. This definition is compatible with the general definition given above of a configuration as

a linear space closed under a group of transformations. Since all choices of $\{\mu_i\}$ are implied, an N-electron configuration can be symbolized by

$$\prod_i (n\lambda)_i^{d_i} = (n_1\lambda_1)\ (n_2\lambda_2)\ \ldots\ (n_N\lambda_N) \tag{89}$$

where d_i is the occupation number of the subshell with quantum numbers $(n\lambda)_i$. Index n (principal quantum number) is used to denote different subshells of the same symmetry species.

Although essentially different approaches have been used, it is convenient to represent symmetry-adapted N-electron wave functions as linear combinations of the basis Slater determinants of a configuration as defined above. In the non-relativistic approximation for atoms (Russell-Saunders or L-S coupling), spin and orbital angular momentum representations of the rotation group are uncoupled. The orbital symmetry indices $(\lambda; \mu)$ become the quantum numbers $(\ell, s; m_\ell, m_s)$, with $s = \frac{1}{2}$. The N-electron symmetry indices $(\Lambda; M)$ become quantum numbers $(L, S; M_L, M_S)$. Orbital quantum numbers $\ell = 0, 1, 2, \ldots$ are denoted by the conventional symbols s, p, d, ..., respectively, and quantum numbers $L = 0, 1, 2, \ldots$ are denoted by S, P, D, ... The spin quantum numbers $m_s = -\frac{1}{2}, +\frac{1}{2}$ are denoted by β, α, respectively. An N-electron symmetry-adapted function is an eigenfunction of operators $\vec{L}^2$ and $\vec{S}^2$, with eigenvalues $\hbar^2 L(L+1)$ and $\hbar^2 S(S+1)$, respectively. These quantum numbers are condensed into a term symbol of the form

$$^{2S+1}L_{M_L}\ (M_S) \tag{90}$$

As an example, consider the $1s^2 2s^2 2p^2$ ground-state configuration of atomic carbon. The closed subshells $1s^2$ and $2s^2$ can be ignored, since only one choice of (m_ℓ, m_s) values is possible. The basis Slater determinants for this configuration can be grouped according to quantum numbers $M_L = \Sigma m_\ell$ and $M_S = \Sigma m_s$ and listed in a Slater table as shown in Table I.

TABLE I. SLATER TABLE FOR CONFIGURATION p^2

	$M_S = 1$	0	...
$M_L = 2$		^1D $p_1^\beta p_1^\beta$	...
1	^3P $p_0^\alpha p_1^\alpha$	$^3P + {}^1D$ $p_0^\beta p_1^\alpha,\ p_0^\beta p_1^\alpha$	...
0	^3P $p_{-1}^\alpha p_1^\alpha$	$^3P + {}^1D + {}^1S$ $p_{-1}^\beta p_1^\alpha,\ p_0^\beta p_0^\alpha,\ p_0^\beta p_{-1}^\alpha$	...
...	...	...	

Since all configurations with a single np^2 open shell have the same structure, the principal quantum number n can be omitted, and each determinant is denoted by a symbol

$$p^{m_s}_{m_\ell}\, p^{m'_s}_{m'_\ell}, \tag{91}$$

Here m_ℓ has values $-1, 0, +1$ and m_s has values $-\frac{1}{2}, +\frac{1}{2}$ denoted by β, α, respectively.

The possible L, S terms in a given configuration can be ascertained by simple inspection of the Slater table. This follows from the fact that the linear space defined by each box of the table must be closed under unitary transformations of the configuration that preserve quantum numbers M_L and M_S. Moreover, any L, S term that occurs in the configuration accounts for exactly one function in the linear space of each box with $M_L = -L, -L+1, \ldots, L$ and $M_S = -S, -S+1, \ldots, S$, because by definition the configuration is closed with respect to symmetry operations, which mix up all M_L and M_S values. This property can be expressed most directly in terms of ladder operators:

$$\begin{aligned} L^+ &= L_x + iL_y \\ S^+ &= S_x + iS_y \end{aligned} \tag{92}$$

constructed from the indicated angular momentum operators [11]. For generalized angular momentum j, m, the ladder operator has the property that

$$j^+ \psi_{jm} = \psi_{j\,m+1} \tag{93}$$

Since the "ladder" of m values terminates with m = j, it follows that

$$j^+ \psi_{jj} \equiv 0 \tag{94}$$

An immediate consequence of this result is that

$$j^+ \psi_m \equiv 0, \quad (\psi_m | \psi_m) > 0 \tag{95}$$

is a necessary and sufficient condition for ψ_m to be a symmetry-adapted function ψ_{jm} with j = m. As applied to L, S configurations, the L^+, S^+ operators connect adjacent boxes in the Slater table either vertically or horizontally, respectively. The theorem of Eq. (95) implies that any box containing more determinant symbols than both the adjacent boxes for $M_L + 1$ and $M_S + 1$ must represent a linear space containing a term $L = M_L$, $S = M_S$. As indicated in the example of Table I, the terms can be enumerated by working inwards through the Slater table, starting with the largest values of M_L and M_S.

In Table I, a unique determinant occurs with $M_L = 1$, $M_S = 1$, with vacant boxes above and to the left. Hence a ^3P term (L = 1, S = 1) is present in the configuration, including all nine possible M_L, M_S combinations.

Similarly, a unique determinant has $M_L = 2$, $M_S = 0$, with vacant boxes above and to the left, indicating a ^1D term ($L = 2$, $S = 0$). These two terms account for the entire table except for the box $M_L = 0$, $M_S = 0$, which contains three determinants, indicating the final term to be ^1S ($L = 0$, $S = 0$).

The most straightforward method of constructing L, S term functions is to apply the theorem of Eq. (95) to those boxes in the Slater table with M_L, M_S equal to L, S values for expected terms. This gives a set of homogeneous equations for coefficients of the symmetry-adapted functions, obtained from matrix elements of the ladder operator L^+ and S^+. The L, S eigenfunctions obtained in this way from Table I are, in the notation of Eq. (90),

$$
\begin{aligned}
{}^3_1P_1 &= p_0^\alpha\, p_1^\alpha \\
{}^1_0D_2 &= p_1^\beta\, p_1^\alpha \\
{}^1_0S_0 &= 3^{-\frac{1}{2}}\,(p_{-1}^\beta\, p_1^\alpha - p_0^\beta\, p_0^\alpha + p_1^\beta\, p_{-1}^\alpha)
\end{aligned}
\tag{96}
$$

Three distinct methods for constructing symmetry-adapted N-electron wave functions will be considered here.

a. Default

This designation is intended to emphasize the fact that symmetry-adapted functions represent a convenience and not a necessity in N-electron wave function calculations. In matrix methods, if the set of basis functions is chosen to span complete configurations, an eigenvector of the matrix of an invariant Hamiltonian necessarily represents a symmetry-adapted variational wave function except in cases of accidental degeneracy. When there are additive symmetry quantum numbers, such as M_L, M_S, and parity for atomic wave functions, only basis functions with given values of these quantum numbers need be included.

To ensure that a variational calculation should produce symmetry-adapted functions in L-S coupling, any variational Hilbert space that includes a given determinant should be extended to include all determinants of the same configuration with the same M_L, M_S, and parity. This means that each box in a Slater table should be included or excluded in its entirety from the variational wave function. If this rule is followed, the method of variational calculations of wave functions based on lattice decomposition of the N-electron Hilbert space leads to symmetry-adapted functions at each stage of calculation [19]. A similar procedure can be followed for the more complicated variational wave functions required in electron scattering theory.

b. Progressive vector coupling

When the computational saving due to reduction of the size of matrices or of systems of coupled equations outweighs the consideration of simplicity of computational procedure and programming, symmetry-adapted functions can be constructed explicitly. An effective general method has been proposed by Fano [54] and implemented in a published computer program by Hibbert [55].

This method uses the technique of progressive vector coupling. Subshells of equivalent orbitals (same $n\ell$ values) are adjoined in sequence and progressively coupled to definite L and S values for antisymmetric wave functions. The term structure for antisymmetric functions for each subshell $(n\ell)^d$ is obtained by progressive coupling of one $n\ell$-orbital to the terms of $(n\ell)^{d-1}$. This coupling within each subshell is defined by fractional parentage coefficients, which can be computed and tabulated independently for each possible subshell. A program for p- and d-subshells has been published [56].

The method of Fano and Hibbert proceeds through the construction of symmetry-adapted functions Ψ to the evaluation of matrix elements $(\Psi|H|\Psi')$. Progressive vector coupling schemes must be specified for Ψ and Ψ'. In addition to fractional parentage coefficients, general coupling coefficients (n-j symbols) are required for the calculation. These are supplied by the program of Burke [52].

c. Projection operator tables

Properties of projection operators can be used to simplify matrix elements of an invariant operator [57]. The general formula is

$$(\Psi_{I\mu}|H|\Psi_{II\nu}) = (k_\nu/k_\mu)^{\frac{1}{2}} \sum_{i=1}^{n_I} \sum_{j=1}^{m_{II}} x_{\mu i}^{I*}\, a_{\nu j}^{II} (\Phi_i|H|\Phi_j) \tag{97}$$

where the functions Ψ are orthonormalized symmetry-adapted functions, each with a dual expansion of the form

$$\Psi_\mu = k_\mu^{\frac{1}{2}} \sum_{j=1}^{m} a_{\mu j}\ \theta_j = k_\mu^{-\frac{1}{2}} \sum_{i=1}^{n} x_{\mu i}\, \Phi_i \tag{98}$$

The function

$$\theta_j = \mathcal{O}_{LS}\Phi_j, \text{ where } \mathcal{O}_{LS}^2 = \mathcal{O}_{LS} \tag{99}$$

is defined by action of the L, S projection operator $\mathcal{O}_{LS}$ on a Slater determinant Φ_j. In Eqs (98), if there are n determinants of specified M_L, M_S, and parity in a given configuration, index $m \leq n$ is the number of symmetry-adapted functions of given L, S in that configuration. The essential simplification achieved by Eq. (97) is that m, which for configurations with several open subshells may be very much smaller than n, defines the range for one of the two summation indices. Properties of reduced matrix elements may be used, as in other methods, to simplify computation of the matrix elements $(\Phi_i|H|\Phi_j)$ between Slater determinants.

The coefficients $\{a; x\}$ for any given configuration can be obtained directly by an algorithm that makes use of the homogeneous equations, Eqs (95), constructed from matrix elements of the ladder operators L^+ and S^+. The only auxiliary coupling coefficients required in using this method are the Gaunt coefficients $c^k(\ell m; \ell' m')$ tabulated by Condon and Shortley [53]. Matrix elements for the two ^2D terms in configuration d^3 have been computed in detail as an example of this method [57].

CASE STUDY: ELECTRON-ATOM SCATTERING

As indicated in Eq. (4), the wave function for electron scattering by an N-electron atom is of the form

$$\Psi = \sum_{p} \mathcal{A}\ \theta_p\ \psi_p + \sum_{\mu} \Phi_\mu\ c_\mu \qquad (100)$$

Index p is used here to denote different partial waves as well as energies. In principle, each function θ_p is an exact stationary state wave function for the target atom, corresponding, for open channel p, to target atom energy $E_p < E$, where E is the total energy of the N+1-electron system. The functions $\{\Phi_\mu\}$ constitute a Hilbert space of normalizable N+1-electron wave functions, which can be expressed in terms of virtual excitations as in the stationary state problem. Thus the summation over μ is infinite, in principle, and techniques such as the lattice decomposition of $\{\Phi_\mu\}$, in analogy to the structure illustrated in Fig. 1, must be used to make calculations feasible.

In the close-coupling formalism, the summation over $\{\Phi_\mu\}$ is severely truncated, but each such closed-channel wave function that is included in the sum is computed variationally. The closed-channel N+1-electron wave functions are written in the form

$$\Phi_\mu\ c_\mu = \mathcal{A}\ \theta_q\ \psi_q \qquad (101)$$

where ψ_q is a quadratically integrable closed-channel orbital, and θ_q is a specified "pseudostate" or polarization function, chosen so that the electric dipole polarizability of some function θ_p can be computed from a first-order wave function proportional to θ_q. The variational equations for the open- and closed-channel orbitals (ψ_p; ψ_q), for N_p open and N_q closed channels, form a system of $N_p + N_q$ coupled integro-differential equations in the radial variables. Since the number of coupled equations increases with each addition to the set of functions $\{\Phi_\mu\}$, it is impractical to carry such calculations to the quantitative limit of completeness of the Hilbert space $\{\Phi_\mu\}$.

To take the full Hilbert space $\{\Phi_\mu\}$ into account, it was proposed that matrix methods be used for all closed-channel functions, and a systematic hierarchy of variational calculations be carried out based on a lattice decomposition of this Hilbert space [58]. The variational equations take the form of a system of coupled integro-differential equations for the open-channel orbitals ψ_p, coupled to a system of matrix equations of large dimension.

The system of coupled integro-differential equations can be eliminated completely if the channel orbitals ψ_p are themselves expanded as linear combinations of specified basis functions. Then the scattering calculation becomes a pure matrix calculation to determine these expansion coefficients. For a single scattering channel, the radial factor $f_p(r)$ of ψ_p satisfies the usual bound-state boundary condition at $r = 0$ and has the asymptotic form, for orbital angular momentum ℓ_p,

$$f_p(r) \sim r^{-1} \sin\left(k_p r - \tfrac{1}{2}\ell_p \pi + \delta_p\right) \qquad (102)$$

This can be written in the more general form, also applicable to multichannel scattering,

$$f_p = \alpha_{0p} S_p + \alpha_{1p} C_p \tag{103}$$

where

$$S_p \sim r^{-1} \sin\left(k_p r - \tfrac{1}{2}\ell_p \pi\right)$$
$$C_p \sim r^{-1} \cos\left(k_p r - \tfrac{1}{2}\ell_p \pi\right) \tag{104}$$

Scattering cross-sections can be computed from the coefficients $\{\alpha_{1p}\}_p$, obtained by setting each of the coefficients α_{0p}, in turn equal to unity, with all others zero. Except for constant factors, the square array of coefficients α_{1p} obtained in this way is equal to the reactance matrix R, often denoted by K in the literature [59]. Because of the form assumed for the wave function, Eq. (100), the functions S_p and C_p can be orthogonalized to all of the normalizable basis orbitals in a given calculation. This will be assumed here. Then the functions Φ_μ serve to expand the inner part of the channel orbitals, as well as to represent correlation and polarization effects through virtual excitations.

A serious practical difficulty occurs in attempting to apply standard variational methods [60, 61] to evaluate the coefficients α in Eq. (103). The computed reactance matrix elements have spurious poles whose location, as a function of energy, varies with an arbitrary change of the basis set used for linear expansion of a variational wave function [62]. New analysis of this problem, for multichannel as well as single-channel scattering [63], showed that these spurious singularities could be avoided by alternative use of the Kohn [60] or inverse Kohn [61] methods, referring to variational calculation of the elements of the reactance matrix or of its reciprocal, respectively. The choice of formalism depends on an easily computed criterion. Because of the complexity of calculations required, it would be very difficult to design a reliable general computer program if unpredictable results could arise from any given calculation.

For electron-neutral atom scattering, the regular spherical Bessel functions $j_\ell(kr)$ satisfy the boundary conditions required of the functions S in Eq. (103). The actual functions used are obtained by orthogonalizing $j_\ell(kr)$ to all normalizable basis orbitals with the same ℓ-value. The radial factors of these basis orbitals are taken to be of the form $r^{n-1} \exp(-\zeta r)$. Functions C are obtained from a linear combination of $j_{\ell+1}(kr)$ and $j_{\ell+2}(kr)$, chosen to agree with the first two terms of the asymptotic expansion of $n_\ell(kr)$, the irregular spherical Bessel function [64].

Despite the fact that all integrals involve only one centre, the electrostatic integrals involving mixed exponential and Bessel functions are nontrivial. The most difficult integral required in electron-atom scattering, for two or more open channels of different energy, is the exchange integral,

$$X(\lambda\mu pq\,|k_1 k_2 \alpha\beta) = \int_0^\infty j_\lambda(k_1 r_1) r_1^{p-\lambda} e^{-\alpha r_1} \int_{r_1}^\infty j_\mu(k_2 r_2) r_2^{q-\mu} e^{-\beta r_2}\, dr_2 dr_1 \tag{105}$$

It was found necessary to develop a number of new formulas and methods to evaluate these electrostatic scattering integrals efficiently to high accuracy [65]. The even more difficult problem of evaluating electrostatic integrals over Coulomb wave functions, required for variational calculations of electron-ion scattering, has recently received attention [66].

In applying the Kohn or inverse Kohn variational formalism to the wave function indicated in Eq. (100) it is necessary to construct the variational functional

$$(\Psi \,|H-E|\,\Psi) = \sum_{ij} \sum_{pq} \alpha_{ip}\, M_{ij}^{pq}\, \alpha_{jq} \tag{106}$$

where the coefficients α are defined in Eq. (103). The auxiliary matrix M_{ij}^{pq} is defined by [63]

$$M_{ij}^{pq} = M_{IJ}^{pq} - \sum_{\mu} \sum_{\nu} M_{I\mu}^{p} \,(M^{-1})_{\mu\nu}\, M_{\nu j}^{q} \tag{107}$$

where i = I, j = J with values 0, 1 as in Eq. (103), and p, q are open channel indices. The matrices combined in Eq. (107) are the bound-bound matrix (Hermitian)

$$M_{\mu\nu} = H_{\mu\nu} - E\ \delta_{\mu\nu} \tag{108}$$

where

$$H_{\mu\nu} = (\Phi_\mu,\ H\,\Phi_\nu) \tag{109}$$

the bound-free matrix (Hermitian)

$$M_{\mu I}^{p} = (\Phi_\mu,\ (H-E)\ \theta^{Ip}) \tag{110}$$

and the free-free matrix (non-Hermitian)

$$M_{IJ}^{pq} = (\theta^{Ip},\ (H-E)\,\theta^{Jq}) \tag{111}$$

Here

$$\theta^{Ip} = \mathscr{A}\,\theta_p\,\psi_{Ip} \tag{112}$$

The matrix $M_{\mu\nu}$ has the dimension of the N+1-particle Hilbert space. As indicated by Eq. (55), this dimensionality grows very rapidly with N_b, the number of basis orbitals, and with the level of virtual excitation taken into account. It is easily possible, in calculations on relatively light atoms, to have a matrix $M_{\mu\nu}$ with several million non-zero elements. Standard methods of matrix inversion are extremely inefficient for such a matrix, because the elements must be accessed in a non-sequential order. When only a small fraction of the matrix can be contained in the random access fast memory of a computer, non-sequential processing presents a very difficult data handling problem.

Instead of inverting the matrix $M_{\mu\nu}$, Eq. (107) can be evaluated indirectly by a method that avoids non-sequential data processing, and in fact bypasses an entire step of matrix processing as compared with use of Gauss elimination [67]. Since all matrix elements considered here are real numbers, the bound-bound matrix can be factorized in the form

$$M_{\mu\nu} = \sum_{\rho} \sigma_\rho \, T_{\mu\rho} \, T_{\nu\rho} \tag{113}$$

defining the lower triangular matrix $T_{\mu\rho}$ and a diagonal sign matrix σ_ρ whose diagonal elements are ± 1 only. An auxiliary rectangular matrix B is defined in terms of the bound-free matrix by

$$\sum_{\rho} T_{\mu\rho} \, B^{p}_{\rho I} = M^{p}_{\mu I} \tag{114}$$

The matrices T and B can both be constructed by simple algorithms requiring only sequential data processing [67]. Equation (107) is evaluated in the form

$$M^{pq}_{ij} = M^{pq}_{IJ} - \sum_{\mu} \sigma_\mu \, B^{p}_{\mu I} \, B^{q}_{\mu J} \tag{115}$$

In implementing this method, the problem of constructing computer programs of general applicability was simplified by not constructing symmetry-adapted L, S eigenfunctions. Instead, by including complete boxes in the relevant Slater tables in constructing the determinant list $\{\Phi_\mu\}$, and by diagonalizing the computed reactance matrix, advantage is taken of the fact that the final eigenchannel wave function is necessarily symmetry-adapted. This is an example of the default method of symmetry algebra. At a later stage of development, when the general method and programs have been thoroughly tested, the linear dimensions of matrices required in this method can be considerably reduced by explicit construction of L, S eigenfunctions prior to the evaluation of matrix elements.

REFERENCES

[1] SCHWARTZ, C., Phys. Rev. 124 (1961) 1468; ARMSTEAD, R.L., Phys. Rev. 171 (1968) 91; GAILITIS, M., in Physics of Electronic and Atomic Collisions (Proc. 4th Int. Conf. Montreal, 1965) 10.

[2] SMITH, F.T., in Atomic Physics (Proc. 1st Int. Conf. New York, 1969) Plenum Press (1969) 353.

[3] LITTLE, W.A., J. Polymer Sci., Pt. C, Polymer Symposium (1967) 3.

[4] LITTLE, W.A., J. chem. Phys. 49 (1968) 420; GUTFREUND, H., LITTLE, W.A., Phys. Rev. 183 (1969) 68; J. chem. Phys. 50 (1969) 4468, 4478.

[5] LEFEBVRE-BRION, H., MOSER, C.M., NESBET, R.K., J. mol. Spectr. 13 (1964) 418.

[6] TILFORD, S.G., VANDERSLICE, J.T., WILKINSON, P.G., Can. J. Phys. 43 (1965) 450.

[7] NESBET, R.K., Phys. Rev. Letts 24 (1970) 1155; Phys. Rev. A2 (1970) 1208.

[8] MATCHA, R.L., NESBET, R.K., Phys. Rev. 160 (1967) 72.

[9] CLEMENTI, E., IBM J. Res. Develop. 9 Suppl. (1965) 2; J. chem. Phys. 38 (1963) 2248; 39 (1963) 175.

[10] NESBET, R.K., Phys. Rev. 175 (1968) 2; A3 (1971) 87.

[11] EDMONDS, A.R., Angular Momentum in Quantum Mechanics, Princeton University Press, Princeton (1957); ROSE, M.E., Elementary Theory of Angular Momentum, John Wiley, New York (1957).

[12] VANDERSLICE, J.T., MASON, E.A., MAISCH, W.G., J. mol. Spectr. 3 (1959) 17.
[13] LANDAU, L., Z. Phys. Sov. Union 2 (1932) 46; ZENER, C., Proc. Roy. Soc. (London) A137 (1932) 696.
[14] LESTER W.A., Jr., Methods in Computational Physics 10 (1971) 211; see also these Proceedings.
[15] HARTREE, D.R., The Calculation of Atomic Structures, John Wiley, New York (1957).
[16] ROOTHAAN, C.C.J., Rev. mod. Phys. 23 (1951) 69.
[17] KELLY, H.P., Adv. chem. Phys. 14 (1969) 129.
[18] BRUECKNER, K.A., Phys. Rev. 96 (1954) 508; 97 (1955) 1953; 100 (1955) 36; The Many-Body Problem (DE WITT, B., Ed.), John Wiley, New York (1959) 47.
[19] NESBET, R.K., Adv. chem. Phys. 14 (1969) 1; Phys. Rev. A2 (1970) 661.
[20] BURKE, P.G., SEATON, M.J., Methods in Computational Physics 10 (1971) 1.
[21] WEISS, A.W., Phys. Rev. 162 (1967) 71.
[22] TAVARD, C., J. de Chim. Phys. 66 (1969) 1130; TAVARD, C., BONHAM, R.A., J. chem. Phys. 50 (1969) 1736.
[23] KOLOS, W., Adv. Quantum Chem. 5 (1970) 99.
[24] DE VOGELAERE, R., J. Res. Natl. Bur. Std. 54 (1955) 119; LESTER, W.A., Jr., J. comput. Phys. 3 (1968) 322.
[25] ALLISON, A.C., J. comput. Phys. 6 (1970) 378.
[26] GORDON, R.G., J. chem. Phys. 51 (1969) 14.
[27] COOLEY, J.W., Math. Comput. 15 (1961) 363; SHARE Program No. 1072.
[28] NUMEROV, B., Publ. de l'Observ. Astrophys. Central Russie 2 (1933) 188; HAMMING, R.W., Numerical Methods for Scientists and Engineers, McGraw-Hill, New York (1965) 215.
[29] FROESE, C., Can. J. Phys. 41 (1971) 1895.
[30] FROESE-FISCHER, C., Comput. Phys. Communs 2 (1971) 124.
[31] HARTREE, D.R., The Calculation of Atomic Structures, John Wiley, New York (1957).
[32] FROESE-FISCHER, C., Comput. Phys. Communs 1 (1970) 151.
[33] BURKE, P.G., Adv. Phys. 14 (1965) 521; CONNEELY, LIPSKY, SMITH, BURKE, HENRY, comput. Phys. Communs 1 (1970) 306.
[34] ROOTHAAN, C.C.J., Rev. mod. Phys. 23 (1951) 69.
[35] NESBET, R.K., Rev. mod. Phys. 33 (1961) 28; 35 (1963) 552; ROOTHAAN, C.C.J., BAGUS, P.S., Meth. Comput. Phys. 2 (1963) 47.
[36] GOMES, L.C., WALECKA, J.D., WEISSKOPF, V.F., Ann. Phys. (New York) 3 (1958) 241.
[37] BETHE, H.A., GOLDSTONE, J., Proc. Roy. Soc. (London) A238 (1957) 551; SZASZ, L., Z. Naturf. 14a (1959) 1014.
[38] NESBET, R.K., J. chem. Phys. 43 (1965) 311; SHAVITT, I., J. comput. Phys. 6 (1970) 124.
[39] GOLDSTONE, J., Proc. Roy. Soc. (London) A239 (1957) 267.
[40] BOYS, S.F., HANDY, N.C., Proc. Roy. Soc. (London) A310 (1969) 43, 63.
[41] BOYS, S.F., Proc. Roy. Soc. (London) A200 (1950) 542.
[42] SHAVITT, I., Meth. comput. Phys. 2 (1963) 1.
[43] HARRIS, F.E., MICHELS, H.H., Adv. chem. Phys. 13 (1967) 205.
[44] RUEDENBERG, K., J. Chem. Phys. 19 (1951) 1459; MERRYMAN, P., unpublished program (1957); MERRYMAN, P., MOSER, C.M., NESBET, R.K., J. chem. Phys. 32 (1960) 631; HARRIS, F.E., J. chem. Phys. 32 (1960) 3.
[45] CORBATO, F.J., J. chem. Phys. 24 (1956) 452.
[46] GELLER, M., J. chem. Phys. 41 (1964) 4006.
[47] SILVERSTONE, J., J. chem. Phys. 47 (1967) 537; 48 (1968) 4098, 4106.
[48] HARRIS, F.E., J. chem. Phys. 51 (1969) 4770.
[49] LYONS, J.D., NESBET, R.K., J. comput. Phys. 4 (1969) 499; HARRIS, F.E., MICHELS, H.H., J. comput. Phys. 4 (1969) 579.
[50] HAMERMESH, M., Group Theory, Addison-Wesley, New York (1962); WIGNER, E.P., Group Theory and its Application to the Quantum Mechanics of Atomic Spectra, Academic Press, New York (1959).
[51] ECKART, C., Rev. mod. Phys. 2 (1930) 305.
[52] BURKE, P.G., comput. Phys. Communs 1 (1970) 241.
[53] CONDON, E.U., SHORTLEY, G.H., The Theory of Atomic Spectra, Cambridge University Press, New York (1951).
[54] FANO, U., Phys. Rev. 140 (1965) A67.
[55] HIBBERT, A., comput. Phys. Communs 1 (1970) 359.
[56] ALLISON, D.C.S., comput. Phys. Communs 1 (1970) 15.
[57] NESBET, R.K., J. math. Phys. 2 (1961) 701.

[58] NESBET, R.K., Phys. Rev. 156 (1967) 99.
[59] MOTT, N.F., MASSEY, H.S.W., The Theory of Atomic Collisions, Oxford University Press, New York (1965).
[60] KOHN, W., Phys. Rev. 74 (1948) 1763.
[61] HULTHEN, L., Ark. Mat. astron. Fysik 35A 25 (1948); RUBINOW, S.I., Phys. Rev. 98 (1955) 183; WILLIAMS, K.L., Proc. Phys. Soc. (London) 91 (1967) 807.
[62] SCHWARTZ, C., Ann. Phys. (New York) 16 (1961) 36.
[63] NESBET, R.K., Phys. Rev. 175 (1968) 134; 179 (1969) 60.
[64] ARMSTEAD, R.L., Phys. Rev. 171 (1968) 91.
[65] LYONS, J.D., NESBET, R.K., J. comput. Phys. 4 (1969) 499; HARRIS, F.E., MICHELS, H.H., J. comput. Phys. 4 (1969) 579.
[66] BOTTCHER, C., J. comput. Phys. 6 (1970) 237; RAMAKER, D.E., J. math. Phys. 13 (1922) 161.
[67] NESBET, R.K., J. comput. Phys. 8 (1971) 483.

ACCURATE CALCULATION OF CROSS-SECTIONS FOR NON-REACTIVE MOLECULAR COLLISIONS

W.A. LESTER, Jr.
IBM Research Laboratory,
San José, Calif.,
United States of America

Abstract

ACCURATE CALCULATION OF CROSS-SECTIONS FOR NON-REACTIVE MOLECULAR COLLISIONS.

Computational considerations encountered in the accurate calculation of cross-sections for rotational and vibrational excitation of molecules by heavy-particle collision are presented and discussed in the context of the model problem of scattering by a rigid rotator. The nature of the problem is delineated: calculation of the potential-energy hypersurface on which the scattering takes place and the computation of cross-sections using the computed hypersurface. Computational aspects of both parts of the problem are exemplified by recent calculations of a potential-energy surface and rotational energy transfer cross-sections for the $Li^+ - H_2$ system.

1. INTRODUCTION

Computation is playing an increasingly significant role in the elucidation of physical phenomena on the microscopic scale. Elsewhere in this volume, Nesbet discusses computational methods of atomic and molecular physics. This is such a broad topic, however, that a useful purpose might be served here by focusing briefly on the origin of some of the more important computational problems and methods of solution that have arisen in a branch of molecular physics that has undergone increased activity in recent years: the determination of cross-sections for non-reactive molecular collisions. In the present paper, attention is limited to inelastic encounters; Bernstein [1] has given an excellent review of elastic scattering.

Recent experimental advances [2] in molecular-beam scattering [3], ultrasonic dispersion [4], and optical fluorescence [5] have begun to yield cross-sections for rotational and vibrational energy transfer by heavy-particle collision between selected initial and final quantum states of target molecules. To assist the interpretation of these experiments, efforts have been initiated by a number of groups to compute accurately cross-sections for these processes [6]. Extensive discussions of computational methods in molecular scattering have very recently appeared [7].

Cross-sections of interest here are of importance in a variety of physical processes. These include cooling of interstellar space [8], line broadening [9], relaxation phenomena in gases [10], and chemical reactions [11].

The goal of scattering studies is to understand the laws of interaction between colliding systems. There are basically two approaches to the problem. One is to start with experimental results and then, based on the interpretation of measurements, infer the form and magnitude of the interaction. This procedure is naturally limited by the validity of the assumptions made and is intertwined with the unfolding of distributions

necessary to yield information on individual scattering events. The other approach is to compute cross-sections directly from the basic laws of physics. The latter point of view is followed here and forms the basis for the computational aspects that follow. Of course, the test of calculations is agreement with precise measurements.

2. NATURE OF THE PROBLEM

A. Potential-energy surfaces [12]

The ab-initio calculation of atomic and molecular properties requires, in general, the solution of the Schrödinger equation which in the time-independent form is

$$H\Psi = E\Psi \tag{1}$$

Denoting the masses, charges, and co-ordinates of the particles as follows (in atomic units),

Particle	Mass	Charge	Co-ordinates
nucleus α	M_α	Z_α	$\vec{R}_\alpha(X_\alpha, Y_\alpha, Z_\alpha)$
electron i	1	-1	$\vec{r}_i(x_i, y_i, z_i)$

the Hamiltonian (in the absence of external fields) may be written

$$H = -\frac{1}{2}\sum_{\alpha=1}^{N}\frac{\Delta_\alpha}{M_\alpha} - \frac{1}{2}\sum_{i=1}^{n}\Delta_i + V \tag{2}$$

where

$$V = -\sum_{i,\alpha}\frac{Z_\alpha}{r_{i\alpha}} + \sum_{i<j}\frac{1}{r_{ij}} + \sum_{\alpha<\beta}\frac{Z_\alpha Z_\beta}{R_{\alpha\beta}} \tag{3}$$

and the denominators of the terms of Eq. (3) are interparticle distances. The terms on the right-hand side of Eq. (2) represent the total nuclear and electronic kinetic energies, and the total potential energy, respectively. Those on the right-hand side of Eq. (3) designate the total nuclear-electronic, electronic-electronic, and nuclear-nuclear potential energies, respectively.

Because of mass differences, nuclear velocities are small compared with electronic velocities. This forms the basis of the clamped-nuclei approximation or Born-Oppenheimer separation of nuclear and electronic motion. In this approximation, the Schrödinger equation for electronic motion,

$$H^{el}\Phi^{el} = E^{el}\Phi^{el} \tag{4}$$

is obtained by deleting the total nuclear kinetic energy term from Eq. (2) so that the electronic Hamiltonian is simply

$$H^{e\ell} = -\frac{1}{2}\sum_{i=1}^{n}\Delta_i + V \tag{5}$$

In Eq. (4), $E^{e\ell}$ is a function of the nuclear co-ordinates

$$E^{e\ell} = E^{e\ell}(\vec{R}_1, \vec{R}_2, \ldots, \vec{R}_N) \equiv E^{e\ell}(\vec{R}) \tag{6}$$

and

$$\Phi^{e\ell} = \Phi^{e\ell}(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n, \vec{R}_1, \vec{R}_2, \ldots, \vec{R}_N) \equiv \Phi^{e\ell}(\vec{r}, \vec{R}) \tag{7}$$

For a particular system and specified nuclear geometry, one can, in principle, solve Eq. (4) and determine the eigenvalues $E^{e\ell}$. These solutions, together with solutions for other nuclear arrangements, define a set of potential-energy surfaces. For applications to date, only the ground-state surface (determined by the lowest eigenvalue for each nuclear arrangement) or a few of the lowest-lying potential-energy surfaces have been of immediate interest.

The potential energy for nuclear motion is defined as the electronic energy $E^{e\ell}(\vec{R})$ (which is not strictly electronic since it contains the internuclear repulsion term) so that the nuclear Schrödinger equation may be written

$$\left\{-\frac{1}{2}\sum_{\alpha=1}^{N}\frac{\Delta_\alpha}{M_\alpha} + E^{e\ell}(\vec{R})\right\}\Phi^{nucl}(\vec{R}) = E\Phi^{nucl}(\vec{R}) \tag{8}$$

Note that the zero of potential energy is the infinitely separated nuclei.

In non-reactive molecular scattering, the physically appropriate zero of energy is the constituent molecules at infinite separation. Therefore, $E^{e\ell}(\vec{R})$ in Eq. (8) is replaced by

$$V(\vec{R}) = E^{e\ell}(\vec{R}) - E^{e\ell}(\infty) \tag{9}$$

where $E^{e\ell}(\infty)$ designates the electronic energy for infinitely separated fragments and intramolecular distances set equal to those of the isolated fragments. In addition, a term describing the internal motion of the separated molecules must be included. Thus the nuclear Hamiltonian has the form

$$H^{nucl} = -\frac{1}{2}\sum_{\alpha=1}^{N}\frac{\Delta_\alpha}{M_\alpha} + H_{int} + V(\vec{R}) \tag{10}$$

where H_{int} is the total Hamiltonian for internal motion of the fragments. See Section 2.B for the explicit form of expression (10) for rotational excitation of diatomic molecules by atom impact.

As sketched above, the calculation of cross-sections breaks down into two parts: (1) the calculation of electronic eigenvalues as a function of nuclear geometry of the super-molecule (composed of the collision fragments) which defines the potential-energy surface (or surfaces if higher roots are retained), and (2) calculation of cross-sections using the pre-determined potential-energy surface.

The computation of potential-energy surfaces from the electronic Schrödinger equation is traditionally the task of the quantum chemist. It is not possible to cover here the extensive developments that have occurred in methods of solution for electronic energies. The interested reader is referred to a number of books [13] and review articles [14].

Because of the aim of computing accurate cross-sections, interest here is in potential-energy surface calculations that are as exact as practicable. Such calculations are referred to as ab-initio or non-empirical [13c]. For semi-empirical procedures, see Karplus [12]. Purely empirical methods are discussed by Johnston [15] and Bunker [16]. A wide variety of methods and techniques has been employed in seeking accurate energy eigenvalues. However, almost without exception [17] energy surfaces have been determined by employing wave functions computed by the expansion method [13c, 14d].

In this method, $\Phi^{e\ell}$ is written as a linear combination of configurations (normalized Slater determinants),

$$\Phi^{e\ell}(\vec{r}, \vec{R}) = \sum_{\nu} c_\nu \psi_\nu(\vec{r}, \vec{R}, \vec{\lambda}) \tag{11}$$

where $\vec{r}$ and $\vec{R}$ have been defined previously, $\vec{\lambda}$ is a set of parameters that determine the functions ψ_ν, and c_ν are linear expansion coefficients. The coefficients c_ν and parameters $\vec{\lambda}$ can be determined from the variational principle by minimizing the functional

$$E^{e\ell} = \frac{\langle \Phi^{e\ell} | H^{e\ell} | \Phi^{e\ell} \rangle}{\langle \Phi^{e\ell} | \Phi^{e\ell} \rangle} \tag{12}$$

This leads to an infinite system of linear equations

$$\sum_{\mu} (H_{\mu\nu} - E^{e\ell} S_{\mu\nu}) c_\nu = 0 \tag{13}$$

and

$$H_{\mu\nu} = \langle \psi_\mu | H^{e\ell} | \psi_\nu \rangle \tag{14}$$

Here

$$S_{\mu\nu} = \langle \psi_\mu | \psi_\nu \rangle \tag{15}$$

where both $H_{\mu\nu}$ and $S_{\mu\nu}$ are, generally, functions of $\vec{\lambda}$. Equation (13) has normalizable solutions only for values of the energy $E^{e\ell}$ that are roots of the secular equation

$$\left| H_{\mu\nu} - E^{e\ell} S_{\mu\nu} \right| = 0$$

The principal limitations in obtaining accurate solutions are due to the following computational difficulties:

(a) Integral evaluation [18]. Significant advances are being made in the calculation of multi-centre integrals and associated data handling which arise from expansions of the functions ψ_ν in linear combinations of basis functions [13b, c, 14d]. The difficulties arise, on the one hand, from the mathematical complexity of the integrals that must be computed, e.g. Slater-type functions [13b], and, on the other hand, from the large number of integrals that must be treated for satisfactory accuracy for other choices of functions, e.g. Gaussian-type functions [13b].

(b) Scope of computations. In collision problems, knowledge is required of computed potential-energy surfaces from large to small intermolecular distances as a function of relative orientation. This means, in general, that many energy calculations must be performed in order to gain a representation of the intermolecular interaction adequate for scattering studies. In the non-overlap region one may often take advantage of perturbation theory of intermolecular forces [19] to reduce the number of computations ordinarily needed.

(c) Accuracy requirements. At the present time, it is not known how errors in potential energy surfaces are quantitatively reflected in the errors of derived cross-sections. Of course, qualitative considerations such as the predominant dependence of rotational cross-sections on the long-range region and vibrational energy transfer on short-range collisions can be tested on comparatively crude surfaces.

(d) Lack of direct checks. Since potential-energy surfaces are not observables, the determination of the validity of computed surfaces is a more difficult problem than the establishment of errors in calculated properties of stable systems. This places a heavy burden on numerical methods and procedures employed in the evaluation of molecular interactions to insure accurate results.

At the present time, variational calculations of potential energy surfaces for non-reactive systems can be usefully distinguished by the number of terms retained in Eq. (11). The single-term calculation which reaches its ultimate form in the Hartree-Fock approximation [13, 14], corresponding to the optimum specification of the parameters $\vec{\lambda}$, is the one most commonly carried out. Such calculations for individual molecules are now essentially routine and a number of computer programs following this method has been widely disseminated [20]. However, relatively few calculations of intermolecular surfaces have been performed in this approximation [21].

Except for calculations on purely hydrogen-containing systems (H_3^+, H_3, H_3^-, H_4) [20, 22], few calculations beyond the single-configuration level have appeared. Such computations are not as yet routine, but a number of efforts are in progress [23].

B. Cross-sections

The use of potential energy surfaces in the determination of molecular dynamics may be exemplified by the consideration of rotational excitation of diatomic molecules by atom impact. The scattering formalism described below possesses a number of features common to treatments of scattering

in other areas of physics that are easily recognizable if the nature of the particles and origin of angular momentum coupling are appropriately changed. The present development, due to Arthurs and Dalgarno [24], follows from a treatment of nuclear scattering theory [25].

The theory is applicable to scattering of a structureless particle (e. g. a ^{1}S atom) by a rigid rotator (diatomic molecule) of moment of inertia I. The incident relative kinetic energy is restricted to be sufficiently low that there is no excitation of vibrational and electronic degrees of freedom. For simplicity, consideration is restricted to $^1\Sigma$ states.

The nuclear Hamiltonian for an atom A and molecule BC takes the form, see (10),

$$H^{\text{nucl}} = -\frac{1}{2}\left(\frac{\Delta_A}{M_A} + \frac{\Delta_B}{M_B} + \frac{\Delta_C}{M_C}\right) + H_{\text{rot}} + V(\vec{R}_A, \vec{R}_B, \vec{R}_C) \tag{16}$$

where H_{rot} replaces H_{int}. In centre-of-mass co-ordinates, it is convenient to express formula (16) as follows:

$$H^{\text{nucl}} = -\frac{1}{2\mu}\Delta_\rho - H_{\text{rot}} + V(\vec{\rho}, \vec{r}) \tag{17}$$

where the first term is the kinetic-energy operator for relative motion in which

$$\mu = M_A(M_B + M_C)/(M_A + M_B + M_C) \tag{18}$$

Here, the potential-energy surface V is expressed in terms of $\vec{\rho}$, the position of incident particle A relative to the centre of mass of molecule BC, and $\vec{r}$, the co-ordinate specifying the relative positions of B with respect to C. In addition, it should be noted that

$$[H_{\text{rot}} - j(j+1)/2I]\, Y_{jm_j}(\hat{r}) = 0 \tag{19}$$

where the functions Y_{jm_j} are the familiar spherical harmonics and $\hat{r}$ specifies the orientation of the intermolecular axis.

Basis functions are constructed that are eigenfunctions of $\vec{J}^2$ and J_z where $\vec{J}$ is the total angular momentum compounded from the rotational angular momentum $\vec{j}$ and the orbital angular momentum $\vec{\ell}$, i. e. $\vec{J} = \vec{j} + \vec{\ell}$ with associated projections on the z-axis denoted by M, m_j, and m_ℓ, viz.

$$\mathscr{Y}^{JM}_{j\ell}(\hat{\rho}, \hat{r}) = \sum_{m_j, m_\ell} (j\ell\, m_j m_\ell \,|\, j\ell\, JM)\, Y_{jm_j}(\hat{r})\, Y_{\ell m_\ell}(\hat{\rho}) \tag{20}$$

where the coefficients are the well-known Clebsch-Gordan coefficients. The nuclear wave function is expanded in this set,

$$\Phi^{\text{nucl}}(\vec{R}) \equiv \Phi_j(\vec{\rho}, \hat{r}) = \sum_{Jj'\ell'} \rho^{-1}\, u^{Jj\ell}_{j'\ell'}(\rho)\, \mathscr{Y}^{JM}_{j'\ell'}(\hat{\rho}, \hat{r}) \tag{21}$$

Substitution of expression (21) into the nuclear Schrödinger equation with Hamiltonian (17) and carrying out the usual further reductions yields the system of coupled equations

$$\left[\frac{d^2}{d\rho^2} + k_{j'}^2 - \frac{\ell'(\ell'+1)}{\rho^2}\right] u_{j'\ell'}^{Jj\ell}(\rho) = \sum_{j''\ell''} U_{j'\ell',j''\ell''}^{J}(\rho)\, u_{j''\ell''}^{Jj\ell}(\rho) \tag{22}$$

where

$$k_{j'}^2 = 2\mu\,[E - j'(j'+1)/2I] \tag{23}$$

and

$$U_{j'\ell',j''\ell''}^{J}(\rho) = 2\mu \iint \mathscr{Y}_{j'\ell'}^{JM} V \mathscr{Y}_{j''\ell''}^{JM} d\hat{\rho}\, d\hat{r} \tag{24}$$

Note that E consists of contributions from the rotator initially in state j and the kinetic energy of relative motion. At sufficiently large r such that the intermolecular potential is negligible, the functions $u_{j'\ell'}^{Jj\ell}(r)$ may be written in terms of the defining relation for the scattering matrix:

$$u_{j'\ell'}^{Jj\ell}(\rho) \sim \delta_{j'j}\,\delta_{\ell'\ell} \exp\left[-i\left(k_j\rho - \frac{\ell\pi}{2}\right)\right] - (k_j/k_{j'})^{\frac{1}{2}} \underset{\sim}{S}^{J}(j'\ell', j\ell) \times \exp\left[i\left(k_{j'}\rho - \frac{\ell'\pi}{2}\right)\right] \tag{25}$$

where $\underset{\sim}{S}$ is diagonal in J and independent of M.

Asymptotically, before collision, $\Phi_j(\vec{\rho}, \hat{r})$ is a plane wave which in the absence of external fields may be chosen incident along the z axis ($\theta = 0$),

$$\Phi_j(\vec{\rho}, \hat{r}) = \frac{i\pi^{\frac{1}{2}}}{k_j} \sum_{J=0}^{\infty} \sum_{M=-J}^{J} \sum_{\ell=|J-j|}^{J+j} (j\ell\, m_j 0 \,|\, j\ell\, JM)\, i^{\ell} (2\ell+1)^{\frac{1}{2}}\, \rho^{-1} \times \exp\left[-i\left(k_j\rho - \frac{\ell\pi}{2}\right)\right] - \exp\left[i\left(k_j\rho - \frac{\ell\pi}{2}\right)\right] \mathscr{Y}_{j\ell}^{JM}(\hat{\rho}, \hat{r}) \tag{26}$$

After the scattering event, the long-range behaviour of the wave function is written

$$\Phi_j(\vec{\rho}, \hat{r}) \sim \exp(i\vec{k}_j \cdot \vec{\rho})\; Y_{jm_j}(\hat{r}) + \sum_{j'} \frac{i}{k_j} \left(\frac{k_j}{k_{j'}}\right)^{\frac{1}{2}} \rho^{-1} \exp(ik_{j'}r) \times \sum_{m_{j'}=-j'}^{j'} q_{j'm_{j'},\, jm_j}(\hat{\rho})\, Y_{jm_j}(\hat{r}) \tag{27}$$

where

$$q_{j'm_{j'},\,jm_j}(\hat{\rho}) = \sum_{J=0}^{\infty} \sum_{M=-J}^{J} \sum_{\ell=|J-j|}^{J+j} \sum_{\ell'=|J-j'|}^{J+j'} \sum_{m_\ell=-\ell'}^{\ell'} i^{\ell-\ell'} \pi^{\frac{1}{2}}$$

$$\times (2\ell+1)^{\frac{1}{2}} (j\ell\, m_j 0 \,|\, j\ell\, JM)\ (j'\ell'\, m_{j'} m_{\ell'} \,|\, j'\ell'\, JM)$$

$$\times (\delta_{j'j}\delta_{\ell'\ell} - S^J_{j'\ell',j\ell})\ Y_{\ell' m_{\ell'}}(\hat{\rho}) \qquad (28)$$

Total cross-sections are obtained from the ratio of the flux of inelastically scattered particles to the incident flux. For excitation from channel (jm_j) to channel $(j'm_{j'})$, these cross-sections are given by

$$\sigma_{j'm_{j'},\,jm_j} = k_j^{-2} \left| q_{j'm_{j'},\,jm_j}(\hat{\rho}) \right|^2 d\hat{\rho} \qquad (29)$$

Averaging expression (29) over m_j and summing over $m_{j'}$, yields the total cross-section for $j \to j'$ transitions

$$\sigma_{j',j} = \frac{\pi}{(2j+1)k_j^2} \sum_{J=0}^{\infty} \sum_{\ell=|J-j|}^{J+j} \sum_{\ell'=|J-j'|}^{J+j'} (2J+1) \left| \delta_{j'j}\delta_{\ell'\ell} - S^J_{j'\ell',j\ell} \right|^2 \qquad (30)$$

C. Scope of the computational problem [26]

Equation (30) provides a useful starting point for examining the magnitude of the computational effort. It is clear that for a $j \to j'$ transition at energy k_j^2, scattering matrices are required for each term of a large sum (in principle, infinite). In addition, the evalution of the scattering matrix for each J involves the solution of an, in principle, infinite system of ordinary second-order coupled differential equations, Eq. (22).

The evaluation of Eq. (30) from known scattering matrices is trivial; the practical difficulty lies in solving the coupled equations to yield scattering matrices for the requisite number of J values needed to establish convergence of Eq. (30), and, in addition, to solve systems of equations that are large enough to permit quantitative assessment of errors in $\underset{\sim}{S}^J$.

By far the greatest computing time is spent in the point-by-point integration of the system of differential equations. Allison [27] has recently compared the novel approach of Gordon [6g], De Vogelaere's method [6c, 7b, 28] and the iterative Numerov procedure. He concludes that for solving large sets of coupled second-order ordinary linear differential equations for a set of energies or associated parameters using potential functions that are known only to a few per cent accuracy, full consideration should be given to Gordon's method. On the other hand, he states that if the potential functions are known more accurately and higher precision is required such as for the evaluation of differential cross-sections, or if it is not convenient to repeat the calculation immediately for different energies, then the established step-by-step methods are still very satisfactory and of comparable speed.

3. PRELIMINARY RESULTS FOR ROTATIONAL EXCITATION OF H_2 BY Li^+

Recently a number of experimental groups have reported ion-molecule collision experiments in which the energy loss of the scattered ion is observed [3b]. Such measurements provide for the first time direct information on the energy and angular dependence of cross-sections for rotational and vibrational energy transfer. To assist the interpretation of experiments concerned with vibrational excitation of H_2 by Li^+ collisions, calculations near the Hartree-Fock limit were carried out at approximately 210 nuclear arrangements to determine a potential-energy surface in sufficient detail for accurately computing cross-sections for rotational-vibrational excitation [21b, c].

Preparatory to computations of these cross-sections, the region of the potential energy surface for fixed H_2 bond length [21b] has been used in calculations of integral cross-sections for pure rotational excitation following the formalism outlined above. A summary of these results follows.

Figure 1 presents a representative sampling of computed interaction energies for the equilibrium H_2 bond length (chosen as 1.4 a.u.). These

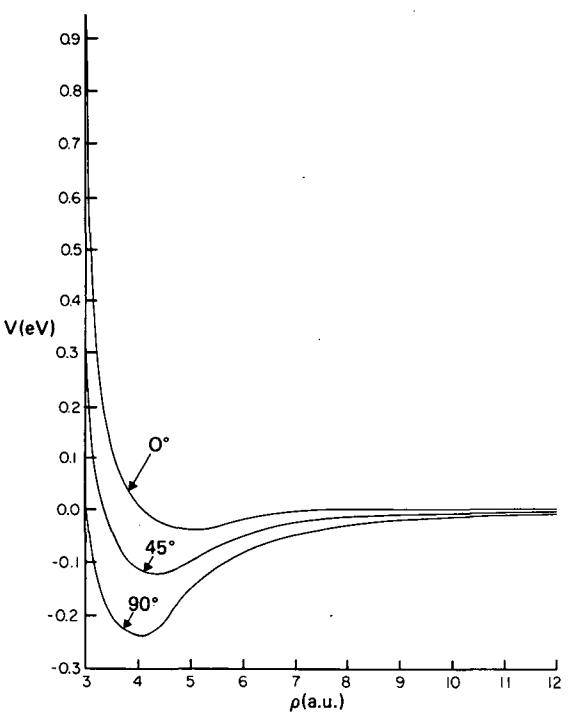


FIG. 1. $Li^+ - H_2$ interaction potential for R_{HH} = 1.4 a.u. for 0° (collinear), 45°, and 90° approaches.

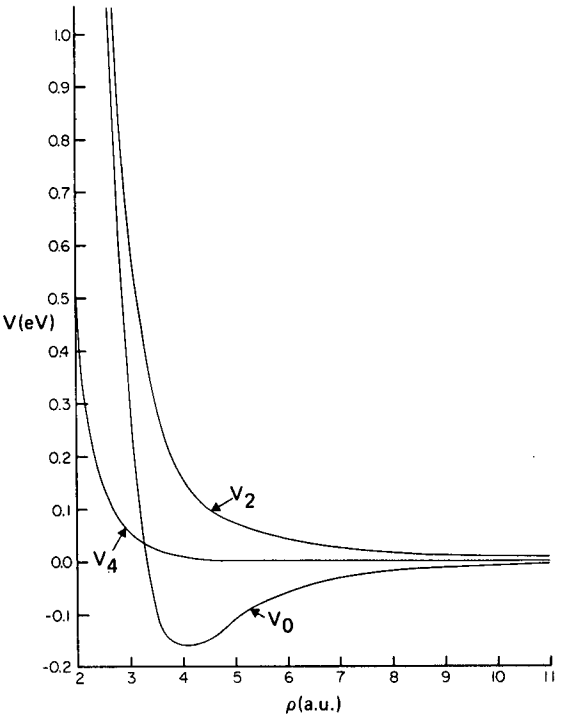


FIG. 2. Radial components of the potential-energy function for $Li^+ - H_2$ for $R_{HH} = 1.4$ a.u. See Eq.(31).

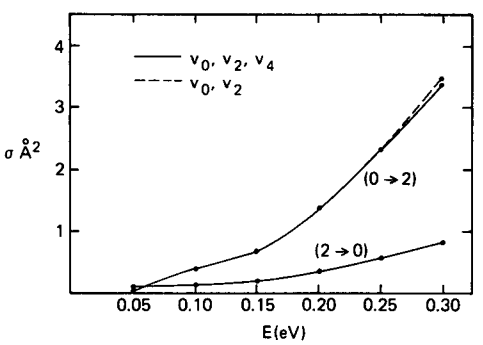


FIG. 3. Integral cross-sections for the $0 \rightarrow 2$ and $2 \rightarrow 0$ rotational transitions in H_2 employing the full potential-energy function and the leading two terms. See Eq.(31).

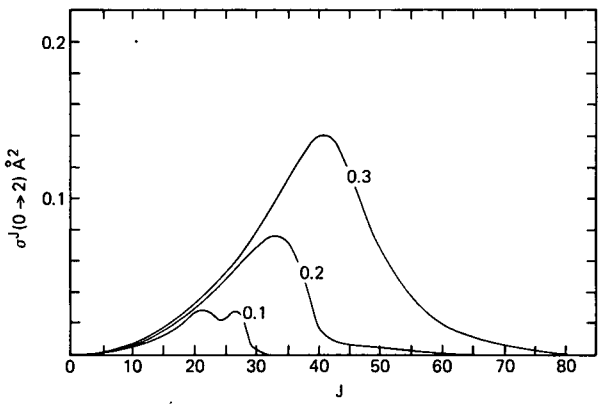


FIG. 4. Partial wave contributions to integral cross-sections for the 0 → 2 rotational excitation of H_2 by Li^+ collision at selected centre-of-mass energies (in electron volts).

results together with those not displayed were fitted[21b] to the following analytical form to facilitate evaluation of coupling-matrix elements, i.e. (in electron volts)

$$V(\vec{\rho}, \hat{r}) = \sum_{\ell=0}^{4} v_\ell(\rho) P_\ell(\hat{\rho} \cdot \hat{r}), \qquad 2 \leq \rho \leq 12 \text{ a.u.}, \quad \ell \text{ even} \tag{31}$$

where

$$v_0(\rho) = 599.3367 \exp(-2.03\rho) + 84.8930 \exp(-2.06\rho) + 6.6183\rho^{-3} - 118.6957\rho^{-4} \tag{31a}$$

$$v_2(\rho) = -153.946 \exp(-2.03\rho) + 366.0221 \exp(-2.06\rho) + 11.9284\rho^{-3} - 19.0837\rho^{-4} \tag{31b}$$

and

$$v_4(\rho) = -365.08443 \exp(-2.03\rho) + 408.99089 \times \exp(-2.06\rho) - 0.4614379\rho^{-3} + 4.15251\rho^{-4} \tag{31c}$$

The radial components of the potential function are displayed in Fig. 2. Based on the dissociation energy of H_2 and the energy of Li^+, the interaction potential is presumed to be approximately 0.01 eV from the Hartree-Fock limit.

Figure 3 shows integral cross-sections for the transition j = 0 → j' = 2 obtained from solutions of Eqs (22) retaining, at most, four channels, and j = 2 → j' = 0 determined by detailed balancing considerations. Figure 4

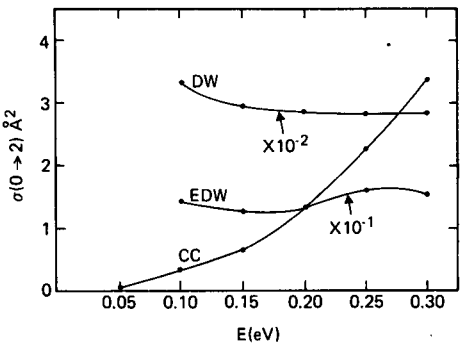


FIG. 5. Comparison of integral cross-sections for the 0 → 2 rotational transition in H_2 by Li^+ impact computed by the DW (distorted-wave approximation), EDW (exponential-distorted-wave approximation based on Seaton, Ref.[30]), and CC (close-coupling method).

reveals the partial wave contributions to the integral cross-sections. Finally, in Fig. 5, the coupled-equations solutions are compared with various perturbation approaches; the distorted-wave approximation [24, 29] and a method suggested by Seaton [30]. Note the striking poorness of the approximation methods for the present ion-molecule system which is in distinct contrast to results of previous studies of neutral atom-H_2 rotational energy transfer studies [6a, b].

Although the close-coupled results are a marked improvement over the approximate methods (due, in part, to the fact that they numerically satisfy certain conservation conditions for the $\underset{\sim}{S}^J$), the accuracy of these results has not yet been ascertained. Calculations are in progress to determine this by testing the sensitivity of the cross-sections to the target state expansion. In addition, the rigid rotator approximation has been found to be a rather poor approximation for the H + H_2 system [31]. This point will also be tested by similar computations employing, however, the vibrationally averaged ab-initio surface [21c].

REFERENCES

[1] BERNSTEIN, R.B., Adv. chem. Phys. 12, Wiley, New York (1966) 75.

[2] An excellent review of the capabilities of various rotational and vibrational relaxation experiments has been given by GORDON, R.G., KLEMPERER, W., STEINFELD, J.I., A. Rev. phys. Chem. 19 (1968) 215.

[3] (a) BERNSTEIN, R.B., in Potential Energy Surfaces Chem. (Proc. Conf. San José, California, 1971) RA18, IBM Corp., 27 and references contained therein;
(b) TOENNIES, J.P., ibid., 34;
(c) MORAN, T.F., PETTY, F., TURNER, G.S., Chem. Phys. Letts 9 (1971) 379 and references contained therein.

[4] PRANGSMA, G.J., HEEMSKERK, J.P.J., KNAPP, H.F.P., BEENAKKER, J.J.M., Physica 50 (1970) 433 and references contained therein.

[5] MOORE, C.B., in Potential Energy Surfaces Chem. (Proc. Conf. San José, California, 1971) RA18, IBM Corp., (1971) 40 and references contained therein; BERGMANN, K., DEMTRÖDER, Z., Phys. 243 (1971) 1 and references contained therein.
[6] (a) ALLISON, A.C., DALGARNO, A., Proc. phys. Soc. (London) 90 (1967) 609;
(b) JOHNSON, B.R., SECREST, D., J. chem. Phys. 48 (1968) 4682;
(c) LESTER, W.A., Jr., BERNSTEIN, R.B., J. chem. Phys. 48 (1968) 4896;
(d) ERLEWEIN, W., VON SEGGERN, M., TOENNIES, J.P., Z. Phys. 211 (1968) 35;
(e) KINSEY, J.L., RIEHL, J.W., WAUGH, J.S., J. chem. Phys. 49 (1968) 5269;
(f) BURKE, P.G., SCRUTTON, D., TAIT, J.H., TAYLOR, A.J., J. Phys. B 2 (1969) 1155;
(g) GORDON, R.G., J. chem. Phys. 51 (1969) 14;
(h) SAMS, W.N., KOURI, D.J., J. chem. Phys. 51 (1969) 4809 and subsequent papers in J. chem. Phys.;
(i) FREMEREY, H., A Computer Program for the Calculation of Differential Scattering Cross Sections for Coupled Vibrational and Rotational Excitation of Diatomic Molecules by Atom Collision, Especially Several Preliminary Calculations for the Vibrational Excitation of H_2 by He Collision, Thesis, Physical Institute of the University of Bonn (1970), (in German).
[7] (a) GORDON, R.G., Methods in Computational Physics, 10 (1971) 81;
(b) LESTER, W.A., Jr., ibid., 10 (1971) 211;
(c) SECREST, D., ibid., 10 (1971) 243.
[8] FIELD, G.B., SOMERVILLE, W.B., DRESSLER, K., A. Rev. Astron. Astrophys. 3 (1966) 207.
[9] BIRNBAUM, G., Adv. chem. Phys. 12 (1967) 487.
[10] HERZFELD, K.F., LITOVITZ, T.A., Absorption and Dispersion of Ultrasonic Waves, Academic Press, New York (1959).
[11] SPICER, L.D., RABINOVITCH, B.S., A. Rev. phys. Chem. 21 (1970) 349.
[12] For a more extensive survey, see KARPLUS, M., Proc. Int. School of Physics, Enrico Fermi, Course 44, Academic Press, New York (1970) 320.
[13] (a) EYRING, H., WALTER, J., KIMBALL, G., Quantum Chemistry, Wiley, New York (1944);
(b) SLATER, J.C., Quantum Theory of Molecules and Solids 1, Electronic Structure of Molecules, McGraw-Hill, New York (1963);
(c) MCWEENY, R., SUTCLIFFE, B.T., Methods of Molecular Quantum Mechanics, Academic Press, New York (1969).
[14] (a) KOTANI, M., OHNO, K., KAYAMA, K., Encyclopedia of Physics 37, part 2, Springer-Verlag, Berlin (1961) 1;
(b) DAVIDSON, E., Physical Chemistry, Academic Press, New York (1969) 113;
(c) PAUNZ, R., Physical Chemistry, Academic Press, New York (1969) 185-247 and references contained therein;
(d) MCLEAN, A.D., In Potential Energy Surfaces Chem. (Proc. Conf. San José, Calif., 1971) (1971) 87.
[15] JOHNSTON, H.S., Gas Phase Reaction Rate Theory, Ronald Press, New York (1966).
[16] BUNKER, D.L., Proc. Int. School of Physics, Enrico Fermi, Course 44, Academic Press, New York (1970) 355-371.
[17] For an exception, see CONROY, H., ibid., 349.
[18] A review of the integral problem has been given by HUZINAGA, S., Prog. theor. Phys. Suppl. 40 (1967) 52.
[19] (a) HIRSCHFELDER, J.O., CURTISS, C.F., BIRD, R.B., Molecular Theory of Gases and Liquids, Wiley (1964) Ch.13;
(b) BUCKINGHAM, A.D., Adv. chem. Phys. 12 (1967) 107;
(c) MARGENAU, H., KESTNER, N., Theory of Intermolecular Forces, Pergamon Press (1969).
[20] For a recent listing, see KRAUSS, M., in Potential Energy Surfaces Chem. (Proc. Conf. San José, California, 1971) RA 18, IBM Corp., 6.
[21] For a recent survey of potential-energy surface calculations, see KRAUSS, M., Ann. Rev. Phys. Chem. 21 (1970) 39. Restricted Hartree-Fock calculations appearing since this review include:
(a) GORDON, M.D., SECREST, D., J. chem. Phys. 52 (1970) 120 (also includes configuration interaction computations);
(b) LESTER, W.A., Jr., ibid. 53 (1970) 1511;
(c) LESTER, W.A., Jr., ibid. 54 (1971) 3171;
(d) LESTER, W.A., Jr., IBM J. Res. Develop. 15 (1971) 222.
[22] LIU, B., Int. J. quant. Chem. (to appear).
[23] HOHEISEL, K., STAEMLER, V., KUTZELNIGG, W., Private communication, $Li^+ + H_2$; BENDER, C.F., SCHAEFER, H.F., Private communication, $F + H_2$; WAHL, A.C., private communication, $Li + H_2$.

[24] ARTHURS, A.M., DALGARNO, A., Proc. R. Soc. (London) A256 (1960) 540.
[25] BLATT, J.M., BIEDENHARN, L.C., Rev. mod. Phys. 24 (1952) 258.
[26] For extensive discussions, see Refs [6, 7].
[27] ALLISON, A.C., J. Comput. Phys. 6 (1970) 378.
[28] DE VOGELAERE, R., J. Res. Nat. Bur. Stand. 54, 119 (1955); LESTER, W.A., Jr., J. Comput. Phys. 3 (1968) 322.
[29] ROBERTS, C.S., Phys. Rev. 131 (1963) 209.
[30] SEATON, M.J., Proc. phys. Soc. (London) 77 (1961) 174.
[31] WOLKEN, G., KARPLUS, M., MILLER, W.H., to be published.

PART IV: SYMBOLIC ANALYSIS

COMPARATIVE SURVEY OF PROGRAMMING LANGUAGES*

J. A. CAMPBELL
Department of Physics, Center for Particle Theory
and
Department of Computer Sciences,
University of Texas,
Austin, Tex.,
United States of America

Abstract

COMPARATIVE SURVEY OF PROGRAMMING LANGUAGES.

The range of programming languages available to the physicist is reviewed. Knowledge of FORTRAN is assumed. Three major classes of languages are distinguished: languages for numerical processing, languages for symbolic processing, and string-processing languages. In that order, the chief representatives of the three classes for the purpose of discussion are FORTRAN, LISP and SNOBOL. Other languages of each type are outlined to show how it is possible to make approaches to certain computing operations which are different from the approaches used by the chief languages of each class, and sometimes better. PL/I is mentioned as a language which is intended to combine attributes of all three classes. Two further classes of less general interest are acknowledged: languages for simulations (e. g. SIMULA 67) and systems for the management of large data-bases (e. g. RFMS). Operations or structural properties which are common to all or most of the languages are set out individually and their treatment by different languages is examined on a comparative basis. Several programming problems are given, together with their solutions in different languages, side by side. Including FORTRAN, 13 languages are discussed.

1. INTRODUCTION: TYPES OF LANGUAGES

The only previous survey of programming languages for physicists which I know (Ref. [B1]) begins with FORTRAN and other languages where purely numerical operations are expressed in a fairly natural way and brings in various non-numerical ideas gradually without calling special attention to what is going on. In an ideal world, this is the ideal way to talk about the subject. In practice, though, one tends to learn FORTRAN first and to believe that FORTRAN = numerical analysis = computing. Therefore, a mental barrier grows up against the idea of non-numerical applications of computers. Any programmer-physicist who has first acquired such a barrier and has then jumped it is aware of a change of outlook which is sudden, not gradual. For that reason, I shall make the distinction between classes of languages in this paper quite sharp.

Class distinctions may be fun, as people who have used them as hooks on which to hang learned articles in sociological journals occasionally tell us, but other more solemn people are dedicated to the idea that such things should be abolished. In computing, the person in this latter category (class?) normally sits in the front row of the audience at lectures and

* Preparation supported in part by National Science Foundation grant GJ-1069.

responds to any description of the use of an exotic language for solving a non-numerical problem with the objection, "But you can do that in FORTRAN". To this, there is no real answer except the generalization, "You can do anything in any language". The point, however, is to use the language best suited to the job. One first identifies the type of computation needed for a particular problem, and then chooses the language designed for that type of computation in which the easiest solutions to the problem can be written. The word "easiest" is the starting point for many debates, of course, but one way to approach the question is to say that the easiest program to write, use, and explain to somebody else is the program which expresses the algorithm for S as nearly as possible (in logical terms) to the expression of the algorithm in the usual mixture of mathematical notation and one's own native language[1].

This paper is organized around the assumption that there are five recognizably different types of programming language:

(1) Languages for numerical computation (e. g. FORTRAN, ALGOL, FRED, APL)
(2) Languages for symbolic computation (e. g. LISP, POP-2, REDUCE, L[6])
(3) String-processing languages (e. g. SNOBOL, CONVERT)
(5) Languages for simulations (e. g. SIMULA 67, GPSS)
(6) Languages for management of large data bases (e. g. RFMS and perhaps COBOL).

In an introductory course for physicists, there is a case for concentration on the first three types at the expense of the other two, because of the greater general interest in the problems which these three can solve. This is not to say that physical scientists have ignored types 5 and 6: simulation of complex experiments in a language such as SIMULA 67 before the experiments are built up may even save money, and W. H. Jefferys (Astronomy Department, Univ. of Texas) has coded de Vaucouleur's Reference Catalogue of 2597 Bright Galaxies in a form in which RFMS can be used to ask scientific questions like, "How many galaxies in the catalogue are spiral and have values of the index U-B between x and y?".

Observant readers will notice that there is no type 4 mentioned above. The divisions among the first three types are open to criticism, on the ground that a good language of any one type can also be used to program operations of the other two types. To choose any one of these languages as a vehicle for discussion of that point, which is a perfectly correct point, may offend enthusiasts for one or more of the other languages. Hence I define:

(4) Languages designed to combine features of the first three types and to offer a little of everything (e. g. PL/I).

---

[1] Because of a combined economic and historical accident, I mean 'English' when I write this, although I realize the different rewards that may follow from the description of algorithms in other especially rewarding natural languages like Czech and Hebrew.

In Section 3, there are subsections for most of the languages which have been mentioned so far. The first language belonging to each type T will be taken as the language which "defines" T. Other languages in any T will be discussed less thoroughly, but the special or important features which distinguish them from the "defining" language will be emphasized. For each one of these subsidiary languages, I have tried to select the features which have given me the most pleasant surprises at the time that I have first studied it. Pleasant surprises seem to come from two categories:

(a) Features which correct or avoid basic faults in the design of already known languages (e. g. the requirement to declare arrays of fixed size in DIMENSION statements in FORTRAN, which is removed by the dynamic array-declarations in PL/I and similar languages or by the complete freedom from array declarations in FRED).

(b) Features which extend the user's imagination about what can be done with a language of a given type. (Examples: dynamic lists in POP-2, recovery from errors with the help of the function ERRORSET in LISP, the wide variety of modes of pattern-matching in CONVERT.) One's first reaction to a good example is probably, "Now why didn't I think of that idea before?". The next reaction, after a few seconds, tends to be, "Let's see if I can think of some new uses for this feature which are not described in the programming manual".

To sum up, the principal language of each type illustrates standard ways of attacking some characteristic programming problems. The other languages are discussed to show that there are interesting alternative ways of doing the same things.

It has been said that languages which are produced by committees are bland and homogenized. The implied corollary, that languages produced by individuals are individualistic and sometimes startling to the taste, has not been said so often, but it is a statement which is worth consideration. To attract the reader's attention to Section 3, I claim that I know four languages (at least one of which is mentioned in that Section) from whose style and structure it is possible to deduce not only some important personal characteristics of the authors, but also their original or present nationalities. Modest but appropriate prizes are offered to people who can guess any of these languages and who submit convincing short essays to explain their reasoning and back up their conclusions.

What is a programming language? The question arises by implication whenever writers of symbolic programs gather to discuss or advertise their products. The intending user sometimes hears of a "language" for algebraic manipulation, and sometimes of a "system". The situation is confused by the fact that systems sometimes evolve into languages. The classic example of this transformation is from A. C. Hearn's system REDUCE of 1966 (a set of programs written in LISP, initially for calculations of special interest to particle physicists and with a small subset of LISP functions for input/output conversion between LISP data and a more readable data-format-like FORTRAN notation) to the present REDUCE language (see Section 3. 2. 2) which has a complete syntax and which can be used for the construction of quite complicated new programs. Unfortunately, there is no clear-cut method of identifying a language, but one can apply a few tests which may be helpful under some circumstances, e. g.

(a) Is there a compiler which will translate the programmer's statements into machine code if asked to do so?

(b) Is it possible to use BNF notation to describe the syntax of whatever the programmer is allowed to write?

Answers of "yes" to both of these questions imply that one is looking at a language and not just a system or something less than that, but the questions do not settle the matter. However, one test which can be applied with a subjective probability of about 95% for being right is:

(c) Suppose that A and B are both candidates for the honour of being regarded as languages. If it is possible to write an interpreter for A in B and an interpreter for B in A, then A and B are both honourable languages. For example, the suggestion that LISP and FORTRAN are both languages is reasonable. It is possible to write a FORTRAN interpreter in LISP (indeed, the first version of FRED came out somewhat like this, except that it contained some improvements over FORTRAN design) and it is also possible to write a LISP interpreter in FORTRAN (in Uppsala University, M. Nordström, E. Sandewall, and D. Breslaw have done it and they have given the result the name LISP F1). It is not difficult to find similar illustrations for other pairs of languages.

After a fashion, the question "What is a programming language?" has been answered, but only at the price of having more questions to answer. What is BNF? What is an interpreter for a language L, and what kind of relation does it bear to a compiler for L? These are questions which can be avoided if one sticks to the familiar ground of FORTRAN, but whose answers may almost be regarded as necessary before one embarks on a comparative survey of programming languages of different types. They therefore lead smoothly into Section 2, where they are answered and where some other important aids to a description of different types of languages are introduced.

References on the subject as a whole are still few enough to be assembled after only a short search. All the entries in part B of the references are very useful. The items in part C are also likely to be of interest to physical scientists who wish to know about past applications of symbolic programming.

2. DISTINCTIONS AMONG TYPES OF LANGUAGES

In Section 1, the fact that a DIMENSION statement can only declare arrays of fixed sizes was quoted as a defect of FORTRAN. While it is true that there are some purely numerical computations which demand dynamic declarations if they are to work efficiently, these are greatly outnumbered by the numerical computations which are happy to abide by the restriction of fixed array sizes. Data structures of sizes which are fixed and known before the computation begins are the rule rather than the exception in numerical processing, but the situation is quite the opposite for most symbolic programming examples. In other words, structures in non-numerical work may grow at unpredictable rates and make unpredictable demands on storage. The FORTRAN array must therefore be replaced by an ordered structure which can be expanded in size at any time during a

computation. A structure which is easy to expand at a moment's notice is the diagram of Fig. 1. All the information in Fig. 1, together with its ordering, is represented if we abbreviate it by the list (AB, CD, EF) or (AB CD EF). To add an element GH at the right-hand end of the diagram is a simple step. First we find a new box like one of the three boxes in Fig. 1 and put GH into its left-hand half and an "end" marker like the diagonal bar at the right-hand extremity of Fig. 1 into its right-hand half. Then we rub out the diagonal bar to the right of EF in Fig. 1 and replace it by an arrow which leads to the new box. If only we can find a means of accommodating a structure like Fig. 1 inside the computer, we have solved the problem of how to represent data which grow in size at unpredictable rates during computation. But this is easy. Let each box be a single word inside the computer's storage. The words do not need to be in any special positions with respect to each other, as long as the arrows in Fig. 1 point out words unambiguously. A word in fast storage can usually be distinguished unambiguously by its address. Therefore, the arrow from any one place P to the next box N in the structure need only be represented inside the computer by the address of N stored in P. Technically these arrows or addresses are called pointers, and entire structures such as those in Fig. 1 are called lists. List processing of some kind is basic to most languages not built purely for numerical analysis.

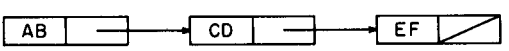


FIG. 1. An expandable structure (list structure).

Even at this early stage of discussion, it is possible to make some amusing technical observations about the consequences of choosing list-processing languages for one's work. The partitioning of boxes or words in Fig. 1 is fairly obviously (two partitions of equal length) the simplest assumption about the form of elements in lists. This may be one of the reasons for the comparative popularity of the language LISP (Section 3.2) or "List Processing Language", which uses it, but one can also build arbitrarily complicated structures with arbitrary partitionings (see Chapter 2 of Ref. [B4]). However, let us begin by considering the deceptively simple question of maximal use of fast storage for a LISP-like language. If, as is usual, the store consists of a number of words equal to a power 2^n, then it takes n bits of storage to represent each address of a word in the store uniquely. Thus, from Fig. 1, a pointer requires n bits, so that the total length of a box or word as shown is 2n bits. Hence, if the manufacturer of a computer presents you with a store of 2^n words, it is impossible to use all of the store properly for a LISP-like language unless the word length is at least 2n bits! For FORTRAN-like languages, which are the only languages that a majority of the world uses, there is no restricting connection between store size and word size, which is why manufacturers can sell computers with 16-bit or 24-bit words and get away with the claim that they are providing "general-purpose" scientific machines. The scorn implied in my statement is based on the fairly reliable folklore that no interesting scientific calculation in a list-processing system is happy

inside a store of less than 32K words (1K = 1024), which requires the word-length to be at least 30 bits for unique addressing.

A second simple question: assuming that one begins the addition of a new element GH to the list (AB CD EF), as in the discussion of Fig. 1, by putting GH into a new box or word, where does this word come from? How do we know that the word which we choose out of storage as a home for GH is not being used already to hold some needed item of data? The answer is that all words which are initially free are strung together on a single list, the "free-storage list". When a new word is required, it is merely picked up from the head (left-hand end) of the free-storage list.

Yet another question: what happens when the free-storage list is reduced to nothing? If this occurs, the storage is full – not a likely happening in FORTRAN, but list-processing computations are less well-disciplined. The only hope for the computation to go on is that some of the words in storage are occupied by material which is no longer needed. The process of identifying these words and putting them together to make a new free-storage list, which is called garbage collection, is discussed in Section 5.6. Garbage collection has no analogue in FORTRAN. At this point it is only appropriate to note that one or two bits in any word in free-storage space must be reserved, over and above the bits used to represent pointers or data, to help with the book-keeping involved in garbage collection. Thus a 2^n-word store used for list processing must actually have a word-length strictly greater than 2n bits.

An alternative and slightly longer introduction to most of these points, in connection with an introduction to LISP, is available in Ref. [C4].

The discussion of lists serves to distinguish numerical languages (type 1) from languages of types 2 and 3. How about a distinction between languages for symbolic processing (type 2) and string-processing languages (type 3)? With the help of Fig. 1, this is not difficult to provide. In Fig. 1, and in its condensed form (AB CD EF), the basic and interesting items of data are obviously AB, CD, and EF, which either stand for themselves or are maybe names of variables (as they would be if we met them in a FORTRAN statement), according to the context. Nowhere are we particularly interested in breaking AB into its component characters A and B. AB is an irreducible item of data, an 'atom' in LISP. (In fact, the left-hand half of the first word in Fig. 1 in LISP does not even contain a display-code representation of AB – consider the difficulty of putting such a thing into half a word of any reasonable computer in the case of a 24-character atom such as the name of a Welsh railway station, which is a legal atom in LISP – but another kind of pointer, to a completely different region of storage where the unique representation of the atom AB and its properties begins.) Moreover, for (AB CD EF) as data for a language like LISP, we do not care whether AB and CD are separated by one blank space or ten; the translation of each alternative into Fig. 1 is the same. On the

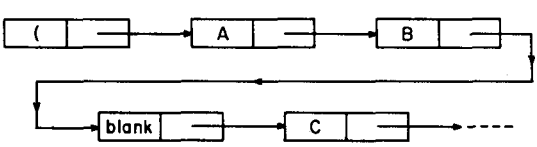


FIG. 2. The beginning of one possible representation of (AB CD EF) for a string-processing language.

other hand, there are certain situations (e. g. text-editing) where every blank is significant and where all characters (including blanks and brackets) should be treated on the same footing. Nevertheless, the data as a whole may run to unpredictable sizes, so that we still need a list to store our information in the computer. In this case, Fig. 1 is replaced by a list which may possibly begin like the list in Fig. 2. For such applications, a string-processing language like SNOBOL is the best choice. Operations such as the editing of texts or programs, or translation between one data-format (or language) and another, are ideally suited to string-processing. At first sight, this application may seem to reward the physicist less than symbol-processing (e. g. in LISP or REDUCE), but there are occasions when one has second thoughts about the appearance of the kilograms of output which sometimes come out of symbol-processing programs, and one wants to transform it into a more readable style by means of an analysis which in effect must go through the output character by character. Reference [C6] gives an example where exactly this type of transformation could have been helpful.

We can now return to one of the questions raised in Section 1, the question of BNF. According to whose books you read, this is an abbreviation for either Backus Naur Form or Backus Normal Form. The inventor of both traditions is John Backus, who was also strongly involved in the original design of FORTRAN. The second name belongs to the same P. Naur whose name appears on the early ALGOL report in Ref. [A3]. BNF is a means of defining the basic items of a programming language and building up a complete description of the syntax of a language in terms of those items. As an example of its use, I shall attempt a description of LISP syntax here (assuming for simplicity that this LISP does not handle octal numbers). Parts of the syntax are named between angular brackets $\rangle$ and $\langle$. A concept is being defined if it occurs to the left of the relational operator ::=. The vertical bar | denotes "or". All other quantities stand for themselves. To save horizontal space, let us assume that we begun by defining $\langle$alf$\rangle$ as any letter of the alphabet, A or B .... or Z. We next go on to the ordinary numbers, numerical atoms in LISP, and follow them with the definition of non-numerical atoms.

$$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

$$\langle \text{digits} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{digits} \rangle$$

$$\langle \text{sign} \rangle ::= + \mid -$$

$$\langle \text{float} \rangle ::= \langle \text{digits} \rangle . \langle \text{digits} \rangle$$

$$\langle \text{exponent} \rangle ::= E \langle \text{sign} \rangle \langle \text{digits} \rangle \mid E \langle \text{digits} \rangle$$

$$\begin{aligned} \langle \text{numerical atom} \rangle ::= {} & \langle \text{digits} \rangle \mid \langle \text{sign} \rangle \langle \text{digits} \rangle \mid \langle \text{float} \rangle \mid \\ & \langle \text{sign} \rangle \langle \text{float} \rangle \mid \langle \text{digits} \rangle \langle \text{exponent} \rangle \mid \\ & \langle \text{sign} \rangle \langle \text{digits} \rangle \langle \text{exponent} \rangle \mid \langle \text{float} \rangle \langle \text{exponent} \rangle \mid \\ & \langle \text{sign} \rangle \langle \text{float} \rangle \langle \text{exponent} \rangle \end{aligned}$$

$$\langle \text{character} \rangle ::= \langle \text{alf} \rangle \mid \langle \text{sign} \rangle \mid * \mid / \mid \langle \text{digit} \rangle$$

$$\langle \text{tail} \rangle ::= \langle \text{character} \rangle \mid \langle \text{character} \rangle \langle \text{tail} \rangle$$

$$\langle \text{LISP atom} \rangle ::= \langle \text{numerical atom} \rangle \mid \langle \text{alf} \rangle \mid \langle \text{alf} \rangle \langle \text{tail} \rangle \qquad (2.1)$$

This is not the complete set of definitions, because nothing more complicated than an atom has been defined above. So far, most of the definitions represent acceptable FORTRAN, but there are two exceptions. Firstly, no numerical atom of a form -.5 or .5 or 5. is allowed here, because the period is a special LISP character with an interpretation which is confused by such forms (-0.5 and 0.5 and 5.0 are allowed, though). Secondly, non-numerical atoms can have characters +, -, * or / anywhere but in the position of the first character, because LISP (unlike FORTRAN) does not regard them as characters with a special meaning.

What we have seen in (2.1) is only a foundation for syntax. The interesting parts of BNF definitions are the parts which build on such foundations and which look radically different for radically different pairs of languages. Moreover, the length of the BNF superstructure can tell us something about the theoretical clarity behind the language. It is possible to assert that long superstructures and many different syntactic terms indicate that the theoretical justification for a language is murky, to say the least. LISP escapes such abuse from the sidelines. Here is the complete superstructure which concludes its definition in BNF:

$$\langle \text{S-expression} \rangle ::= \langle \text{LISP atom} \rangle \mid (\langle \text{S-expression} \rangle . \langle \text{S-expression} \rangle) \qquad (2.2)$$

Thus the only allowed item in LISP more complicated than an atom is the S-expression, defined by (2.2). (Hence it seems that the list (AB CD EF) is not acceptable to LISP – see Section 3.2.)

I shall go no further with BNF here, except to make three comments. First, among computer scientists there is a whole abstract way of life based on comparative studies of languages via BNF. To the physicist comparing languages from an earthy point of view, this news is of no comfort, but the information about the existence of an abstract treatment may be of some help to people who wish to scan through technical papers and textbooks in the future. Second, one practical use for BNF is in the description of a language which is intended to be transplanted from one type of computer to another. Often, the first task in implementing a language on a new computer is to write a compiler for it which is acceptable to that computer. This is a long exercise if it is done by hand, but there exist programs ("compiler compilers") which can produce compilers for a language L to run on a machine M if they are given details of the machine code for M, plus a BNF definition of L.

The third comment deals with an apparent weakness of BNF and eventually leads me into my next topic. BNF allows no simple way of saying, for example, "an atom consists of no more than n characters". If, in Section 1, I try to see if FORTRAN is a language, then apparently FORTRAN fails the test, because (say) there is no method of stating that the maximum length of a Hollerith (H format) string is 49 characters. Actually there is a crude way around the problem: I can define a hierarchy of types

$\langle$ H string of length 1 $\rangle$ ::= 1H $\langle$ character $\rangle$

$\langle$ H string of length 2 $\rangle$ ::= 2H $\langle$ character $\rangle\langle$ character $\rangle$, etc.,

but after 49 of them the elegant flavour of the BNF definition has disappeared!

BNF does not take to this type of iterative description very well, so it must offer some compensation in order to have survived. The compensation is visible in (2.1) and especially in (2.2): quantities given in terms of themselves can be defined in a very tight and compact form. The operation of writing quantities in terms of themselves is called recursion.

Recursive definitions of functions or subroutines are not encouraged in FORTRAN, because of a difficulty in evaluation. Consider the most compact definition of the factorial function for positive integers:

$$\text{factorial (n)} \leftarrow \text{if n = 1 then 1 else n times factorial (n - 1)} \qquad (2.3)$$

For the factorial of 4, the first run through the function produces the result of 4 times factorial of 3. Before the final answer is obtained, the factorial of 3 must be evaluated, but 4 must be stored while 3! is calculated. This is a simple example of the general idea that a recursive computation needs a special area of storage to set aside partial results of the computation which are to be recovered and used later. Further, the partial results are recovered and processed in the reverse order from the one in which they were generated; in 4!, the first actual multiplication to be carried out is 2×1, then $3 \times 2 = 6$, and finally 4×6. We have one new (non-FORTRAN) structure which is ideal to handle such a situation: the list. A list on which all deletions and insertions are made at the same end (conveniently at the head or left-hand end in a LISP list as in Fig. 1) is also suggestively called a first-in last-out stack or a push-down list. Any system which is designed to process recursive functions or subroutines maintains a push-down list automatically to store and retrieve partial results of recursion. If you wish to allow recursion in FORTRAN, you must explicitly create and maintain a push-down list in a special region of storage reserved for that purpose. The questions of push-down lists ("Has the implementation of the language got a push-down list or, if not, how easy is it to create one?") and the ability to handle recursive procedures are good talking-points in comparisons of languages. Recursive procedures, (2.3) for example, usually have terminating conditions unrelated to the lengths of their calculations (contrast this with the need to give an upper limit to an index in a classical DO loop in FORTRAN) so that they are very practical for non-numerical computation in which the lengths of their data are unpredictable.

The last unanswered question from Section 1 is the question of "compilers versus interpreters". For people who have been brought up on FORTRAN, compilation seems to be a compulsory process. Conservatives may insist that compilation means translation into machine code, while liberals may be prepared to concede that a compilation has occurred if there has been a translation into anything not equal to FORTRAN, but both probably agree on its compulsory nature. Both are wrong. There is the radical alternative that a FORTRAN (or other language) structure which its writer would call a "program" can be treated as "data" by some other program which is not necessarily in the same language. On the basis of what it interprets to be the meaning of the first program, this second program can be written to produce the same output as the first program would have produced by itself if it had been compiled and then run. Suppose that the computation "2+3" is specified in a FORTRAN program. On compilation, 2 and 3 are translated into a binary form and stored in certain locations, and the sign + is translated

into a piece of machine code which specifies the hardware operation of addition. Later, when this section of the compiled program is run, the result of 5 is calculated and stored. A suitable interpreter computes and stores 5 at the time that it reads as data the string of symbols "2 + 3", because it contains pre-programmed rules for what to do when it sees the symbol +. Briefly, that is the difference between compilation and interpretation.

What are the special advantages of an interpreter? One advantage is that any one statement of a program can be executed as soon as it is read, without the need to wait for the rest of the program. Thus an interpretative system is the best base for so-called "conversational" languages, in which the user sits at a teleprinter and expects to get the answer 5 as soon as he types 2 + 3 followed by a carriage return. FRED is a language of this type, which has progressed through interpreters in LISP, ALGOL, and FORTRAN. A second advantage of an interpreter is that it can detect forms of a statement which are wrong and send the user of a conversational language an immediate error message which hopefully tells him what to do in order to type a correct version. A third advantage is that, treating a program as data, an interpreter can change the structure of that program. There are problems in artificial intelligence (e. g. the missionaries-and-cannibals problem in Ref. [C7]) in which even the correct answer may be of less interest than the structure of a "learning" or self-optimizing program at the end of a computation, given that the initial program contains only some simple-minded algorithms.

Normally an interpreter is used when there is a need to preserve the original structure of a program, e. g. so that the program can be changed while it runs and still be understandable at the end of a run, or to assist debugging, but it is also used when there is no compiler at all. When both compilers and interpreters are available, the time to call upon a compiler is when a program not designed to make alterations to itself is free from obvious bugs and is ready for long or repetitive production runs. The compiler stands a fair chance of decreasing the space required to store a program (good LISP compilers produce compiled code which usually occupies 80 to 100% of the space taken by the corresponding interpreted program), but compiled programs can be expected to run between twice and ten times as fast as interpreted programs.

Languages which operate happily under the control of interpreters frequently differ from FORTRAN-like languages in some important and clearly-noticeable properties of notation. We have already seen an example, in 2 + 3, which helps to explain the reason for the difference. As long as an operator takes a fixed and known number of arguments and occurs a known number of times in a given context, it is reasonable to turn an interpreter loose on a structure written in the infix notation of 2 + 3, in which the operator is sandwiched between its arguments. However, for the expression 2 + 3 + 4 + 5 + 6 + 7, repeated scans for the context of + waste time and space. The processing is more efficient if repeated operators are replaced by a single occurrence of an operator in a convenient position, e. g. + 2 3 4 5 6 7. This alternative notation is known as prefix notation. Here, each operator stands to the left of all its arguments. The translation of the sum above into a list is (PLUS 2 3 4 5 6 7). In all LISP-like languages, prefix notation, which is the rule, makes the construction of an interpreter rather easy. In processing the arbitrarily complicated list structure in some function-definition, the interpreter can be assured that, whenever it

finds a substructure which is a list of atoms, the first atom is the name of a function and the remaining atoms stand for its arguments. Further, if it finds a list whose first member is an atom, there is a good bet that this member is the name of a function and the remaining members are substructures which, when analysed by recursive application of the same rule of thumb, can eventually be evaluated to generate arguments for the function. The expression (TIMES (PLUS (QUOTIENT 15 5) 8) (MINUS 6)) exemplifies the rule. Without knowing much about LISP, it is still possible to look at the "definition" of the LISP interpreter in Appendix B of Ref. [A11] and realize that it must be much shorter than a FORTRAN interpreter which can take account correctly of all the tricks of FORTRAN IV. The main reason for this shortness is the use of prefix notation as a convention of the LISP language.

The language PL/I is discussed in Section 3.4 as an example of a language which combines features of numerical processing, list processing, and string manipulation. In principle, therefore, anyone who has a problem requiring aspects of all three operations can write his programs in PL/I. However, problems of simulation of complex systems, which have uses for each of the three operations, have been sufficiently widespread to call for special languages of their own. SIMULA 67 is a language of this type. Languages for simulation do not differ in essentials from other languages in the appearance that they present to the user, but internally they must be capable of new tricks, e.g. the organization of each basic task in a complex simulation as a separate "program", the capability of imitating parallel processing of tasks which run in parallel in real life, and the detection, as errors (even at the time of compilation if possible), of situations where data is being used under the wrong assumptions about the previous behaviour of the simulation. If all these advanced capabilities are to be available automatically as a part of a language, it must already be able to call on all the internal aids to operation (e.g. push-down lists), which are characteristic of (say) FORTRAN plus LISP plus SNOBOL, before the user ever sees it. This requirement sets a very demanding standard which may not have been met fully yet in languages for simulations, but it is one reason why they are usually based on existing languages. As an example, the foundation for SIMULA 67 is ALGOL 60, and the ALGOL face is the face that SIMULA 67 presents to the external world.

The problems of management of large scientific data bases are not always well-defined. If I wish to store information on, for example, 2597 bright galaxies, with the idea of discovering all kinds of interesting relationships among members of the sample at some later date, I am somewhat handicapped in my choice of internal representation in storage by the fact that I do not always have a clear impression of the kinds of question which I shall want to ask about the sample tomorrow, or next week, or next year. The best that I can do is to identify all the attributes which are likely to be interesting and arrange lists of all the members with given attributes. The structure of storage then becomes very complicated very quickly, as Fig. 3 illustrates. There, I am assuming that members 1, 3, 4, 6, 8... possess attribute A, members 2, 3, 5, 7, 8... possess attribute B, and 1, 2, 6, 7,... possess attribute C. The result is a non-planar and multiply-linked list structure with each node or element containing pointers of several different types and therefore being several words long. Already we are some distance away from the simplicity of Fig. 1 and

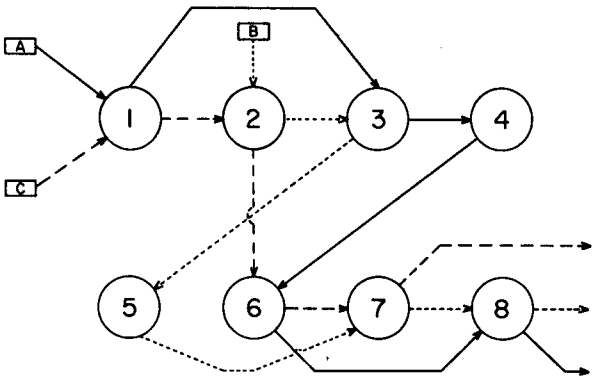


FIG. 3. Storage of part of a large scientific data-base by means of the Heath Robinson or Rube Goldberg method.

LISP. Chapter 2 of Ref. [B4] contains a discussion of multiply-linked structures and how to handle them directly, but fortunately the user of a language which interrogates such a data-base does not have to worry about internal implications. His only concern is with how to ask questions in the language. Individual questions are usually quite short and self-contained, so that there is not even likely to be a need for a "program" in the usual sense. The syntax is then as close as possible to natural language. Labels for concepts cannot always be descriptive, so the user has to learn them from the gentleman who has written the procedures for storing the data-base, but otherwise the conventions are easy to learn. For example, in the case of the question mentioned earlier, the number of spiral galaxies with $x < U - B < y$, one may type

PRINT COUNT C8 WHERE x < C35 AND C35 < y;

(knowing that C8 is a label referring to spiral galaxies and C35 to the index U - B). Thus, from the standpoint of the collector of languages, the user's view of the languages for management of data-bases is neither a complex nor an exciting one. The benefits of knowing about their existence lie entirely in the value of the answers to the questions which we may use them to pose. Therefore, they deserve an honourable mention, for completeness, in a course of lectures which surveys programming languages of interest to physicists.

3. DESCRIPTION OF SPECIFIC LANGUAGES

3.1. FORTRAN

A knowledge of FORTRAN IV is assumed. Therefore, no comment.

3.1.1. ALGOL

FORTRAN is a relatively unstructured language. This is a proposition which becomes clearer if we consider the tendency for FORTRAN answers to many complicated programming exercises to take the form of a giant main program plus a few short subroutines (or occasionally a microscopic main program plus a few giant subroutines). In a small number of highly complicated cases, it may be prohibitively difficult to split up the giant exfoliations because each ten lines or so of the program contain transfers to statement numbers in every other ten-line group in the program. (I choose ten only for the sake of an example.) However, in most cases we can draw horizontal lines across the programs in several places, such that there is either a minimal amount of communication across the lines by transfers or no communication of this type at all, i.e. one crosses a line only by sequential execution of the last statement before a line followed by the first statement below the line. Where we can make such divisions, there is a reason for having a language which encourages us to think of the divisions directly in terms of structured procedures which reflect the actual logical organization of the program in a clear way and to show exactly where each logical building-block begins and ends. ALGOL is such a language which pays close attention to this type of organization. (Some people say that its attention is so close as to be neurotic, but that is another story.*) (For this asterisk, see explanation later on.)

Apart from the consideration of being able to detect an overall logical structure in a program with the naked eye, there is the question of the extent to which the listing of a program should manage to describe what the program itself is intended to do. Binding statements like A = B in FORTRAN are perhaps clear enough and we can always "explain" what is happening in sections of FORTRAN programs by using a plentiful supply of comment cards, but otherwise individual statements do not look much like individual statements in anybody's natural language, so that their meanings are obscure at times. We can write a factorial function in FORTRAN which makes use of the word "if" in IF statements, but to somebody who knows no FORTRAN it is unlikely to be as clear as the definition (2.3). What is needed is a language in which the representation of the English (2.3) is something like

```
PROCEDURE FACTORIAL (N);
IF N = 1 THEN 1 ELSE N × FACTORIAL (N - 1);        (3.70)
```

This is ALGOL (more or less)* – a language in which programs can be self-documenting. ALGOL was designed partly for that purpose; it is now an accepted feature of computing journals that published algorithms are written in ALGOL for ease of understanding.

A third justification for one characteristic of ALGOL is efficiency of computation, although by no means all the characteristics of ALGOL promote efficiency. LISP is a language which accepts both numerical and symbolic data, but manuals warn users that the purely arithmetical operations in LISP are so slow that they should be avoided as far as possible. Numbers and symbols are different types of data, so they are stored differently, and the bias of uses of LISP is such that the symbolic type of data has a

simple representation. When a LISP program accesses a numerical atom for an arithmetical operation, it has to find the "number" attribute of this atom, convert it to a form suitable for the operation, perform the operation, make the inverse conversion on the result, and store the result in a rather non-simple representation. All this trouble comes about because there is no opportunity in the language to declare that some variables are of numerical TYPE and have only numerical values. When such declarations are possible in a program, the compiler (or even an interpreter) can ensure that values of differing types are stored in different regions of the computer, with efficient use of each region and efficient access free from the need for special tests or preliminary searches. FORTRAN II had one implicit declaration, in that names of variables beginning with I through N were assumed to stand for integers whose storage was reserved at the time of compilation on a different and more compact basis than for other variables. Classical ALGOL takes this tendency to the extreme, in the sense that it is regarded as good programming form to declare the types of all variables at the time that they are introduced. From the points of view of documentation and of compiling efficiency, this characteristic of ALGOL is a Good Thing, although people who are responsible for comments like the ones labelled by an asterisk (*) earlier in this section have been heard to complain that one spends so much time in declaring things in ALGOL that one never gets around to any real programming. (This is not necessarily the view of the author.)

What is the standard version of ALGOL? Aside from the issue of local dialects peculiar to individual computing centres, the published references to ALGOL sometimes quote ALGOL 60 and sometimes ALGOL 68. A quick answer to the question is that, at the time of writing, I know of no implementation which has either the full ALGOL 68 or an exact subset of it. Several years of experience with ALGOL 60, which was the original standard, led to the establishment of many new needs that this standard did not meet. ALGOL 68, which is the second try at a fairly universal algorithmic language, has features for almost any algorithmic requirement that a programmer can think of, as a fast scan of Ref. [A4] will demonstrate, but its description is now so long that a complete implementation will require truly Herculean labours. At present, the most hopeful sign is the existence of a system ALGOL 68 R, containing a restricted but useful subset of ALGOL 68 plus some deviations, at the Telecommunications Research Establishment in Malvern, United Kingdom. If at some time in the future you see trailers for Mr. J.-L. Godard's film "Alphaville, Mark 2", in which the villainous central computer is called α 68 instead of α 60, and speaks French with either a Worcestershire or a Dutch accent, you can assume that ALGOL 68 is working somewhere. Until that time, I refer to ALGOL 60.

Here are some properties which contrast ALGOL with FORTRAN.

3.1.1.1. Treatment of symbols

A language for algorithmic description which limits itself to upper-case alphabetical symbols, as on the IBM 026 card punch, is likely to be less expensive than one which allows all alphabetical symbols. The ALGOL convention is to allow all alphabetical symbols, including bold-face letters,

normally indicated by underlining in manuscripts (or on Flexowriters connected to ancient computers). Bold-face type is used only to compose special words in ALGOL. The most important of these are:

go to comment array (3.71)

if then else (3.72)

do for step until while (3.73)

begin end procedure (3.74)

real integer Boolean switch label value (3.75)

The mixture of type faces makes algorithmic descriptions in ALGOL easy to read, but the language still has to be accommodated on computers which can only digest the symbols on an 026 card punch. In these cases, the concession is made that lower-case letters translate into upper-case letters, and the quantities in (3.71) through (3.75) are surrounded by quotation marks, usually either the escape character ' or the character given by ⟨ shift 2 ⟩ on a teleprinter keyboard.

The uses of the special names in (3.71) can be deduced from FORTRAN experience; array is the analogue (though not an exact analogue) of DIMENSION in FORTRAN.

3.1.1.2. Conditional statements

Unlike the variety of IF statements in FORTRAN, ALGOL has only one general form. Its two simplest manifestations are

$$\underline{\text{if}}\ p_1$$

$$\underline{\text{if}}\ p_1\ \underline{\text{then}}\ v_1$$

but in more complicated examples we find the structure

$$\underline{\text{if}}\ p_1\ \underline{\text{then}}\ v_1\ \underline{\text{else if}}\ p_2\ \underline{\text{then}}\ v_2 \ldots .$$

$$\ldots\ \underline{\text{else if}}\ p_{n-1}\ \underline{\text{then}}\ v_{n-1}\ \underline{\text{else}}\ v_n \qquad (3.76)$$

for any set of quantities p (where the type of the value of p is Boolean) and v which are legal ALGOL expressions (including other expressions of the form (3.76)). Binding operations like A = B from FORTRAN can be combined with conditional expressions if they are written in place of a v_i, e.g. if a = 2 then a := b. Notice that ALGOL distinguishes between binding operations and the use of = to test for equality, but FORTRAN does not. There is little chance of ambiguity in the use of = in FORTRAN because the meaning of the operator is usually determined by its context, but ALGOL leaves nothing to chance in this case.

The advantage of the form (3.76), particularly in documentation, is that it is easy to read and understand. Most languages copy roughly the structure of (3.76) rather than the structure of IF statements in FORTRAN.

3.1.1.3. Compact descriptions of iteration

The line of examples of bold-face type, (3.76), contains words which have special uses in ALGOL to regulate the local and repeated computations which FORTRAN handles with DO loops. Many DO loops, particularly those for the assignment of initial values in arrays, surround a single binding statement which is to be executed iteratively. ALGOL allows all the necessary information to be expressed in one statement, in a readable and natural form. For example, the FORTRAN code

```
      DO 25 J = 1, 86, 5
25    N(J) = J**3
```

translates into

```
for j := 1 step 5 until 86 do n[j] := j ↑ 3;        (3.77)
```

in ALGOL.

The last word in (3.73), while, also acts as a modifier for the conditions under which an iteration is performed and precedes do. An illustration of its use is that an ALGOL equivalent of

```
14 Y = X
15 IF(A.LE.B) GO TO 16
   A = A - Y
   GO TO 15
16 ---
```

is

```
n14: for y := x while b > a do a := a - y;
```

If the iterative loop covers more than one statement, e.g. S_1, S_2, S_3, then these statements can still follow do in sequence, provided that they are surrounded by the markers begin and end, as in

```
do begin S1; S2; S3 end;        (3.78)
```

The general use of the markers is described below.

3.1.1.4. Block structure and its delimiters

All ALGOL programs are organized into well-defined blocks. Each block of statements and declarations is marked at the beginning by begin and at the end by end. This is true for isolated instances like (3.78) as well as for full-fledged instances of blocks which would be subroutines in FORTRAN.

Blocks can be nested like DO loops in FORTRAN, and transfers from statements in an inner block to statements in blocks which surround it are

legal. Transfers in the opposite direction are strictly forbidden, though, because it is compulsory to enter an ALGOL block at its beginning in case any declarations local to the block have to be made there. The contrast between FORTRAN and ALGOL in this respect is somewhat like the contrast between a traveller's experience of customs formalities in Belgium and the United States.

For every begin there must be a single matching end later in the program.

A block which is large enough to be dignified by its own name corresponds to the definition of a function or subroutine in FORTRAN. In ALGOL, one defines such a block, together with its name, by using the word procedure in the body of a program. A square-root function which in FORTRAN would look like

```
FUNCTION SQRT(N)
----
RETURN
END
```

might have the form

```
real procedure sqrt(n);
value n; real n; begin
        -----
end
```

in ALGOL.

Uses of real, value, and the other words in (3.75) are outlined below.

3.1.1.5. Declarations of various kinds

Declarations of types of variables have been discussed in Section 3.1.1. Most ALGOL 60 systems are permissive; they do not require that the types of all variables be declared, but they assume that the default type of undeclared variables is real. This is a reasonable assumption (e.g. most FORTRAN variables have floating-point values). The equivalent of FORTRAN fixed-point is the type integer. A third regular type, Boolean, refers to variables which can have either of the two values true or false. If we see x := a > b, it is obvious that x is of Boolean type. In a sense, label is a fourth type if we declare various quantities to be statement-labels, but that exhausts the table of commonly-used types. All interesting extensions to ALGOL 60, official or unofficial, seem to contain proposals for new types to add to the flexibility of the language, e.g. alpha for strings in Burroughs' extended (unofficial) ALGOL, scalar for variables with values equal to LISP S-expressions (in REDUCE), and class for procedure-structure skeletons of particular types in SIMULA 67 (which contains ALGOL as a subset). For this discussion, the advantage of ALGOL is that it raises the issue of variable types and what should be declared about them.

Quantities declared by switch are special arrays: arrays of statement-labels of type label. These arrays allow neat transfers similar to the computed GO TO in FORTRAN. The set of declarations

```
label f, g, h;
switch trans := f, g, h;
```

makes it possible for

```
go to trans[if x < 3 then x else entier (1 + 7/x)]
```

to transfer control to h if x = 3, g if x = 2 or 4 through 7, or f otherwise (as long as x is positive).

3.1.1.6. Call by name and call by value

Suppose that we are in a part of a FORTRAN program and find the statement N = 3. After that statement has been executed, where in the program is the information available that N now stands for 3, and where is this news unknown? If we are in a function or subroutine from which N is not to be returned, and if N is not a COMMON variable, the information is local to that subprogram. If we are in the main program, the information is inaccessible to any functions or subroutines (except those that take this N as an argument), unless N is a COMMON variable. If N is common, the news is available in every region which is headed by a COMMON declaration. We tend to take this state of affairs for granted if we have been brought up on FORTRAN. ALGOL treats the question in a slightly different way, so that we have to think carefully about it.

If we want a certain variable to be purely local to a block, so that bindings made to it do not affect the state of quantities with the same name anywhere outside the block, we must declare it by using value at the head of the block. A declaration

```
value a, b, c;
```

indicates that bindings made to a, b, and c inside the block are to be forgotten when one gets past the block's end marker and that these bindings have no effect on uses of variables named a, b, and c outside that block. Any quantities not so declared have their changes (if any) inside the block accessible in any other block where they are not protected by being named in a value declaration. A quantity which is called in a block where it has occurred as an argument of value is said to be "called by value". Otherwise it is "called by name". This distinction is, under a different label, exactly the same as the distinction between "local" and "global" quantities which comes up again in Section 4.3.

If we attach great value to the idea that one should be protected against situations where casual bindings of variables in some part of a program produce surprising side-effects elsewhere in the program where the same names are used, then FORTRAN is a fail-safe system. Automatic transmission of information does not happen unless we ask for it in a COMMON declaration. Judged by the same standard, ALGOL is unsafe. However,

if this leads to more thought on the part of the programmer, the lack of safety may not be a bad thing.

3.1.2. JOSS or CAL or FRED

All three of these languages share the same essentials. The main reason for the differences in names can probably be traced to local pride. I apologize for my own involvement in what is, after all, one of the seven deadly sins, but I shall nevertheless call all of the languages FRED in these notes. Our choice of the name FRED is in honour of J. Fred Nurke.

While designed originally as a language for numerical processing, FRED differs from FORTRAN and ALGOL in being intended for conversational (rather than batch-processing) use. This intention dictates some special properties of FRED. One obvious property of this kind is that every statement has an identifying number. The number is available for transfers of control, as in FORTRAN, but it is equally important to label each statement uniquely to allow the user to add, modify or delete lines of his program by referring to the labels. These labels are floating-point numbers. The programmer can break his program into logical parts (see Section 3.1.2.2) by giving the same integer number to each statement in a part. Thus statements 2.2, 2.23 and 2.231 are all members of part 2, and their order in the program is fixed by the order of their decimal fractions. The idea behind the fractional notation is that if the user decides that he wants to insert a new statement between 2.23 and 2.231, he can choose the number 2.2305... and so on. If one merely types a statement preceded by a number N, it is automatically inserted into the program in the correct place suggested by N.

3.1.2.1. Desk calculator

The most simple conversational system that one can wish for is a system which will evaluate expressions which are typed in and will print out the results. Examples of expressions may be 2 + 3 or

```
ABS(ENTIER(SIN(LOG(COS(EXP(SQRT(RANDOM(0))))))))
```
(3.79)

Lists of expressions may also be used to save time and space, e.g. 2 + 3, 3 + 4, 4 + 5 on entry calls for the printing of 5 7 9. To imitate a program while still doing something only worthy of a desk calculator, FRED can process

```
1.0 DEMAND X
1.1 Z = SQRT(X)
1.2 "SQUARE ROOT OF", X, "IS", Z
RUN
```

and produce the expected result. DEMAND is the input operation: execution of statement 1.0 causes the output of

X =

on the teleprinter. The program then waits for the user to enter a number (free format) before going on.

All of the conventional infix operators and functions (e. g. (3. 79)) are available in the desk-calculator mode.

3.1.2.2. Compact descriptions of iteration

FRED's answers to the problems of iteration resemble those of ALGOL, but there are some alternatives which are made possible by the structure of the language. The equivalent of

if a = b then for j := 1 step 5 until 86 do n[j] := j ↑ 3;

in FRED is (with the statement number omitted)

N[J] = J ↑ 3 FOR J = 1 BY 5 TO 86 IF A = B

In other words, the order of the operation and the various modifiers in FRED is reversed over that of ALGOL. Such inessential differences are amusing because, while they are no obstacle to understanding, they show how much variety can be found in simple processes of thought.

In FRED, the words BY and TO replace step and until from ALGOL. WHILE and while have the same meaning. However, FRED adds the alternative modifiers UNTIL (not now to be confused with until) and UNLESS to the repertoire, for use in the same context as WHILE. In all cases, we succeed in our intention that iterative FRED statements with modifiers should read like English.

There is no provision for more than one operation per statement in FRED, so that the "do begin S_1; S_2; S_3 end" structure of ALGOL must be handled in a different way. The statements S_i form a logical part of any program, which means that they can be given separate statement numbers in FRED with the same integer value. An equivalent of

if a = b then do begin S_1; S_2; S_3 end;
c := 3;

in FRED is therefore

2.6 DO PART 3 IF A = B
2.7 TO STEP 4.0
3.0 S_1
3.1 S_2
3.2 S_3
4.0 C = 3 (3.80)

Beware the temptation to leave statement 2.7 out of (3.80)! This causes part 3 to be executed twice. For some no doubt psychological reason, such errors — (3.81) — seem to be of the most common type in FRED.

Example (3.80) also demonstrates the distinction between a STEP and a PART and shows that TO STEP n is the analogue of GO TO n in FORTRAN. Statements beginning with TO PART or with DO STEP (also a common source of the (3.81) error) are allowed and have the obvious meanings.

3.1.2.3. Arrays

Section 6.3 contains an account of how arrays are maintained in FRED. Out of that account, I shall mention here only that the FRED interpreter uses a list structure for storage of elements of arrays. If we are dealing with arrays of fixed dimensionality in which all elements from index 1 to a known index n are present, then any use of pointers is unnecessary and wastes space, but the idea behind the list structure is that we do not always want to be limited to FORTRAN-like arrays.

The first benefit of the hidden list structure is that indices can run from any integer (including negative numbers) to any other integer, without having to take steps of +1. If we only want to define A[-5] and A[2] at some stage, there is no need to reserve space for the undefined intervening elements of A, because there is a pointer from the place where A[-5] is stored which points directly to A[2]. To put in a new element is as simple as the alteration of pointers. Therefore it is unnecessary to use DIMENSION statements to declare arrays in FRED. If a quantity is followed by subscripts inside square brackets, that is a sufficient identification for an array element.

A second novelty which becomes possible because of the list structure (see Section 6.3) is that the dimensionality of an array need not be fixed: the quantities A (an ordinary variable), A[1], A[2,-7] and A[0,2,4,6] can all coexist in storage without complications. This is a further stretching of the range of possibilities for what can be done with arrays.

3.1.2.4. Uses for special characters

After setting up the most basic operations in FRED and adopting the conventional uses for some characters (e.g. * for multiplication and ↑ for exponentiation) on the teleprinter keyboard, we noticed that there were a few characters left unused. Although it is no way to design a language (except if the language is APL), we rose to the challenge of finding meaningful uses for them. Here are two results.

Firstly, the pictorial appearance of the character ↓ ⟨shift 1⟩ suggested a form of reduction, at the time that an early FRED program was required to evaluate many expressions of the form "x modulo y". The coincidence was too pleasant to be ignored. In FRED, "x modulo y" is now represented by x ↓ y.

Secondly, the character → (TAB or tabulate) was still unused when there were one or two requests from users for a crude curve-plotting facility. In the FORTRAN-interpreted version of FRED, where we have explicit control over the output buffers, we chose →n to mean "tabulate to column

n from the left margin of the page". As repetitions of the character I in the same column imitate a line or axis and * is a good way of marking points of a curve, we can, for example, produce a good plot of the sine function, with values of the independent variable at the left of the page, if we execute the single FRED statement.

X, → 30, "I", → 15*SIN(X) + 30.5, "*" FOR X = 0.2 BY 0.2 to 7.0

Although the use of the character → was thought up rather casually, it has since proved to be a valuable feature of FRED if one can judge by the reactions of users.

This story has no moral. It merely shows a small piece of history and suggests that certain admired features of other languages may once have been invented in the same unsystematic way.

3.1.2.5. Facilities for tracing

If a program runs without error conditions but produces results which are obviously wrong, one method of collecting information on the fault is to run the program again, causing the value of each binding operation and the destination of each transfer of control to be printed out. In FRED, the user may issue the command TRACE to achieve this effect. Then, every binding operation like A = 3 in the program is printed out as it is made, and any transfer to statement n causes the output of the message BRANCH TO STEP n. The command UNTRACE turns off all of the tracing messages.

As well as being available as commands external to a program, TRACE and UNTRACE may be inserted as actual statements in the program to provide tracing of only a limited part of its operation. The usefulness of the insertions is increased because TRACE and UNTRACE, like any other statements, may be restricted further by modifiers, e.g.

5.92 TRACE IF A = B UNLESS N > 15

Although these facilities may often be of great value to the user, they are not yet generally available in the better-known numerical languages.

3.1.2.6. Errors

A first principle of any good conversational language is that no error should be allowed to end a whole job. The reason, of course, is that the user at his teleprinter or terminal keyboard is in a position to correct the error immediately and try his calculation again, provided that he receives an intelligible message about the nature of the error. For example, the typing of the statement

3.4 Z = (X - Y * (X + Y)

should produce the message

MISSING)

and a chance to put statement 3.4 in again. Similarly, if the command DO PART 16 FOR N = 5 BY -4 TO -7 controls

16.3 A = SQRT (N),

then during the run of the program it is reasonable to expect information about the place as well as the type of the error, e.g.

ERROR IN STEP 16.3:

ARGUMENT < 0 for SQRT.

Both of the examples above work as shown in FRED.

FRED has about 35 error messages to cover most of these eventualities. No system is perfect, however. Only about 34 of them can be understood directly. The last error message, which appears if none of the first 34 may be made to fit the crime, is:

WHAT WAS THAT?

3.1.3. APL

The syntax of FORTRAN is adequate for most numerical programs, and there is a fair range of special characters (e.g. *, +, -, |) which are reserved to denote commonly-used operations. In ALGOL and FRED we have seen suggestions for extensions in syntax, but the possibility of extensions to the set of characters has been put to one side. APL looks rather bare when syntax is considered, but it has a wide range of special characters with mathematical significance, thanks to the requirement that it should see the outside world through an IBM 2741 terminal with an APL keyboard. APL is a conversational language which is processed by an interpreter that must expand the user's compact expressions. Thus APL does not make an easier internal job of processing instructions to do a particular mathematical task than (say) FRED, but it may turn out that the user has less work in typing the necessary instructions in APL because some function of two arguments x and y, which is not simple to express in natural language, happens to be available in APL as x ? y, where ? stands for one of the special characters. This compactness is only part of the story; suitable combinations of the existing operators can express very complicated but useful mathematical manipulations in short forms which are truly remarkable (e.g. (3.82)). The extra importance of such forms is that they give us a completely new picture of what can be done mathematically with a programming language. The point of the present discussion of APL is to concentrate on these educational novelties.

3.1.3.1. Dual uses of operators

To extend the range of distinct operations by roughly a factor of 2, the principal symbolic operators have both unary (?x, where ? is the operator) and binary (x ? y) meanings. For example, if we assume that

A is the argument of unary forms of operators and A and B are the arguments in the binary case, we find:

OPERATION	UNARY	BINARY
x	sign of A	A times B
÷	1/A	A/B
*	e^A	A^B
⍟	$\log_e A$	$\log_A B$
!	A!	binomial coefficient $\binom{B}{A}$
⌈	A if A is an integer, otherwise 1 + entier (A)	maximum of A and B
⌊	entier (A)	maximum of A and B
0	πA	TRIG (B)

where entier (x) = x if x is a positive integer, or the largest integer smaller than x if x is positive and has a decimal-fraction part, and entier (x) = -entier (-x) if $x < 0$. The function TRIG is the sine if A = 1, arcsin if A = -1, cos if A = 2, arccos if A = -2.....

3.1.3.2. Precedence

Given an expression like A + B/C * D ** N in FORTRAN, it is possible to see that it is processed as (A) + (B/C * (D ** N)) because of simple rules of precedence which put ** ahead of * and /, which are in turn ahead of +. To discover which of * and / (which are on the same level of precedence) comes first, we have to apply an altogether different rule which says that the scope of a binary operator does not extend past its right-hand argument and that the scan for operators goes strictly from left to right. Hence B/C * P means (B/C) * P and not B/(C*P). If that explanation does not seem to be clear, then nobody is happy — except the supporter of APL. In APL, there is one simple rule for precedence which replaces the relatively complicated conventions in other languages: the right-hand argument of any operator is everything between that operator and either the end of the line or the first unpaired right-hand bracket, whichever comes first. One surprising (unless you have thought of it before) consequence of this simple rule is that the expression A - B - C - D - E does not mean A - (B+C+D+E), as in FORTRAN, but A - (B - (C - (D - E))). In other words, A - B - C - D - E in APL means the same thing as A - B + C - D + E in FORTRAN! This example generates all kinds of passions for and against APL and is therefore a suitable topic to bring up at parties where many computer scientists are present.

3.1.3.3. Compact operations

For examples of the kind where relatively complicated operations can be compressed into a small number of symbols, APL relies on the rewards bestowed by the rule of precedence mentioned in Section 3.1.3.2 and on the variety of operator interpretations suggested in Section 3.1.3.1. Two examples of operations on scalars quoted in Ref. [B1] are:

(a) Efficient factored form of $a + bz + cz^2 + dz^3$.. without brackets

$$A + Z \times B + Z \times C + Z \times D \ldots$$

(b) Continued fractions, e.g.

$$a + \cfrac{1}{b + \cfrac{1}{c + \cfrac{1}{d}}}$$

which in APL becomes $A + \div B + \div C + \div D$ (3.82)

3.1.3.4. True and false

In ALGOL, variables of Boolean type can take on the values true or false. These values are completely distinct from any other value quantities in the system, in particular the integer values 1 and 0. There is a similar distinction in LISP. On the other hand, there is nothing to stop the construction of a program in any language in which the Boolean truth-value is denoted by the integer 1, and falsity by 0. APL uses this convention automatically. Thus the value of the logical test A > B is the integer 1 if A is greater than B, or 0 otherwise. The value can be bound (binding operator is ← in APL) to some variable without further ado, e.g. C ← A > B. Normally, one may believe that there is no point in doing purely arithmetic computations on numbers which have logical origins, but one practical consequence of relaxing the rule is that conditional transfers (→ is the analogue in APL for "GO TO") can be written in a compact form. For example, the FORTRAN transfer

IF (A - B) 15, 25, 35

may be rendered as

$$\rightarrow 15 + (10 \times A = B) + 20 \times A > B$$

Other applications for numerical operations on the APL logical "true" and "false" values are not hard to find.

3.1.3.5. Arrays and operations on arrays

For the same reasons as in FRED, arrays in APL can be created at any time by the naming of elements, without the need for anything like a

DIMENSION statement. There are two characteristics of FRED arrays which could have been reproduced in APL, but which were not. The first is the freedom to leave gaps in arrays by defining disjoint elements like A[-5] and A[2] while saying nothing about the others (and the freedom to have indices less than 1). The second is the legality of having different numbers of indices for A in storage at the same time. The reason for requiring APL arrays to have conventional behaviour with respect to indices is that there are special array-operations which make no sense or little sense unless the FRED freedoms are abridged. For example, there is the "dimension" operator ρ whose value in the context ρA, where A is an array with n indices running from 1 to u_1, 1 to u_2,... 1 to u_n, is the one-dimensional array with the n components u_1, u_2,... u_n. The quantity $\rho\rho A$ is then also defined quite logically as a one-dimensional array with one element whose value is n. Since arrays whose elements are also arrays are allowed in APL, it is important for the programmer to be able to make these repeated applications of the "dimension" operator to find out what is going on in his book-keeping.

Another justification for the conventional treatment of array indices is in the existence of global operations. In FRED, if the scalar A and elements of the array A exist simultaneously, A + 3 means "add 3 to the scalar A". In APL, where A cannot be both things at once, the value of A + 3, if A is an array (required to be one-dimensional in this case), is an array of the same size, constructed by adding 3 to each element of A. Thus the effects of operators (like +) differ according to the types of their arguments.

An array with given values can be set up in one APL instruction, e.g.

```
V ← 2 3 5 7
```

causes V to be a one-dimensional array with V[1] = 1, ..., V[4] = 7. To stretch this array to include V[5] = 11, we have only to write

```
V ← V, 11                                                    (3.83)
```

and the extra element is concatenated by the comma operation. If both arguments of the comma are named arrays, we produce entire new arrays by concatenation.

The binary use of the operator ρ is for initializing or rebuilding arrays; to make A equal to an array of ten zeroes, we write

```
A ← 10 ρ0
```

At this stage it is easy to see how arrays of arrays arise. We achieve a 10×10 array of zeroes with

```
A ← 10  10 ρ0
```

but

```
A ← 10 ρ0  0
```

is a one-dimensional array with ten elements, each being an array with two elements of zero. (Warning. Apparently this is a minority convention

for home-built APL systems, and it is not true in the canonical IBM system. There, the meaning of $x\rho y$ is that the array of dimensionality x is to be created out of elements of y, repeated from the beginning of y if y is too short to fill up the entire new array.)

Certain global operations which are well-defined "over" a one-dimensional array are signified by the symbol / which stands for the quotient operation in other numerical languages. For example, the sum over the last array V, which is 2 + 3 + 5 + 7 + 11 = 28, is denoted by +/V. The maximum over V, which is 11, is denoted by $\lceil/V$.

Of necessity, this must be only a brief survey of APL. Other features, particularly those operations on arrays which give the language much of its power for expressing long calculations in a line or two of programs, are set out at much greater length in Refs [A5, A6]. Here, the object is simply to show how a new approach to numerical programming which concentrates on compact and well-chosen notation can extend our ideas about how to write languages and programs in this field, which has been dominated for so long by FORTRAN conventions.

3.2. LISP

Section 2 contains an introduction to the ideas of symbol-manipulation and of list-processing at the same time. In all of the symbol-manipulating languages discussed below, the list is the basic structure in storage, but that does not imply that the two ideas are the same. In principle, one can write symbol-manipulating programs (e.g. in FORTRAN, by heavy use of the A format) without lists, and programs in a list-processing language which operate purely on numbers or lists of numbers. For most cases, however, calculations involving symbol-manipulation seem to have the property that intermediate and final results are of uncertain length, so that the natural medium of storage becomes the list, with pointers, rather than the FORTRAN array. Thus list-processing and symbol-manipulation are intimately connected, but they are still not the same thing. Having made this disclaimer, I hope that I can move on safely to the list-processing aspects of LISP first. I quote LISP as the principal example of a symbol-manipulating language because of the historical importance of LISP, the considerable number of uses (see Refs [C4, C5]) to which physical scientists have put it already, and the fact that implementations of LISP exist for all large scientific computers.

We left LISP in Section 2 at the stage (2.2) where the syntax had been completed by the definition of "S-expression". To repeat (2.2), we have

$$\langle \text{S-expression} \rangle ::= \langle \text{LISP atom} \rangle \mid (\langle \text{S-expression} \rangle . \langle \text{S-expression} \rangle)$$

The period "." is thus a special symbol, because it denotes the operation of concatenation of S-expressions into more complicated S-expressions. Every building-up operation in pure LISP must therefore make heavy use of periods. For example, some atoms in LISP are NIL, EF, CD, and AB. These are also S-expressions, by the first alternative in (2.2). By the second alternative, so are (EF . NIL), (CD . (EF . NIL)) and (AB . (CD . (EF . NIL))). In the structure (EF . NIL), we are using the period as an operator to associate EF and NIL. Given the conventional

partitioning of a word in the computer's free storage for LISP, the simplest association of EF and NIL can be arranged if we put a representation of EF into the left-hand half of a word and a representation of NIL into the right-hand half. In the programmer's form of LISP, (EF . NIL) is the equivalent of this single-word structure. (CD . (EF . NIL)) behaves like (CD. x) and requires a new word with CD in the left-hand half and a representation of x in the right-hand half. But here x is too large to fit into half a word, so the "representation of x" is a pointer which points to the word that contains (EF. NIL). If I repeat this argument with (AB. (CD. (EF. NIL))), I get a three-word structure whose specification is identical to Fig. 1. (Remember that the diagonal bar in the last half-word of Fig. 1 is shorthand for NIL.) In Section 2, though, Fig. 1 is described as the LISP representation of (AB CD EF), which is an easier and more natural piece of information than the one with many periods and brackets which I have just built up from (2.2). For convenience, then, LISP accepts the two forms (AB CD EF) and (AB . (CD . (EF . NIL))) as being equivalent; the two notations are called list notation and dot notation respectively. To get from one to the other, it is only necessary to remember the two identities written below. In each identity, dot notation is given on the left and list notation on the right.

(x . NIL) = (x)

(x . (y)) = (x y)

In the identities, x is any S-expression and y is anything.

The basic constructive operation in LISP is the operation of joining two S-expressions (e. g. x and (y)) together to form a compound S-expression (e. g. (x y)). All S-expressions can be built up eventually by repeated uses of the basic operation of construction. In LISP, this operation is the function CONS. CONS applied to arguments x and (y) gives the value (x y); inside the system, CONS takes a new word W off the free-storage list and puts a pointer to x in the left half of W and a pointer to (y) in the right half of W. Conversely, LISP also needs functions to resolve a compound structure into its "left" and "right" parts. The function CAR applied to (x y) has the value of x (i. e. it looks into the left half of W to see what it can find there), and the function CDR has the value (y) (i.e. the quantity which is "found" in the right half of W). These three functions are conceptually the most basic functions of LISP and all list-processing systems contain them under some name or other. The names CAR and CDR permit one helpful abbreviation: frequently we need to apply chains of CAR and CDR operations, e. g. "CAR of CDR" or "CDR of CDR of CAR". The abbreviations here, which LISP recognizes, are CADR and CDDAR respectively. The complete class of legal abbreviations, which contains a total of 28 functions, has members with names CnnR, CnnnR, and CnnnnR, where each n can stand for either A or D.

Having disposed of the necessary background for LISP here and in Section 2, we can now go on to consider individual parts and features of the language.

(a) Boolean logic and predicates

As in ALGOL, there are two special LISP atoms whose job it is to represent <u>true</u> and <u>false</u>. These atoms are T and NIL. (The two uses of

NIL which we now know do not conflict with each other.) Predicate functions, whose purpose is to ask questions (e.g. "Is x NIL?" or "Is x an atom?"), are functions whose values are either T or NIL in LISP.

The most useful predicates in the LISP language are given below. Where I explain the value of a predicate in terms of x and y, I mean that the predicate has these two arguments. If I talk only of x, the predicate has only one argument.

NULL: True (T) if x = NIL, and false (NIL) otherwise.

ATOM: T if x is an atom, NIL otherwise.

NUMBERP: T if x is a numerical atom, NIL otherwise.

EQ: T if x and y stand for the same atom (for the sake of uniformity among all LISP systems, we should assume that EQ gives the right answer only for non-numerical atoms), NIL if x and y are different atoms, otherwise the answer depends on whether (T) or not (NIL) x and y start in the same place in storage.

EQUAL: T if x and y are equal S-expressions, and NIL otherwise. Thus EQ should be used to test for equality if x and y are known to be non-numerical atoms, but the slower function EQUAL should be used in all other cases.

ZEROP: T if x = 0, NIL if x is a non-zero numerical atom, and an error condition otherwise, e.g. if x = NIL.

LESSP: T if x < y (where x is the first argument of LESSP), NIL if the numbers x and y do not satisfy this condition. Otherwise, the same caution as for ZEROP applies.

MEMBER: T if the S-expression x is a member of the list y, and NIL otherwise.

AND: T if all of its arguments are S-expressions or evaluate to give S-expressions which are not equal to NIL, and NIL otherwise.

OR: T if any one of its arguments is an S-expression or evaluates to give an S-expression which is not equal to NIL, and NIL otherwise.

The predicates AND and OR accept variable numbers of arguments up to 20.

(b) Conditional expressions

The translation into LISP of the English "if p_1 then v_1 else if p_2 then v_2.... else if p_{n-1} then v_{n-1} else v_n" is not smoothly readable like (3.76), but it is still unambiguous. It is:

$$(\text{COND } (p_1\ v_1)\ (p_2 v_2) \ldots .\ (p_{n-1}\ v_{n-1})\ (\text{T } v_n))$$

which is probably the simplest form we can get when we apply the boundary conditions that the expression should be an S-expression free from visible

periods in list notation and should distinguish each pairing (p_i, v_i) of proposition and value clearly. LISP uses the function COND to do what IF does in many other languages.

For p_i we can write any S-expression which represents a proposition. A predicate function, with its arguments, is the obvious choice, but T and NIL are also valid propositions by themselves. Further, the convention in LISP is that any S-expression whose value is not NIL can be placed in a position p_i, and COND will then behave as if this p_i were just T.

COND may be used to define functions with a multiple choice of values or in some circumstances to produce transfers of control (like IF in FORTRAN), provided that some v_i is an S-expression corresponding to a transfer statement.

(c) Structure of function-definitions

We can write our own functions F_1 in terms of the functions F_0 already available in the LISP system to do slightly more complicated things than these functions will do. Then perhaps we decide that we want a further level of complication, represented by functions F_2 which we write in terms of F_1 and F_0. After going through about four layers of this process, we arrive at the most complicated programs which have been used in physics. One's "program" is a union of all the functions of each level F_i except F_0. In contrast to the situation in numerical languages, where there is usually a main program followed by a few subroutines or functions, a LISP program may consist of one short function f_N which is written in terms of a few functions in the class F_{N-1}, where each function in F_{N-1} calls perhaps one or two others in F_{N-1} and several in F_{N-2} .... and so on. The program then consists of a collection of function-definitions in no particular order, followed probably on the last card of the deck by a call to f_N, with the appropriate arguments, which sets off the entire calculation. Therefore, to write a program, it is only necessary to know how to define a function.

The expression which must be punched on cards or typed at a terminal to specify a function-definition is:

(name (LAMBDA arguments form))

where "name" is the name of the function, and "arguments" is a list of the names of the dummy arguments to be used inside "form", which is the body of the definition. After the reassurances in Section 2, we can be sure that it is legal to define the function recursively, i.e. to use "name" again inside "form" if we wish. Hence, we have all the information that we need to define the factorial function, using the LISP conventions that we have seen so far. It is

```
(FACTORIAL (LAMBDA (N)
     (COND ((ZEROP N) 1)
          (T (TIMES N (FACTORIAL (SUB1 N)))) )))
```

In this definition, SUB1 is a function which has a value equal to the value of its argument minus one. The complications (?) of form and brackets are all consequences of the conventions which we have seen already, in-

cluding the convention of prefix notation, in which operators are written to the left of all their arguments. Moreover, the operator-name and all of the arguments are always made into one list for simplicity, e. g. (ZEROP N) rather than ZEROP (N) or even (ZEROP (N)). The last two forms are illegal in this context.

Recursion and the use of COND go together well, as the last example shows. For people who prefer the sequential organization of steps, which is possible in FORTRAN, there is an alternative way of setting out a function-definition like a small (FORTRAN) program. The special atom PROG is used to notify the LISP interpreter whenever this is about to happen, and the outline of the definition becomes

(name (LAMBDA arguments

(PROG pvariables $x_1 x_2 x_3 \ldots x_n$)))

where "pvariables" is a list of the names of variables (apart from those named in "arguments") used only inside that small program or PROG feature, rather like the arguments for a value declaration in ALGOL. Each x_i is either an individual statement or an atom which serves as the label for statement x_{i+1}. Occasionally it is rather easier to define a function in terms of the PROG feature than by other methods, e. g. in the case of REVERSE, whose value is the reverse of the list which is fed in. The definition of REVERSE is

```
(REVERSE (LAMBDA (U) (PROG (X)
    G (COND ((NULL U) (RETURN X)))
      (SETQ X (CONS (CAR U) X))
      (SETQ U (CDR U))
      (GO G)          )))
```

This function returns (EF CD AB), given (AB CD EF), by successively picking off left-hand members of the PROG feature's copy U of the structure with CAR and placing them at the left-hand end of the list X with CONS (all program variables like X are initially given values of NIL). U is shortened by one element when CDR is called. Finally, when U has been reduced to NIL, the value built up on X is returned as the answer. The original copy of the input structure is left untouched, however. It is instructive to work through this example, using Fig. 1 as a reference if diagrams are helpful.

REVERSE displays all of the new characteristics of the PROG feature: statement-labels, GO, SETQ, and RETURN. SETQ is the binding operator inside PROG features: the first argument of SETQ stands for itself, while the second argument is evaluated. Similarly, the argument of the transfer command GO stands for itself. Everything else in the individual statements of the PROG is evaluated by the LISP interpreter.

To get FACTORIAL and REVERSE defined, it is not sufficient merely to put the definitions as shown into a card deck. A list of such definitions must first be given as an argument to the LISP function DEFINE. How this is done is shown in an example of a program (see (e) below).

(d) Quote

In any language which allows symbols and names as data, the designers have to contend with the problem of what to do about the ambiguity that a name may sometimes be required to stand for itself and sometimes for another quantity. Suppose that we wish to write a function FN in LISP in which the dummy variable happens to be X and which gives the atom X as a value if the value of the variable is a number and NIL otherwise. This is admittedly a confusing specification, but it serves to underline the problem of quotation. The answer has the form

(FN (LAMBDA (X) (COND ((NUMBERP X) ?) (T NIL)))

where ? cannot just be X, because then the function applied to any number simply gives back that number as a value. We need X with some form of quotation-marks, possibly like "X", but ? must also be a legal piece of LISP data. LISP's solution is to replace ? by (QUOTE X). Whenever a quantity q is to stand for itself and is being used in a context where the interpreter does not take this fact for granted (i.e. almost anywhere inside function-definitions except in the first-argument position for SETQ, or as the argument for GO – Refs [A10, A11] supply more information on this question), it is written as (QUOTE q).

(e) Structure of LISP programs

The last two cards in a deck which contains a simple LISP program are by convention

```
STOP))))))
    FIN
```

where FIN begins in column 8. The STOP card contains several right-hand brackets, one to stop reading of the deck and the other to fill in for brackets which have been left out of the program by mistake (so that the reading can still stop). Before that, in the program itself, there are 2n S-expressions. An odd-numbered S-expression 2m-1 ($m \leq n$) is either the name of a function or a nameless specification of a functional operation with the form "(LAMBDA arguments form)". The list of arguments for the function is then given in the position 2m. If the function 2m-1 has zero arguments, S-expression 2m is ( ); for one argument, the even-numbered structure is (a_1), for two arguments, $(a_1 a_2)$, and so on. The LISP interpreter treats the entire program as data, reading it exactly as it stands (i.e. it makes no attempt to evaluate the arguments) and then applying function 1 to argument-list 2, function 3 to argument-list 4, etc. The non-evaluation of arguments sometimes confuses the novice programmer who wants to carry out some kind of compound operation, so it is worth demonstrating with an example. If I punch

NUMBERP ((TIMES 2 4 6))

expecting the multiplication to give the number 48 and the resulting evaluation of the test in NUMBERP to return the value T, I shall be surprised

by the actual value NIL. It is actually the unevaluated expression (TIMES 2 4 6) which is inspected for "number-ness", and of course the result is NIL. To get the expected answer T, I need to punch

```
(LAMBDA (X) (NUMBERP (EVAL X NIL))) ((TIMES 2 4 6))
```

where I have put in the demand for an extra evaluation explicitly. The function EVAL has this purpose of evaluating its first argument and returning that value; in most simple situations the second argument of EVAL is just NIL.

Next, here is an example of a LISP "program". The function LTIMES has the job of giving the product of all numbers occurring in any list which is submitted to it, or NIL if there are no numbers in the list. The function ATOMSOF gives as its value a list of all the atoms which occur at the top level of a list which may also contain more complicated members. The function XN gives the set-theoretic intersection of its two arguments. Finally, we may want to define a function SUPERFN which can call all of these three, e. g. a function of two arguments which returns the product of all numbers which are common to the top levels of the arguments. (It will not be the most efficient such function, but it will be adequate for this demonstration.) A card deck which sets up the functions and then tests them is:

```
DEFINE ((
(LTIMES (LAMBDA (U) (COND ((NULL U) NIL)
      ((NUMBERP (CAR U)) (LTIMES2 U))
      (T (LTIMES (CDR U)))                    )))
(LTIMES2 (LAMBDA (U) (COND ((NULL U)1)
      ((NUMBERP (CAR U)) (TIMES (CAR U) (LTIMES2  (CDR U))))
      (T (LTIMES2 (CDR U)))         )))
(ATOMSOF (LAMBDA (U)  (COND ((NULL U) NIL)
      ((ATOM (CAR U)) (CONS (CAR U) (ATOMSOF (CDR U))))
      (T (ATOMSOF (CDR U)))       )))
(XN (LAMBDA (X Y)  (COND ((NULL X) Y)
      ((MEMBER (CAR X) Y) (CONS (CAR X)
                                (XN (CDR X) (DELETE (CAR X) Y))))
      (T (XN (CDR X) Y))              )))
(DELETE (LAMBDA (X Y) (COND ((NULL Y) NIL)
      ((EQUAL X (CAR Y)) (CDR Y))
      (T (DELETE X (CDR Y)))      )))
(SUPERFN (LAMBDA (A B) (LTIMES (ATOMSOF (XN A B )))     ))
      ))
ATOMSOF ( (A  (B C) 2 (E 3) FGH))
```

```
LTIMES (((A 2) HOMMES 40 CHEVAUX 8))
XN ( (AB CD EF) (GH CD 9) )
SUPERFN ( (A 2 B 6 C 7) ((W 2) X 7 Y 6))
STOP ))))))
      FIN
```

As you can see, the deck mixes what you might normally have called program and data. The first odd-numbered S-expression is DEFINE, so that the next S-expression (a list of one argument x, where x is a list of function-definitions), which is your "program", can be defined. When this task is finished, any function in the program can be addressed by name and used to process your data, as in the cases shown. In this deck, there are five applications of functions to data. As a result, the output contains the five answers

```
(LTIMES LTIMES2 ATOMSOF XN DELETE SUPERFN)
(A 2 FGH)
320
(CD)
42
```

in a better-annotated form than I have given them here.

This example displays many of the standard characteristics of LISP programs and therefore repays careful study. The best way of study is to work through the given tests of the functions with pencil and paper, regarding yourself as the LISP interpreter.

There is an introduction to LISP in Ref. [C4] and another example of a LISP program in Section 6.2.

(f) Further basic functions

In the most basic LISP systems for large computers, a little over 80 functions are available. We have seen about half of this collection by name here already. While some of the remaining functions are for advanced applications, others belong to useful categories which have not been considered so far. Eight functions which represent some of the basic categories are discussed below. The lower-case letters x, y, and z are used to stand for the values of their arguments when they are called from inside some other function-definition, e.g. one which the programmer may have written for himself.

(LENGTH x): The value is the number of elements in the list x at the top level, e.g. 3 for (A B C) and 2 for ((A B) (C D)). When x = ( ) or NIL, the value is zero.

(APPEND x y): If x and y are lists of arbitrary lists, the value is a single list formed by the concatenation of copies of x and y, e.g. when x = (A B C) and y = (D E F), the value is (A B C D E F).

(LIST x y z ...): LIST accepts variable numbers of arguments, between 1 and 20. The value is a single list exhibiting x, y, z ... as members. For example, if x = (A B C), y = (D E F), and z = 6, the value is ((A B C) (D E F) 6).

(SUBST x y z): The value is a copy of z which has been changed by the replacement of each occurrence of y at any level by x, i.e. it is a function for substitution of x for y in z.

(PRINT x): The value is x, although PRINT is interesting for its effect rather than its value. We may be satisfied with the answer which a function produces in some computation and which the interpreter automatically puts into the output, but on some occasions we may want to have news (e.g. for debugging) of intermediate results like x which are achieved in the middle of the computation. We can cause this information to appear also in the output by changing x to (PRINT x) at the relevant parts of the program.

(MAPLIST x f): Frequently we need to perform operations on a list x which involve repeated applications of some function f to x, then to CDR of x, then to CDDR of x, etc. This is a global mapping operation in which the effect of f is mapped onto x in a particular way. For flexibility it may be a good thing to have a standard LISP function which can accept functional arguments like f, control their application to f, and return the value of the mapping. MAPLIST is the simplest of these mapping functions. It behaves as if its LISP definition were

```
(MAPLIST (LAMBDA (X,FN) (COND ((NULL X) NIL)
      (T (CONS (FN X) (MAPLIST (CDR X) FN))) )))
```

With MAPLIST, it becomes possible to define LISP functions in alternative styles, so that one can choose the best or most efficient style for a given situation. One example of alternatives is a function DOUBLE which, given a list purely of numbers, returns a list in which each number from the original list is doubled. Recursively the definition is

```
(DOUBLE (LAMBDA (U) (COND ((NULL U) NIL)
      (T (CONS (TIMES 2 (CAR U)) (DOUBLE (CDR U)) )) )))
```

but with the use of MAPLIST we can also write

```
(DOUBLE (LAMBDA (U) (MAPLIST U
      (FUNCTION (LAMBDA (J) (TIMES 2 (CAR J)) ))) ))
```

The special atom FUNCTION is used to label any functional argument for a mapping function.

(DEFLIST x y): The form of x is a list of two-element sublists, e.g. $((a_1 v_1) (a_2 v_2) \ldots)$. The second quantity y has an atom as its value. The value of DEFLIST is the list $(a_1 a_2 \ldots)$, but this is unimportant beside the effect of the function, which is to associate with each a_i the property v_i under the indicator y. If y = EXPR, the operation of DEFLIST does exactly

the same as (DEFINE x), i.e. the indicator which the system maintains internally to identify stored function-definitions is EXPR. A quick look back at the program under (e) above shows that the argument for DEFINE has the required form for the x given here, so that the stored function-definition under the indicator EXPR for any function-name is always (LAMBDA arguments form). DEFLIST, a generalization of DEFINE, permits us to decide on as many properties as we like for any atom and to associate them with the atom by using DEFLIST repeatedly. How the association is maintained inside storage in such cases is described in Section 5.5. In particle physics and the analysis of Feynman diagrams in LISP (Ref. [C8]), the atoms which identify particles are labels P1, P2 ... for momenta, but we can carry along all the other discrete information about particles by using DEFLIST to associate with, e.g., P1 a mass under the indicator MASS, a spin with SPIN, a polarization with POL, and so on.

(GET x y): Having used DEFLIST to store information, we need an inverse function to recover it. This is GET. If x stands for an atom and y is an indicator, the value of GET is the property stored under that indicator. If x does not have the property y at all, the value of GET is NIL.

(g) ERRORSET

Sometimes we may have a good idea of where errors may occur in a complex program but be unable to try out our ideas without running the program to see what happens. In terms of LISP, the evaluation of some error-prone expression S should behave as: "if S is error-free, the answer is S, but otherwise some special value is returned to denote an error". Thus, if we make an error in S, the computation does not collapse as in most languages, but we can recover control and direct the computation to try some other alternative. In LISP, this highly valuable state of affairs can be brought about with the use of the function ERRORSET to control the evaluation of S. The form of calls to ERRORSET is

.... (ERRORSET s n i a) ....

where s is the expression to be evaluated, n is the maximum number of CONS operations to be allowed before an error trap occurs (which is useful in detecting infinite loops), i is an indicator whose value is T if we wish error messages to be printed out but NIL otherwise, and a is a list of pairs of arguments from s and their values. For example, if we have a piece of program in which s stands for (FN X Y), and the values of the variables are respectively (A B) and 5, then the value of a is ((X A B) (Y . 5)).

If the evaluation of the expression denoted by s is correct and has the value v, then ERRORSET returns the value (v). If there is an error, the value is NIL. Therefore the user or his program can identify errors immediately.

Once the unusual idea of an error-recovery operation like ERRORSET has been established, intending users can easily think of improvements, e.g. a range of atomic values 1, 2, 3 ... instead of NIL in the case of errors, with each value signifying a different type of error so that the program can react accordingly. Some of these facilities are being added to

languages which I have not surveyed here. The important contribution of LISP has been to show that a function ERRORSET is practical to build and operate.

(h) Uses of LISP in physics

Most of the symbolic computations with applications to physics have been in LISP (see Refs [C4, C5]). In turn, most of these computations have used the systems REDUCE-1 and REDUCE-2. For a physicist who does not wish to learn LISP in order to use existing programs, the chief barriers to understanding are brackets and prefix notation. Most LISP-based systems are now either evolving into languages (as REDUCE has done already) or are equipped with special functions which accept FORTRAN-like input in infix notation and translate it secretly into LISP before processing it. This makes the user's life easier. More details are given in Sections 3.2.2, 6.2 and 6.6. However, if you have problems for symbolic computation which do not seem to be suited to existing programs, there is enough information here to enable you to make your own experiments with LISP. The rewards which a study of LISP has to offer are substantial, quite apart from the consideration of whether or not you have any immediate physical problems to solve.

3.2.1. POP-2

LISP and POP-2 grew up in the same kinds of environment, in the neighbourhood of designers and users with an interest in the topic of artificial intelligence. The most noticeable differences of POP-2 from LISP at first inspection are that POP-2 looks much more like ALGOL and does not seem to contain so many bewildering brackets as LISP. An explanation for the first difference is that exposure to the influence of ALGOL is a more or less continuous process in the United Kingdom but a rather random one in the United States. There are several reasons for the second difference, but one which may be quite convincing is that LISP was first implemented in 1959, when large conversational computing systems were not available, but POP-2 was built as a conversational language. An impatient user sitting at a teleprinter is not rendered any more patient by the need to count his brackets as carefully as a LISP programmer must do.

The absence of bracket-counting problems, and the presence of some fast arithmetical facilities for conversational users, means that POP-2 resembles FRED for some simple applications, e.g.

```
A = 5 ↑ 2 + 5/2
B = 2*A - 1
B
```

which is the sequence of FRED instructions that causes the eventual printing of 53, is rendered as

```
VARS  A, B;
5↑2 + 5/2 -> A
2*A - 1 -> B
B =>
```

in POP-2. (Note the requirement that variables are declared, to reserve space for them, and also note the conventions for binding and printing.) When the applications are symbolic, the notation of POP-2 is close to that of either ALGOL or REDUCE, which makes conversational computing easier than in LISP. In the example of the definition of the function LENGTH (see Section 3.2 (f)), contrast the LISP input

```
DEFINE ((
(LENGTH (LAMBDA (X) (PROG (N) (SETQ N 0)
G (COND ((NULL X) (RETURN N))) (SETQ N (ADD1 N))
  (SETQ X (CDR X)) (GO G) ))) ))
```

with the rather simpler POP-2 construction

```
FUNCTION LENGTH X; VARS N;
0 -> N;
G: IF NULL(X) THEN N EXIT;
N + 1 -> N; TL(X) -> X; GOTO G END;                    (3.84)
```

which also displays the POP-2 conventions for the LISP concepts of GO, PROG, SETQ and RETURN. The basic list-processing functions CAR and CDR have the mnemonic names HD (head) and TL (tail), an old British convention from the days of the language CPL which makes it impossible to abbreviate chains of HD and TL functions in the manner of LISP.

The main difference between POP-2 and LISP programs, as in (3.84), is in readability. Basic symbol-processing functions tend to have the same names (apart from CAR and CDR, and also CONS, which is the infix operator :: in POP-2) and basic ideas in the two languages, but the added virtue of POP-2 is that it contains extra features that extend one's ideas about symbol-manipulation. To somebody brought up on LISP, a reading of Ref. [A16] causes the same kind of educational mind-stretching that a reading of an APL manual produces in somebody whose sole background is in FORTRAN. The following deals with some of the more interesting extra features.

3.2.1.1. Dynamic lists

The LISP lists which we have seen so far are known as "static lists" in POP-2. Even if we add and subtract elements, the lists are still "static" in the sense that they are always sitting inside the storage and occupying some space there. However, there are occasions when calculations need to operate on collections of data of uncertain but great length whose natural representations are lists. Consider the possibility that we need a list L of prime numbers in increasing order, so that we can find and make calculations with the n^{th} prime. If nothing is known about n in advance except that it may be quite large, a static list of primes is not appropriate because it may occupy a large amount of storage between uses. There are further chances of inefficiency if we have generated and stored statically either less than n primes or more than the largest n which is wanted. The best

solution is to have L associated not with a static list but with a rule R which specifies how to generate the n^{th} element of L, assuming that the list of n-1 elements exists. This is possible in POP-2: L is then called a dynamic list.

To define a dynamic list, we first define R (usually as a function) and then associate R with L by means of the function FNTOLIST which is reserved for this purpose. To take a simpler example than the list of primes, for which R is somewhat complicated, suppose that we need a dynamic list of harmonic numbers H_n defined as the sum of the reciprocals of j as j runs from 1 to n. The code is

```
VARS M,N;  0 -> N;  0 -> M;
FUNCTION HNEXT;  N + 1 -> N;  M+1.0/N -> M;  M  END;
FNTOLIST(HNEXT) -> L;
```

Following this construction, L can be processed as if it were an ordinary list, e.g. a function NTH(n,x), which selects the n^{th} element of the list x, will give the 15^{th} harmonic number if it is called in the context NTH(15,L).

Other uses for dynamic lists, which are obviously very economical in occupation of storage, are given in Refs [A16, B1].

3.2.1.2. Pervasive influence of the push-down list

The LISP programmer uses the language's internal push-down list implicitly whenever he calls a recursive function and whenever one of his functions f_1 must have partial results of its computation stored before it can call some other function f_2. However, there are no LISP functions which he can use to operate explicitly on this push-down list or stack. In POP-2, any computed quantity goes onto the top of the stack, and the operator - >, whose first purpose is to achieve bindings, actually has the more basic purpose of taking the top element off the stack and binding it to the quantity to the right of the > sign. In other words, the step 5↑2 -> X says "calculate the square of 5, put the result at the top of the stack, and then remove the top element of the stack and bind it to X".

The consequence of this convention is that the two halves of what is taken for granted as a single binding operation in other languages each have a separate interpretation in POP-2. The statement

```
Y
```

means "put the value of Y on top of the stack", and this meaning extends to a succession of quantities Y, Z, ... which are added to the stack one after the other. The statement

```
-> Y
```

means "take the top element off the stack and bind it to Y".

The stack conventions make it possible to give sensible meanings to functions which return more than one value, e.g. the quotient-plus-remainder function //, whose values in a case such as 37//13 are 11, 2.

Since these both go onto the stack, the way to assign the quotient to Q and the remainder to R is to write

37//13 - > Q - > R

This flexibility is extended to list-processing by means of functions like DEST, which takes a list x as its argument and puts both HD and TL of x onto the stack as values. If we want to write a function FN of x with n values $v_1, v_2, \ldots v_n$, we have only to make a definition with the following characteristics:

FUNCTION FN X; ... ; $v_1, v_2, \ldots v_n$ END;

Finally, the explicit accessibility of the stack allows several different meanings for closely-related statements. For example, if I use SIN(1.57), the sine of 1.57 is placed onto the stack. If I choose SIN( ), this is an instruction to calculate the sine of whatever is on top of the stack (the call .SIN has the same effect, where the period behaves like the EVAL command in LISP). It is also possible to write only SIN, in which case the "value" of SIN is loaded onto the stack. This is a risky business in LISP, but the POP-2 rule is that the value of the name of a function is the definition of that function.

3.2.1.3. Definition of infix operators

Sometimes, for convenience of notation, a frequently-used prefix operator or function-name is easier to write as an infix operator. There are two steps to this replacement: firstly, the association of prefix and infix operators, and secondly, the assignment of a precedence to the infix operator so that it has an unambiguous meaning in unbracketed expressions, where existing prefix operators like +, *, and / have the implied precedence which we meet in languages like FORTRAN. Precedence is described by an integer N, the highest-priority operators having the smallest N (e.g. N = 2 for ↑, 5 for +, and 7 for =, the test for equality).

The second step is taken care of by a single POP-2 statement of the form:

VARS OPERATION precedence operator;

while the first step is simply a matter of associating a prefix operator or function-definition with "operator". As an example, consider the definition of a predicate operator / = (as near as we can get to ≠ in POP-2 in the absence of that character) for "not equal", with the same precedence 7 as =. We make it with

VARS OPERATION 7 /=;

LAMBDA X Y; NOT(X = Y) END - > NONOP /=;

where NONOP acts here like QUOTE in LISP. Note that - >, which assigns a value, assigns a function-definition to /=. Unlike LISP (which has the use of DEFLIST and an indicator for every occasion), then, POP-2 normally allows an atom to represent either a variable or a function-name, but not both at once.

3.2.1.4. Selectors and updaters

If we write a binding operation like Y = CAR(X) in a FORTRAN-like notation, it can be understood to mean "set Y to the first element of X". The operation CAR(X) = Y, or "set the first element of X to the value of Y", is not defined in most languages despite the obvious convenience of the notation. (In the same notation, the LISP alternative would be simply RPLACA(X, Y) — not very revealing.) POP-2 allows certain named functions F to occur on either side of the binding operation, which helps to increase the convenience of programming. Naturally, the system does not make use of exactly the same function-definitions in both contexts; which one of two alternatives is actually used is determined from the position of F. On the left of - >, F is called a selector, and on the right of the operator it is an updater. The functions HD and TL behave in this way.

Any function F which we define is automatically a selector. To give F also the meaning of an updater, we invent and define some new function, say XF, which can actually perform the updating operation which we would like F to carry out when it is placed to the right of - >. Then, the final step to give F the required property is just

```
XF - > UPDATER (F);
```

This simple and helpful facility is also available in SNOBOL and PL/I.

3.2.1.5. Records

Although this subsection carries an insignificant numerical heading, it raises an important new question in list-processing and symbol-manipulation. Suppose that we want to make our computations on structures which are too complex (e. g. with too many subdivisions of data or too many necessary pointers) to fit easily into the LISP list, which is the only pointer-dependent method of storage described in detail above. Must we then go to a new language which is built around exactly our structure? Surely not, because unless we are very lucky we shall not find a system programmer somewhere who has guessed our needs in advance and has already written a language to meet them. The answer to our problem is contained in any language in which there are commands to create general new data-types and structures which the user can tailor to his own demands. Although L[6], SNOBOL and PL/I (out of the languages mentioned below) also have such features, POP-2 embodies them in a particularly convenient form.

An item of any general data-type is a record in POP-2. If a record contains n distinct components of information, we need at least n+2 functions to manipulate it — n of them to give as values the corresponding components of the record, one (of n arguments) to create a record and fill it with new information, and one to abolish it. The improvement in convenience which POP-2 offers over the other languages which handle general data-types is the ability to define and name all n+2 functions at the time that the structure of any particular type of record is first stated. The POP-2 function RECORDFNS (x, y), where x is a general name for the class of record being defined, creates a pattern for all records of type x, and places n+2 function-definitions on the stack. The top definition is for record-creation, the one underneath is for record-abolition, and the

remaining n definitions are for access of components n, n-1,...,1. The argument y is a list of n elements e_i which specifies the sizes and natures of the n components. The value of e_i is m if component i is always to be a non-negative integer less than 2^m, COMPND for certain compound items (e.g. non-numerical data), or 0 if the component obeys no particular restrictions. Thus if we want to build a record of type FARM whose components are respectively the number of cows, dogs, cats, chickens and armadillos on the farm, we should write

```
RECORDFNS(FARM,[8,4,5,14,0]) -> ARMADILLOS -> CHICKENS
    -> CATS -> DOGS -> COWS -> ABOLISHF -> CREATEF;
```

I have guessed at the maximum populations in the second argument, which is by convention of POP-2 delimited by square brackets. Subsequently I can create a farm MCDONALDS by a statement like

```
CREATEF(200,10,6,1500,139) -> MCDONALDS;
```

and later see what the armadillo population is by

```
ARMADILLOS(MCDONALDS) => ;
```

which prints 139. The functions which assess individual components may be used as updaters as well as selectors, e.g. I may wish to record an armadillo plague on McDonald's farm by writing

```
2844 -> ARMADILLOS(MCDONALDS);
```

Finally, because I may forget what data-type MCDONALDS represents, I can use an interrogative function DATAWORD whose value is the type, e.g.

```
DATAWORD(MCDONALDS) =>
```

prints the word FARM.
Europeans unfamiliar with the concept of "armadillo" should consult Fig. 4.

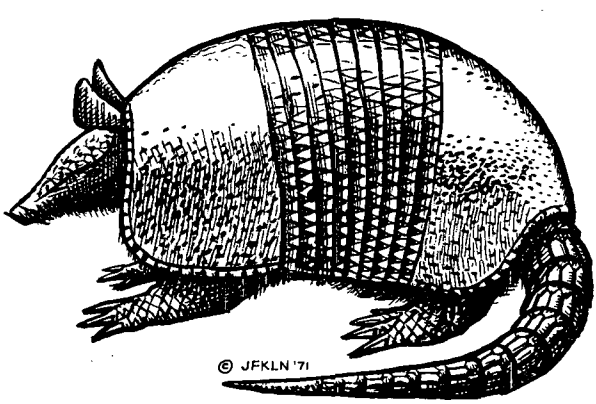


FIG. 4. Armadillo, by courtesy of Jim Franklin.

3.2.1.6. State-saving and backtracking

As a generalization of the idea behind the ERRORSET function in LISP (Section 3.2 (g)), which allows local recovery of control of a computation in the case of an error, it may be necessary in some computations (e.g. "learning" programs in artificial intelligence, or imitations of parallel processing in simulations of systems) to jump back to some previous state S of the entire program, recovering all or any specified part of S on demand. A recently added feature of POP-2 allows jumps back to S, given that a function APPSTATE is used at the original point of achievement of S to preserve information about S for later reference. If the value of APPSTATE is bound to some x, then later calls of REINSTATE(x) cause jumps back to S. Values of global variables (see Sections 2 and 4.3) and records cannot yet be saved as part of S. Any local variables may have their values preserved, and a third function BARRIERAPPLY is used to identify them correctly, amongst other things.

The facilities described here complement rather than overlap those of LISP, because there is no function ERRORSET in POP-2. However, it is possible that the generality of state-saving in POP-2 will be extended in the future. A LISP system for artificial intelligence, with facilities for state-saving and backtracking, is under development by E. Sandewall and his colleagues at Uppsala University.

3.2.2. REDUCE or NEW LISP

The previous history of REDUCE is well documented (Refs [A17, A18, C9, C10]). Originally, as REDUCE-1, it was a system of LISP programs for symbolic manipulation and specialized calculations (e.g. evaluation of traces of products of Dirac matrices in quantum field theory) in particle physics. As physicists preferred FORTRAN-like notation, e.g.

UB(L1,P4)*G(L1,MU)*(G(L1,P6)+PM)*G(L1,NU)*U(L1,P2) (3.85)

for

$$\bar{u}(p_4)\,\gamma_\mu(\gamma\cdot p_6 + m_\rho)\,\gamma_\nu u(p_2)$$

on fermion line 1, rather than an equivalent prefix-notation form in LISP, the LISP programs soon grew functions MATHPRINT and MATHREAD to communicate between expressions like (3.85) and S-expressions for input and output. Originally the input consisted of a series of commands, ranging from simple subscript-declarations such as

```
INDEX MU, NU;
```

up to the maximum complexity of SIMPLIFY x, where x was any expression of the form (3.85). The syntax was minimal. The subsequent history of REDUCE-1 and REDUCE-2 has been a history of increasing capabilities in syntax, with the aim of achieving agreement with ALGOL rules plus extensions for symbolic manipulation. The ALGOL-like steps have until recently still been processed by greatly extended versions of the early functions MATHPRINT and MATHREAD, written in LISP, but now the set

of permissible statements for input to REDUCE-2 looks to the user like a language "REDUCE", and the system REDUCE-2 (previously in LISP) is presently being rewritten in REDUCE. As an example of contrasting styles, a definition of the function REVERSE (see Section 3.2(c) for the LISP definition) in REDUCE is

```
LISP PROCEDURE REVERSE U;  BEGIN SCALAR X;
G: IF NULL U THEN RETURN X;
X := CAR U . X;
U := CDR U;  GO TO G;  END;
```

It is easy to see how the LISP-like operations are adapted to the syntax of ALGOL and also what is the justification for the new types lisp (for procedures producing S-expressions) and scalar (an exact analogue of PROG in LISP). The prefix operation CONS has become an infix operator "." in REDUCE. Lastly, there is no ambiguity if functions of one argument do not have their arguments surrounded by brackets.

Because any LISP function can be rewritten in this syntax as a "LISP procedure" whose notation is quite close to one's natural usage, the syntax is an ideal basis for the teaching of list-processing. The general feeling at present is that the name REDUCE, which has no obvious LISP connotations, should be changed to reflect them; suggestions so far have been REDUCE LISP, RLISP, and New LISP (where LISP → Old LISP). As the last of the three alternatives is my own suggestion, I shall use it further here, but only history can decide what the new name shall be.

I shall not go into a long description of REDUCE here, because of the existence of Ref.[A17] and because Professor Hearn has described REDUCE in paper SMR-9/27 in these Proceedings, but some of the special features of the language are potentially very useful and should not be overlooked in a general survey.

3.2.2.1. Ordering

In POP-2 it is possible to define any new operator with a given precedence with respect to the existing operators, but to do so it is necessary to know the precedence-numbering of those operators. The requirements of REDUCE are not quite so stringent. If x is an operator to be defined with a precedence which the user desires to be just ahead of or equal to the precedence of an existing operator y, the declaration

```
PRECEDENCE x,y;
```

achieves that result.

Long and badly organized symbolic output is one of the banes of the life of the user of programs for symbol-manipulation. REDUCE alone seems to concede that fact, and to do something about it. If it is desirable to have quantities a,b,c,d... in that order everywhere in the output (and especially inside individual terms), then

```
ORDER a,b,c,d...;
```

is the REDUCE command which sets up the correct ordering. Finally, the benefits of factorization of common sub-expressions in long symbolic output are obvious. If such sub-expressions are a, b, c ... , then

```
FACTOR a,b,c ... ;
```

causes each one to be factored in the output. A quantity can be a variable-name X, a prefix function of X, such as COS(X), or a prefix function of anything, e. g. merely COS.

3. 2. 2. 2. Substitution

How to deal with substitution is one of the key ill-formed problems of symbolic manipulation. The user of such programs never gets to see the substantial amount of work which goes on behind the scenes to make substitution more efficient, but occasionally the appearance of a spectacular new operation for substitution in a symbolic language may alert him to the fact that progress is being made in this field. REDUCE contains a remarkably wide range of facilities for substitution, which is the product of a great deal of heroic work. The fact that one can describe a certain type of substitution easily in REDUCE or English should not be allowed to obscure the other fact that no actual implementation of substitution is trivial.

The most popular type of substitution is one in which a given expression is to be replaced everywhere by another expression. REDUCE uses a command LET for such cases, e. g.

```
LET Y↑3 = 5*K - 19, Z = COS(X);
```

Moreover, the substitution may extend to a class of expressions, not just to an expression in specific variables, e. g. we may want to make the approximation $\cos x = 1 - x^2/2$ for all x, not just x = X. In REDUCE, we can do this by writing

```
FOR ALL X LET COS(X) = 1 - (X↑2)/2;
```

In other words, LET accepts modifiers, including the usual range IF, UNLESS, etc. which we find in FRED. Therefore, LET can set up conditional substitutions. A final use of LET is to declare rules which improve simplifications, e. g.

```
FOR ALL X LET LOG(E↑X) = X;
```

or extend existing functions, e. g.

```
FOR ALL Y LET DF(FN(Y), Y) = G(Y);
```

where DF performs symbolic differentiation.

There are cases where casual use of LET can give the system an acute attack of indigestion, e. g.

```
LET A = B + C, C = D + A, X = X + 1;
```

because after each substitution with LET the system checks again to see if renewed substitutions are possible within the existing set of rules. Whenever a replacement is to be made just once, REDUCE uses SUB, a different operator. An example of its use is

```
SUB(X, X+1, 2*X - 41);
```

i.e. not only does it make a single replacement, but the user is required to say exactly where it is to be made.

3.2.2.3. Matrix operations

A variable-name used to denote a matrix must be so declared via the MATRIX command. Then, any such variable can be bound to a matrix with MAT, whose arguments are lists r, where each r gives the elements of a specific row, e.g.

```
U := MAT((1,0,0),(0,1,0),(0,0,1));
```

which sets U equal to the 3×3 unit matrix.

The advantage of the compressed notation is that global operations on matrices, e.g. U*V for a matrix product and U↑(-1) for matrix inversion, can be denoted simply. Some numerical languages (e.g. PL/I) offer the same notation for numerical matrices (but remember that here the matrix elements can also be arbitrary symbolic expressions!). As for substitution, the simplicity of notation hides a great deal of work on ingenious algorithms for symbolic operations, which are now a part of REDUCE.

Calculation of traces and determinants of matrices is rendered easy by the existence of REDUCE functions TRACE and DET.

3.2.3. L^6

The explanation of the peculiar name is that L^6 is Bell Telephone Laboratories' Low-Level Linked List Language. While L^6 has been overtaken in terms of widespread use by other languages for list-processing, it contains two properties which are of interest in a general discussion of list-processing languages. These are:

(1) Ability to handle a programmer's definitions of list elements with 2^n words of storage and arbitrary partitioning in each word, instead of the single word with two equal-sized partitions in LISP. (We have already seen this property in records in POP-2, but L^6 preceded POP-2, and unlike POP-2 it allows the user to say exactly where the boundaries between partitions shall be drawn.)

(2) Individual instructions which come much closer to the appearance of machine code than in most languages and which are in 1:1 correspondence with machine-code instructions in some cases.

The advantage of these two properties, for occasions when it is needed, is that a programmer in L^6 is much closer to the machine-code level than he is in rival languages. He must therefore give up the convenience of

having his programs resemble natural language, but in return he gets the convenience (characteristic of machine code, if one has the patience to use it) of being able to write programs which may run faster and make more efficient use of storage than programs in higher-level languages.

3.2.3.1. Definition of elements of storage ("blocks")

When I want to define a block (an example of the entity described in (1) above) in L^6, I must reserve a space for it inside storage, define the partitions which I want inside each block, and construct some naming scheme which gives each partition (and the block itself) a name which is unique. The naming system in L^6 is unusual but effective. There are 26 fixed starting-places in storage, named A through Z, in which pointers to blocks can reside. If I want to reserve a block of 2^n words which is identified by a pointer from starting-place p, I write the L^6 instruction (p, GT, n), where GT denotes "get block". (Most L^6 instructions have this form, with the principal argument first, the named operation second, and subsidiary arguments in the remaining positions.) Next, I can define partitions ("fields") inside the block, by means of a series of instructions

(m, Df, first, last),

where each field has a single-character name which is substituted for f, m is the number of the word (beginning from 0) in which the field is to lie, and "first" and "last" are the numbers of the first and last bits of the field. For example, if I wish to imitate IBM 7090 LISP structures in L^6 (single-word elements, 15-bit fields for CAR and CDR parts inside a 36-bit word with bits numbered from 0 to 35), I must write a sequence such as

(p, GT, 0) (0, DA, 3, 15) (0, DD, 21, 35)

for some p. Subsequently, I can find the field equivalent to CAR(p) by writing pA, and the CDR field by using pD.

To set up lists in the LISP sense, I must repeat the sequence of operations given immediately above (or write an L^6 subroutine for it) for each new element of the list, but with p replaced by some $q \neq p$, then insert into the field pD the pointer found in q. In other words, I am doing step by step in great detail what LISP does automatically with CONS. This is a good example for the contention that L^6 forces the programmer to think carefully on a level close to that of machine code.

3.2.3.2. Samples of operations

If we are to allow the range of data which can be accommodated in LISP, fields in L^6 must be able to hold pointers, numbers, or Hollerith strings (in imitation of non-numerical atoms). For most L^6 operations on pointer-like fields, there are separate names for operations on fields of the other two types, e.g. (K, E, L) tests for equality of the fields K and L, while (K, EO, 56) tests for the presence of the octal number 56_8 in the field K, and (K, EH, L) tests for the presence of the literal atom L in K. In LISP, the same tests are, respectively, (EQUAL K L), (EQUAL K 56Q), and (EQ K (QUOTE L)). Wherever appropriate, it can be assumed that for any operator x there are also operators x0, xD (decimal) and xH.

A further selection of available operators is:

A - arithmetical addition

S - arithmetical subtraction

M - arithmetical multiplication

V - arithmetical division

DP - copy an entire block

P - make the principal argument (which precedes P) point to the same block as the subsidiary argument

O - logical "OR"

N - logical "AND"

IN - the only input operation: (p, IN, n) reads a card and puts the first n columns of the card, left-shifted, into p.

PR - printed output

PU - punched output

BZ - representative of a conversion of types for input or output, BZ converts blanks to zeroes.

FC - (S, FC, x) saves the current contents of field x on an auxiliary push-down list which L^6 maintains for that purpose. (R, FC, x) takes the top element off this list and puts it in x.

3.2.3.3. Example of a program

One short example is sufficient to give the flavour of L^6 programs and to display the distinctive appearance which the syntax dictates. Suppose that we want to write a function whose job it is to see if x is a member of a LISP-like list y, and to print the answer, which is either TRUE or FALSE. In New LISP a definition is

```
LISP PROCEDURE MEMBER* (X, Y);
  IF NULL Y THEN PRINT 'FALSE
    ELSE IF X = CAR Y THEN PRINT 'TRUE ELSE MEMBER* (X, CDR Y);
```

In L^6, a corresponding description is

```
        THEN (S, FC, Y)
ROUND   IF (X, E, YA)  THEN (4, PRH, TRUE) RESET
        IF (YD, EQ, 0) THEN (5, PRH, FALSE) RESET
        THEN (Y, P, YD) ROUND
RESET   THEN (R, FC, Y) DONE
```

where I use the standard convention that 0, as a pointer to word 0, acts like the LISP list-terminator NIL, and the labelling of A and D fields is

by analogy with Section 3.2.3.1. The extra L^6 feature which is missing from the New LISP program is the provision to save a picture of the complete y before the calculation (because (Y, P, YD) changes the pointer out of the base field Y at each step, therefore losing the original y) and recover it afterwards with (R, FC, Y). LISP must of course do the same thing, but it does so automatically. Once again, therefore, L^6 requires the programmer to think about the question, and hence to work one level closer to machine code.

For any L^6 statement, the syntax is rather bare. In particular, the fact that many lines begin with THEN is a consequence of the absence of an ELSE option. Apart from the forms shown above, the options IFANY, IFALL and IFNALL ("if not all") are available.

An interesting alternative to L^6 which permits sections of programs that display both of the special properties which I have concentrated on in this section, but which is embedded in an ALGOL-like syntax, is the language PL360 (Ref. [C11]), not to be confused with PL/I or APL360 or APL. I do not discuss PL360 here, because I have not yet had a chance to use it.

3.3. SNOBOL

String-manipulation, the third main class of programming, leads to languages which are distinct in appearance from anything else which we have explored in Section 3 up to this point. SNOBOL programs are laid out sequentially, as in FORTRAN or FRED, one statement to a line, and they do contain statements which are purely numerical, e.g. N = N + 1 (with compulsory blanks on each side of infix operators such as + and =), but they have also some statements whose overall form is quite unfamiliar. Most of those refer to comparisons of strings and to pattern-matching on strings. In SNOBOL, the structure of string-matching statements is either

```
ABC   DEF = G
```

which means "update the value v of ABC to the string obtained by replacing the first occurrence in v of the string which is the value of DEF by the string which is the value of G", or

```
ABC   DEF                                                    (3.86)
```

which "succeeds" if the value of DEF is a string contained in v and "fails" otherwise. For example, if we have previously executed the binding statements

```
ABC = 'THE TIME IS *'
DEF = '*'
G   = 23.59
```

the value of ABC after ABC DEF = G is the string 'THE TIME IS 23.59'. In SNOBOL the single quotation marks are used to delimit strings.

The fact that (3.86) either succeeds or fails suggests that the whole structure may take on values true or false which can be used elsewhere. This idea is not used in SNOBOL; instead, one distinguishes the two values by having the option of transferring to one place in the program in the case of success and another place after failure. That is the reason for the unusual form of "GO TO" in SNOBOL. Any statement may have on its right a colon (:) followed by one of these alternatives:

(x) - go to x

S(x) - go to x if the statement succeeds, otherwise process the next statement

F(x) - go to x if the statement fails, otherwise process the next statement

S(x)F(y) - go to x if the statement succeeds, otherwise go to y.

A statement-label x begins in column 1 of a card and is separated from the rest of a statement by a blank; unlabelled statements begin to the right of column 1.

If we set out in a language such as REDUCE or New LISP to write (3.86) with transfers to D1 in the case of success or D2 in the case of failure, the best we can do is

```
IF MEMBER(DEF,ABC) THEN GO TO D1 ELSE GO TO D2;
```

while SNOBOL is satisfied with

```
ABC  DEF                    :S(D1)F(D2)          (3.87)
```

which is shorter, as befits the description of what is the most basic matching operation. Moreover, the New LISP example is wrong, because MEMBER operates on S-expressions and not strings. (Remember the contrast between Fig. 1 and Fig. 2 in Section 2.) The concise form of (3.87) follows from the requirement that a good string-processing language should express its basic operations as simply as possible.

Inside individual statements, SNOBOL allows some structures which are not paralleled in other languages. The vertical bar "|" denotes "OR", as in BNF. The expression

```
'ABC' | 'WXY' | Z                                (3.88)
```

means "the string ABC or the string WXY or the value of Z". Patterns such as (3.88) permit greater flexibility for string-matching operations which must be expressed in terms of alternatives. For example, if Z stands for '123', the match

```
XXX     'ABC' | 'WXY' | Z
```

succeeds if the value of XXX includes any of the strings ABC, WXY or 123.

The usefulness of matches can be carried one step further with the binding operator ".". If the match above succeeds, I can see which of the three alternatives has caused the success by binding it to YYY, e.g.

```
XXX  ('ABC' | 'WXY' | Z) . YYY
```

Still another step is the test for multiple parts of a pattern, e.g.

```
XXX ('ABC' | 'WXY' | Z) . YYY  ('U' | 'V') . ZZZ
```

which succeeds if the value of XXX contains a string from the first pattern followed immediately by a string from the second pattern. If this value is 'UVWXYU', then the match succeeds with YYY = 'WXY' and ZZZ = 'U'. If the value is 'UVWXYZ', the match fails and in particular nothing is bound to YYY. However, if we want the results of partial successes like the match to 'WXY', we can replace the "conditional" binding operator, the period (which works only upon total success of a matching statement), by the unconditional binding operator "$".

With these descriptions, it is possible to understand the broad outline of SNOBOL programs. In (a) to (f) below, I introduce the SNOBOL answers to other demands which can reasonably be made on a general string-processing language.

(a) Input and output

Any naming of the atom INPUT causes a card image to be read from the input medium, and the value of INPUT is the resulting string (80 characters, including blanks, from an IBM card). Thus x = INPUT sets x to the character-string from the first available card image in input.

A call of the form OUTPUT = x prints x or transmits it to the standard output file, while PUNCH = x punches it on cards.

(b) Concatenation

An analogue of CONS from list-processing languages is the concatenation of strings. In SNOBOL, it is sufficient to write n strings (n > 1) or their names, one after the other, separated by blanks, to cause concatenation. If X = 'DEF', then

```
W = 'ABC' X 'GHJ'
```

binds to W the single string 'ABCDEFGHJ'. The three originals of the strings on the right-hand side of the binding operator are unchanged.

Here is the place for another important comparison between string-processing and symbol-manipulation. Following concatenation of the symbols AB and CD into the S-expression (AB . CD), CAR and CDR in LISP or HD and TL in POP-2 can be used to recover the original components. In string-processing, on the other hand, the concatenation gives 'ABCD', which carries within it no marker to indicate where the join has been made. Thus there is no unique operation in SNOBOL corresponding to CAR or CDR.

(c) Functions to describe special patterns

A pattern such as (3.88) is a static pattern, analogous to a "static list". Frequently, though, we need to make "dynamic" matches to patterns whose

form we can only describe by giving a rule, e. g. "any pattern containing only decimal digits", or "all characters up to the next =". Several functions and patterns with standard names are available for the dynamic construction of patterns:

NULL	- the null string, a string of length zero
ANY(x)	- a pattern matching any single character in the string x
NOTANY(x)	- the complement of ANY(x)
SPAN(x)	- a pattern matching any string which contains only the characters given in x
BREAK(x)	- a pattern matching any string up to, but not including, the first occurrence of a character given in x
LEN(n)	- a pattern matching any string of n characters
POS(n)	- behaves like a predicate function which has the value true if we have currently just finished looking at the n^{th} character of a string, and false otherwise, i.e. it is a pattern which "matches" under the true condition, and fails otherwise
BAL	- a pattern which matches any string that contains balanced parentheses — hence useful for checking LISP programs!

(d) Conventional functions with values

The largest group of functions with analogues in languages of other types contains the functions for logical comparisons, e. g. EQ(x, y), which tests for equality of two numbers. The other five names NE, GE, GT, LE and LT have the same meanings as in FORTRAN. If x and y are not numbers, but strings, the correct test for equality is not with EQ but with IDENT(x, y). DIFFER(x, y) tests for inequality. Another amusing function which comes into its own for string-processing is LGT(x, y), which is true if x follows y in dictionary order ("x is lexically greater than y"), and false otherwise. Use of the logical test-functions is exemplified by a piece of SNOBOL code to ensure that two strings named by A and B are concatenated in alphabetical order and bound to C:

```
        C = LGT(A, B) B A                    :S(NEXT)
        C = A B
NEXT ....
```

Apart from the logical functions, there are three useful functions with accessible values. TRIM(x) has as its value the string less any trailing blank characters. SIZE(x) gives the length of the string x. DUPL(x, n) gives an n-character string made out of n occurrences of the single-character string x.

(e) Definition of new data-types

SNOBOL has an analogue of the POP-2 function RECORDFNS, which means that (although it is primarily a string-processing language) SNOBOL can also indulge in such things as list-processing. The function which declares new types is DATA, in the context

DATA('name(field1, field2,....)')

somewhat similar to RECORDFNS, except that there is no need to declare anything about the sizes of fields. I can, for example, allow LISP lists inside SNOBOL with the declaration of DATA('LISPWORD(CAR, CDR)'), and later construct the equivalent of (A . B), bound to C, by the statement C = LISPWORD('A', 'B'). The functions CAR and CDR can then be used as in LISP, as selectors, but also like updaters in the sense of POP-2, e.g. CAR(C) = 'X' changes the value of C to ('X' . 'B').

To test for types, the function DATATYPE acts like DATAWORD in POP-2. Thus the value of DATATYPE(C) in SNOBOL is LISPWORD.

(f) Some conclusions

For any process involving editing of text or data, or translation from one language to another, a string-processing language which operates directly on sequences of characters is the most convenient tool available. When a language like SNOBOL is on its own home-ground in such processes, it provides the means of writing programs which are shorter and usually easier to understand than programs in rival languages designed primarily for symbolic or numerical computations. Because I cannot illustrate that point here except by contrasting a short SNOBOL program with a non-short program in another language, I prefer to leave it until Section 6.4, which contains a discussion of a problem which is ideally suited to string-processing.

On the subject of translation from one "language" to another, it sometimes happens that there is a system of programs in some language L which requires of the user an ungainly type of input with which he is not at ease. For example, we may consider the case of a physicist forced to communicate with the system REDUCE-2 (Ref. [A17]) in LISP S-expressions with prefix notation and many brackets. The designer of the system then usually comes to the rescue with a specification of a "language" which the user can handle easily, plus an extra set of operations which translates commands automatically into a form acceptable to L. In REDUCE-2, the functions to translate between REDUCE and LISP are presently in LISP, although they are equally easily (perhaps more easily) written in SNOBOL. Closer to home (i.e. Austin, Texas), the system TRIGMAN for symbolic computations in celestial mechanics (Ref. [C12]) now has a high-level "language" for the user and a SNOBOL translator to bring his commands into a language which the system can accept. Further work of this type is going on in Texas.

3.3.1. CONVERT

The basic version of the pattern-matching language CONVERT (Ref. [A7]) is operated under the control of an interpreter in LISP, although there is

no reason why CONVERT should be impossible to detach from its original foundation and provide with an interpreter or compiler in another suitable language, including machine code. For examples of the exact way in which CONVERT programs are put into the computer as arguments of the LISP function CONVERT, Ref. [A7] is the best source. Here, I shall confine my attention to some of the constructions which play the same parts in CONVERT as either patterns or functions in SNOBOL. CONVERT offers a wider range of options than basic SNOBOL.

First, some basic pattern-names:

=SAME= - a structure which matches whatever structure e (not necessarily only a string) is currently under examination

== - a pattern which matches any expression

=ATO= - a pattern matching any atom in the LISP sense

=NUM= - a pattern matching any numeral

=== - the "indefinite fragment" which matches any fragment. (This is not the same as ==, as can be seen from (=== ==) which matches any list containing at least one element.)

(=AND= P1 P2 ... Pn) - all patterns P1 ... Pn must match e

(=OR= P1 P2 ... Pn) - at least one of P1 ... Pn must match e

Next, rather than the simplistic approach of SNOBOL where there is only one "type" of match, CONVERT works, on a LISP base, as if it were under the control of a LISP function (RESEMBLE p h e), where h is a list of descriptions defining how p must "resemble" e in order for a match to be successful. The quantity h is a list of 3n elements, where the middle element of each triple is a name for the mode of resemblance. Examples of triples which are understandable without further discussion of the internal workings of CONVERT are:

(x) REP (p n) - x stands for a fragment in which p is repeated n times, e.g. (== 3) for a list of 3 elements

(x) CUV p - successful if p occurs in e, and also associates the number of such occurrences with x

(x) UAR p - x matches in e whatever fragment is larger than p

(x) BUV p - only fragments in e matching p will be accepted and associated with x for later reference

Finally, some functions:

(=PRNT= x) - the same as OUTPUT = x in SNOBOL

(=RAND= x y) - causes a random choice to be made between x and y

(=COMP= x y) - set-theoretical difference of x and y, e.g. if x = (A B C) and y = (B C D), the result is (A)

(=UNON= x) - where x is a list of two lists, =UNON= treats these like the two arguments of the LISP function UNION in Section 3.2(e)

(=ARRY= m n r) - creates an array of n elements, whose m^{th} element is given by the rule r, which of course may be a function of m

(=ENTR= x m a) - sets the array-element a[m] to x

(=EXTR= m a) - has a[m] as its value

For an indication of how an actual CONVERT program is written, see Ref. [A7] for symbolic differentiation. The main purpose of the discussion in this section is to show how one can approach some basic SNOBOL-like operations in a manner different from SNOBOL, and occasionally extend their scope.

3.4. PL/I

In Section 1, I defined as "type 4" the languages which are actually designed to offer equally usable facilities for all of the principal operations which we have seen in the first three classes. The object of the present short section is simply to comment on what PL/I provides in facilities for numerical processing, symbol-manipulation and list-processing, and string-manipulation. When it was first devised, PL/I was promoted in circles not unrelated to IBM as a successor to FORTRAN, with added features to increase the range of problems to which it could be applied, but it does not seem to have replaced FORTRAN as a language for use in scientific computing centres. Perhaps this is because numerical programmers are pleased with FORTRAN, symbolic programmers are pleased with Old or New LISP, and string-manipulators are pleased with SNOBOL. For what it is worth, I must add that I have been told that one of the largest European exclusive users of PL/I is (was?) the Rolls-Royce company. A rather interesting recent scientific application of PL/I by Drouffe [C13] is in the automatic writing of FORTRAN programs for phenomenological analysis of data from experimental high-energy physics. Even in that case, though, one wonders why the PL/I programs were not used for automatic writing of PL/I programs instead. In the absence of "pure PL/I" applications, I shall not feel the need for an exhaustive description of the language. What I give below is a quick summary of the special properties of PL/I in each of the three categories that I have defined already.

(a) Numerical processing

For numerical processing, the binding operations of FORTRAN are retained, but the structure of programs and the form of conditional statements are borrowed from ALGOL. Each part of the program is a PROCEDURE. By tradition, a program begins with PROCEDURE OPTIONS(MAIN); everything else, including definitions of other named procedures corresponding to functions and subroutines, occurs inside

the scope of the main procedure. The characteristics of a PL/I program in its basic statements are well shown in the example below, which is a laborious operation, to read in 5 floating-point numbers which are each equal to either an integer or an integer plus 1/2, calculate their factorials, and print out the answers.

```
PROCEDURE OPTIONS(MAIN);
  DECLARE ANS(5) FLOAT, VALS(5) FLOAT, PARKF FIXED,
     (PARK, PARK2) FLOAT, SPI FLOAT INITIAL(1.77245385);
FAC: PROCEDURE(N) RETURNS (FLOAT);
          DECLARE (N,L) FLOAT, M FLOAT INITIAL(1.0);
          L = N;
INIT: IF L <= 0.0 THEN RETURN(M);
          M = L*M; L = L - 1.0; GO TO INIT;
     END FAC;
     GET LIST (VARS);
D2:  DO J = 1 BY 1 TO 5;
          PARK2 = VALS(J); PARKF = PARK2; PARK = PARKF;
                           ANS(J) = FAC(PARK2);
          IF PARK ¬= PARK2 THEN ANS(J) = ANS(J)*SPI;
     END D2;
     PUT LIST (ANS);
END;
```

As you can see in the example, the parentage of each step can be assigned to either FORTRAN or ALGOL, except that the argument of DO seems to come from FRED (a temporal illusion, I think), and that the PL/I principle is to use the DECLARE statement to declare the type of everything in sight. The range of attributes which is available as part of any declaration is very wide, including BINARY, BIT, CHARACTER, REAL, COMPLEX, FIXED, FLOAT, POINTER, PICTURE, and initial-value via INITIAL. Most of these names are self-explanatory, but we shall meet CHARACTER and POINTER again below. There is a larger example of a PL/I program in Section 6.4.

(b) String processing

The method of writing a character-string into a PL/I program is exactly the same as the method in SNOBOL, i.e.

```
A = 'STRING'
```

has the same meaning in both languages. In PL/I, the only extra precaution is that A should be declared as a variable of the appropriate type. If it is known that A will stand for a character-string of exactly n characters, the

relevant argument for DECLARE is A CHARACTER(n). Otherwise one guesses at a maximum length m that the string will attain, and declares A CHARACTER (m) VARYING.

The character-handling functions do not have the variety of SNOBOL, but in principle one can write procedures around them to define any function or matching operation which exists in SNOBOL. The functions are LENGTH(s), whose value is the number of characters in s, SUBSTR(s,n,m), whose value is the substring of length n + m - 1 which is found in s, beginning at the n^{th} character of s, and INDEX(s,c), whose value is n if the first occurrence of the character c in s is character n of c or 0 if c does not occur in s. In all cases, an error will occur if s is not of CHARACTER type.

Concatenation of strings in PL/I requires the strings or their names to be associated through a double vertical bar, e.g. if X has the value 'DEF', then the value of

'ABC' | | X | | 'GHJ'

is the string 'ABCDEFGHJ'.

(c) Structures, including list structures

A "structure" in PL/I is the same type of thing as a record in POP-2 or a block in L^6, although its declaration contains some useful new features. A structure, for example, can have its attributes organized on as many different hierarchical levels as the programmer wants, with the generic name for the structure on level 1. One structure in which there are four levels may turn up in banks which use PL/I programs, as

```
DECLARE 1 CUSTOMER  BASED (P),
          2 NAME,
            3 SURNAME CHARACTER (25) VARYING,
            3 FIRSTNAME CHARACTER (20) VARYING,
          2 ACCOUNT,
            3 CHEQUE,
              4 BALANCE FLOAT,
            3 SAVINGS,
              4 BALANCE FLOAT;                    (3.89)
```

where P now represents a pointer to the structure which can be used for operations like the linking of structures into lists when such a block is created later. (3.89) serves as a template for creation of one such structure every time that

ALLOCATE CUSTOMER;

is called, in exact parallel with the methods in POP-2 and SNOBOL. When the structure is available, individual fields are addressed in a new but quite logical notation, e.g.

```
CUSTOMER.ACCOUNT.SAVINGS.BALANCE                    (3.90)
```

for the savings-account balance. Constructions such as (3.90) can be used both as selectors in the POP-2 sense, to return values of components, or as updaters to attach new values to components.

To create a LISP-like word for symbol-manipulation, we can get away with much less than the complexity of (3.89). In fact, a sufficient treatment is

```
DECLARE 1 LISPWORD BASED (P),
          2 CAR CHARACTER (25) VARYING,
          2 CDR POINTER;
```

Then, suppose that we have a list structure to which LISTHEAD (another variable of type POINTER) points, and we wish to join a new element X to the left-hand end as we may do with CONS in LISP. The CONS operation can be written[2]

```
CONS: PROCEDURE(X, LISTHEAD) RETURNS (POINTER);
        DECLARE X CHARACTER (25) VARYING, P POINTER;
        ALLOCATE LISPWORD;
        LISPWORD.CAR = X; LISPWORD.CDR = LISTHEAD;
        LISTHEAD = P; RETURN(P);
    END CONS;
```

To set U and V to CAR and CDR of the structure, we write

```
U = LISTHEAD -> CAR;
V = LISTHEAD -> CDR;
```

where the sequence x -> y indicates "the y component of the structure pointed to by x".

From such modest beginnings, one can hope to build up a list-processing system in PL/I. Unfortunately this is very much of a do-it-yourself job, because PL/I does not maintain such automatic aids to list-processing as a push-down list. Not only must we write such simple functions as CONS if we wish to imitate LISP in PL/I, but we must also write a complete LISP interpreter. This may be difficult, but it is not impossible, i.e. the claim that list-processing systems based on PL/I and of interest to physicists can be written is correct. At least one version of the somewhat controversial algebraic system FORMAC (see Refs [B3, C14]) is supported by a PL/I foundation.

---

[2] This is not the most efficient way of writing CONS, but it serves the purpose of description, and does not require discussion of additional features of PL/I.

3.5. SIMULA 67

"Simulation" usually means "numerical simulation of a complex process which can be described in terms of the interaction of simpler processes". It is possible to reduce problems of this type to FORTRAN programming problems if one perseveres for long enough, but there are particular characteristics of simulations (e. g. modelling of the performance of tasks in parallel, or interrupts that switch from one process to another and back again) which are important enough to be described with special terminology. In such cases, where a type of computation with many specialized concepts sits on top of a general foundation, there is a tendency to build specialized languages to cope with the problems. Simulation is one case of this kind. For sufficiently esoteric applications, a casual user can lose sight of the foundations because of the peculiar appearance of the language which may be offered to him. In order to describe simulations concisely, and at the same time to make available all the standard capabilities of numerical computation (which is the parent of simulation), a good language for simulation should contain a good and clear numerical language L as a subset and define its own special extensions in terms of the syntax of L. My personal choice of the most pleasing language for simulation is SIMULA 67, which contains (most of) ALGOL 60 as a subset. One of the additional minor reasons for the choice, but one which is worth stating, is that SIMULA 67 has available (for programmers who wish to construct working models of the real world) the widest range of procedures for generating random numbers with different distributions that I have encountered in any language. People with a taste to learn more about other languages for simulation, and about what facilities they have in common, may find many answers in Ref. [C15].

In contrast to FORTRAN, where a small number of functions or subroutines, each with a distinct form and purpose, interact in a fixed way with a main program and are not ordinarily required to have much communication with each other, simulations handle classes of "subroutines" whose many members may share a given overall form and interact frequently with each other or with subroutines in other classes. For example, consider a network of 200 small shops, each one belonging to the class "small shop" but having differences in details from the other members, interacting with a central agency for distribution of some product and also interacting with each other for information on things like especially successful sales campaigns in one store or news of the local effects of sales at large discounts. This description implies two differences from standard FORTRAN or ALGOL programming. Firstly, we need the labour-saving ability to define the essential features of a class, so that we can later call up the definition automatically to produce a new member of the class. (This is rather like the ability to define a new data-type, as in POP-2, SNOBOL and PL/I, along with a function for creating examples of the data-type, but now we must consider the extension that the new data-types are functions or subroutines instead of being static structures.) Secondly, because the subroutines interact with each other in a complex way, e. g. by forming queues to compete for some resource, by suspending their computation at some time and resuming it at a later time, or otherwise behaving like real-life objects, sequential storage alone is not enough: the simulator must use list structures and complicated systems of pointers to take

account of the actual behaviour of the system. In this respect, we have found the key difference between languages for simulation and pure numerical languages.

SIMULA 67 meets the first of the two conditions by introducing the new concept of class to its ALGOL base. A class is actually an ALGOL block structure preceded by the declaration class. Where actions are specified inside the block, it looks like an ALGOL function-definition, but it is also possible merely to define variables, data-types, and facilities (e. g. lists and a list-processing facility, amongst other things, if a declaration begins with SIMSET class...). In both cases, there are obvious advantages in the further SIMULA 67 facility for definition of hierarchies of classes, e. g.

```
      class K1 ..... ;
K1 class K2 ..... ;
K2 class K3 ..... ;                    (3. 91)
```

One property of (3. 91) is unusual. In one sense, K3 is a subclass of K2, which is a subclass of K1, but in another sense K1 is a subclass of K2, which is a subclass of K3. The first sense refers to modifiers of the name K1, e. g. if K1 = "bottle", K2 = "whisky bottle", and K3 = "enormous whisky bottle". The second sense is that all the variables and facilities defined as part of K1 automatically become part of K2 after the declaration K1 class K2 ..., and then all these properties from both K1 and K2 are available to K3 after K2 class K3 ... . I can make a concrete example by supposing (as is possible in principle) that SIMULA 67 is used to simulate the operation of FRED. If I write such a simulation, then the class K1 may contain a complete specification of the LISP interpreter and LISP, K2 a specification of the LISP program which is the interpreter for FRED, and K3 a specification of FRED itself. (In fact, there is a legal connection between K3 and K2 (or K1): FRED calls LISP through a command LISP which I have not mentioned previously.)

A further use of (3. 91) comes naturally to hand if one's problem for simulation can be defined with only a small number of concepts in a class Kn, where these concepts are too general to connect with existing SIMULA 67 operations in a class K1. Then, if the problem is highly structured, one writes the concepts in terms of slightly less general concepts in K class n-1, connects this class with n-2 ... and so on, down to a class K2 which is simple enough to connect smoothly with K1.

Once a class K is defined with general arguments labelled in the definition by, say, X1, X2 ..., a new member of K is created (analogously to ALLOCATE K in PL/I or $K(x_1, x_2 \ldots)$ in SNOBOL) by a call

$$\text{new } K(x_1, x_2 \ldots)$$

The second condition above is met, whenever I begin a simulation which would merely look like

```
SIMULATION begin ....
```

if I happened to be simulating a FORTRAN-like process, if I write

SIMSET class SIMULATION begin ....

because SIMSET contains the facilities for handling the lists which simulate various types of queue in more complex processes. The lists in SIMSET are "doubly-linked lists" (see Section 6.3), in which there is a pointer from element x to element y for every pointer from y to x. The reason for this duality is that there are (in addition to the single-ended queue which is modelled by a LISP list) many situations for simulation where additions to a list are at one end while deletions are at the other (e.g. a well-behaved queue at a bus stop), or where additions and deletions go on at both ends (e.g. the 08.30 queue in Trieste, Piazza Oberdan, for the bus to I.C.T.P. and Miramare; deletions at the tail of the queue occur whenever the conductor announces a change of mind about where the bus will stop). Reference [B4] discusses queues and their definitions and uses in detail.

The chief automatic use which SIMULA 67 makes of its queues in SIMSET is to imitate parallel processing of blocks. In actual situations which are suitable for simulation, quite often we find that several processes are running at once, while the simulation on a computer can only handle one thing at a time. SIMULA 67 uses the queues for the book-keeping which is necessary to convert parallel processing to an equivalent form of serial processing, after SIMSET has also invented a variable which imitates the behaviour of real time.

We can expect the insides of procedures which are members of any class in SIMULA 67 to look like ALGOL procedures for the most part, as most simulations are closely related to numerical processing, but also to contain extra features which refer specifically to the behaviour of simulations. A selection of such features, whose meanings in broad outline can be guessed from their names, is: activate, after, before, at, in, delay, inspect, is, prior, reactivate, and when.

In this brief summary, it is not possible to show any convincing examples of programs for simulations, because the interesting programs tend to be rather long. I have used SIMULA 67 here in the manner of a text for a sermon, to comment on the special properties of simulations. More information can be obtained from Refs [A20, C15].

3.6. RFMS

The general characteristics of systems for management of large data-bases are discussed in Section 2 and illustrated somewhat confusingly in Fig. 3. However, all one needs to know about the data-base is a list of the types of questions which one can ask of it and the types of additions and changes which one can make to the data inside it. RFMS is an example of a system in which it is easy for the scientific user to communicate with his data. Besides the use of RFMS to maintain a catalogue of galaxies which is mentioned in Section 2, it is not hard to think of many potential applications which may treat the RFMS data-base and language as a scientific information system. At present, such systems require a very large external storage and their use is expensive, but as our experience with data-bases increases and computer technology develops, it is reasonable to hope that more people working in physics will have access to them.

I include comments about RFMS here for completeness of the survey, because RFMS (as a representative of several other systems) presents an appearance to the user which is rather different from anything that I have reviewed so far.

RFMS consists of four modules: DEFINE, LOADER, RETRIEVAL and UPDATE, whose names describe their purposes.

DEFINE defines the structure of a data-base and gives it a name. The form of the definition bears an extremely close resemblance to the definition of hierarchical structures with DECLARE in PL/I (Section 3.4(c)). A typical declaration may be

```
DEFINE;
NEW DATA BASE PERSON;
1) NAME (NAME WITH 50 PERCENT PADDING)
2) BIRTHDATE (DATE)
3) AGE (INTEGER NUMBER)
4) SEX (NAME)
5) BANK BALANCE (DECIMAL NUMBER)
6) OVERDRAFTS (DECIMAL NUMBER WITH MANY FUTURE ADDITIONS)
7) GENERAL COMMENTS (TEXT)
MAP;                                                    (3.92)
```

The set (3.92) declares the form of a new data-base and associates it with the type PERSON as soon as MAP is called. The layout in (3.92) is easy to understand and very easy to write. I have included a fairly wide range of RFMS descriptors along with the components. The only ones which require explanation are the PADDING and ADDITIONS options, which are used to reserve extra storage space for new material corresponding to the component which they describe. PADDING and ADDITIONS mean roughly the same thing; MANY translates to 60%.

The LOADER, which is called immediately afterwards, is used to enter data in the form specified within DEFINE, e.g.

```
LOAD;
ENTRY TERMINATOR IS FAREWELL;
1) NURKE, J. FRED          2) 10/10/1582
3) 388                     4) MALE
5) 1531.78                 6) 0.00
7) BAD CREDIT RISK - LIKELY TO DIE AT ANY TIME
   FAREWELL
SAVE DATA BASE ON x;                                    (3.93)
```

Again, the example is almost completely self-descriptive. The name x is the name of the file on which the information is to be saved. I have

chosen the elderly gentleman in (3.93) to illustrate something which is actually an error as I have written it — RFMS refuses to accept dates earlier than 15 October 1582!

The RETRIEVAL stage allows us to ask questions about the data-base. The standard command is

PRINT x BY LOW y WHERE z; (3.94)

The BY and WHERE options may be omitted if desired. If there is any natural ordering possible on the component y, the inclusion of BY LOW y produces that ordering of the output. The first quantity x stands for either a single component of the data-base or a list of components. Each component of number n may be referred to by its name or by Cn, e.g. PRINT NAME and PRINT C1 have the same effects here. Also in x (and in z), the name or Cn label may be preceded by COUNT, SUM, AVERAGE, MAX, MIN or SIGMA (standard deviation) if the component is a number, with the obvious results. The WHERE clause may include a single condition z or a sequence of conditions separated by AND or OR or NOT. The action to the left of WHERE is then carried out under the control of the condition(s) z. In individual conditions, we are permitted to make tests on any components of the data-base with the six relational operators (e.g. EQ, GT) whose names are the same as in FORTRAN. Other modifiers for use in tests include EXISTS, HAS and AT.

The UPDATE module accepts individual statements of the form (3.94), but here PRINT can be replaced by CHANGE, REMOVE, ADD, and ASSIGN or by certain other operators like INSERT under suitable conditions. Modification of the data-base therefore has great generality.

At present, there are not many languages for management of data-bases which are widely available for use outside their home institutions. Hence it is unlikely that you can obtain immediate experience of such languages in the same way that you can expect to seek out experience in the languages of the other types. For that reason, I have not gone into more details. However, there is not much scope for improvisation in data-base languages; I predict that the standard commands in future will continue to resemble the RFMS commands which I have introduced here.

4. POINTS OF COMPARISON WITH ANALOGUES IN FORTRAN

4.1. Input

One important distinction to make in a survey is between languages (like FORTRAN) with requirements that the programmer use FORMAT statements to give structure of a highly specific kind to his data, and free-format languages (e.g. SNOBOL, APL, FRED) with input commands but no formats at all. In an intermediate category there are languages where the rules for writing data are specific but unchangeable (S-expressions in LISP), but where the free-format situation still exists. Free format is good for convenience, but format specifications are helpful to check data-types and to make some forms of output more pleasing to the eye.

PL/I is in a special category, because it allows both alternatives. The input statement GET LIST ($a_1, a_2 \ldots$) simply reads $a_1, a_2 \ldots$ free of formats, but the data can be structured if the EDIT option is used. This has the form

$$\text{GET EDIT } (a_1, a_2, \ldots)\ (f_1, f_2, \ldots)$$

where f_i is essentially the format corresponding to a_i. Individual formats resemble those in FORTRAN quite closely, e. g. the equivalents of FORTRAN Ex. y, Fx. y, Ix, Ax and yX are respectively E(x, y), F(x, y), I(x), A(x) and X(y). The same alternatives are available for output, when PUT EDIT replaces PUT LIST.

4. 2. Declarations of types

ALGOL is the fashion-setting language for the subject of declarations of types. The basic numerical and Boolean types are discussed in Section 3.1.1; Ref. [A4] (ALGOL 68) gives a picture of the variety of types which is necessary for a perfectly general algorithmic language.

When FORTRAN and PL/I are also considered, ALGOL 60 occupies the middle-of-the-road position on declarations. FORTRAN does not make use of exotic types, and in most numerical calculations it is possible in any case to identify the default type of a variable by looking at the first character of its name. As befits a language for numerical analysis, FORTRAN also offers the types DOUBLE (double-precision) and COMPLEX which are not generally available in ALGOL systems. PL/I, in its eagerness to declare everything, covers the much wider range of types (including CHARACTER for string-processing and POINTER for hardy list-processing pioneers) which is given in Section 3. 4 (a).

Next in complexity, there are languages which allow special variable-types in connection with special applications, e. g. class for a class-of-procedures in SIMULA 67 and MATRIX for matrices in REDUCE.

Finally, and most interesting, we have the languages which contain instructions which the programmer may use to define new data-types. If we define a new type t, we can later conjure up new storage for one specimen of t with each call of ALLOCATE t in PL/I or $t(x_1, x_2 \ldots)$ in SNOBOL (where t has fields $x_1, x_2 \ldots$). POP-2 additionally requires a naming function n to be associated with t, after which reference to $n(x_1, x_2, \ldots)$ as a selector picks up the necessary storage. More detailed discussion of these facilities is available in Section 3, under the headings of the languages concerned.

In effect, L^6 is also accepting a new data-type every time that a block with a new structure is defined, but that is not educational news because no explicit TYPE declaration is involved.

4. 3. Local versus global variables

Once again, ALGOL is the language-heading (Section 3. 1. 1) under which the question is first raised. Briefly, ALGOL variables in any block are global (accessible to blocks nested inside that block) unless they are "locked out" of any block by a value declaration there, while FORTRAN variables are local to their blocks (not that there is much block-structure in FORTRAN)

unless a COMMON declaration gives them permission to roam freely through different parts of a program. PL/I adopts the same convention as ALGOL, except that value is replaced by DECLARE to render a variable local to a block. This leads to the PL/I principle that a variable used in a block but not present in the block's DECLARE statement is global. Other languages with leanings towards ALGOL syntax (POP-2, REDUCE) follow the conventions of ALGOL. The same is true of SNOBOL.

For languages with a block structure, there are of course varying degrees of "globality", according to how many blocks the programmer allows to see any variable in its global role. In LISP, where there are no type-declarations and many more "blocks" (each function is a block) than in a typical ALGOL program, the question is posed in black and white: "Is a variable local to a given function or is it global to all functions?". Purely local values are bound to variables by the function SETQ, which is discussed in Section 3.2, but the different binding function CSETQ is used whenever a variable is to be given a global value. Usually, for safety, one arranges that no variable-name which will be given a global property is ever used as a local variable (some LISP interpreters are so confused by variables with both types of value that they always override the local with the global value, which can cause nasty surprises if one is accustomed to the opposite convention of ALGOL).

4.4. Precedence of operators

APL is unique among languages with its one simple rule for precedence of operators: "The argument of an operator is everything to the right of the operator up to the first unpaired right-hand bracket or the end of the line, whichever comes first". As noted in Section 3.1.3, this can lead to some unusual consequences, e.g. that A-B-C-D-E in APL gives the same value as A-B+C-D+E in FORTRAN and other languages, but to devotees of APL the rule also has its virtues.

Other languages follow the lead of FORTRAN for how to assign precedence of familiar operators in unbracketed expressions. REDUCE and POP-2 give the programmer the option of changing the rules for precedence if he so wishes, and this property is described in Sections 3.2.1 and 3.2.2. One noteworthy extra feature of REDUCE is that it allows any named prefix operator p to be used alternatively as an infix operator with the same name, by means of the declaration

```
INFIX p;
```

This helps to make REDUCE programs resemble natural language more closely, e.g. one may think of a general predicate function IS(x,y), which is true if x has the attribute y in some sense, and write a conditional statement such as

```
IF IS(X,SMALL) THEN GO TO BLOAT;
```

but if IS has been declared as an infix operator, we have the option of using the excellent self-descriptive form

```
IF X IS SMALL THEN GO TO BLOAT;
```

I have drawn on this freedom in REDUCE or New LISP examples in Sections 6.2 and 6.4.

4.5. =

Here is a pleasant subsection with no deep meaning. Its main purpose is to show how many different answers the human mind can provide to one simple question. We have seen in the preceding Sections that = can be used either as a binding operator or as a test for equality. Sometimes programming languages use it for both tasks, but most languages have a separate convention for each operation. Below, I give a list of what several languages do in each instance. You may draw your own conclusions.

LANGUAGE	BINDING	EQUALITY
FORTRAN	=	EQ.
ALGOL	:=	=
FRED, PL/I	=	=
APL	←	=
LISP	SETQ (local), CSETQ (global)	EQ, EQUAL
POP-2	->	=
REDUCE	:=, ←	=, EQ, EQUAL
SNOBOL	=, ., $	EQ, IDENT

4.6. Conditional expressions

This is another subsection in the spirit of Section 4.5. I shall first write the forms of several languages for the test $X \geq Y$, and then use the abbreviation p for this test, in substitution into each language's version of the English "if p is true, go to statement 6, otherwise go to statement 7".

(1) Test p

FORTRAN: (X.GE.Y)

REDUCE, FRED, APL: $X \geq Y$

PL/I, POP-2: $X >= Y$

LISP: (OR (GREATERP X Y) (EQUAL X Y)), also (LESSP Y X)

L^6: (X,G,Y)(X,E,Y), also (Y,L,X)

SNOBOL: GE(X,Y)

(2) Conditional expression

FORTRAN: IF p GO TO 6
7 ....

REDUCE, PL/I, ALGOL: IF p THEN GO TO SIX ELSE GO TO SEVEN

POP-2: IF p THEN GOTO SIX ELSE GOTO SEVEN CLOSE

FRED: TO STEP 6.0 IF p
TO STEP 7.0

APL: → 7-p

LISP: (COND (p (GO SIX))) (GO SEVEN)

L^6: IFANY p THEN SIX
THEN SEVEN

SNOBOL: p :S(SIX)F(SEVEN)

4.7. Functions and subroutines

The distinction between a function and a subroutine in FORTRAN is that variables whose values are to be returned are mentioned in the argument-list along with variables transmitted to the subroutine, while a call to a function f contains only input to the function, whose unique value is bound to the name f before execution of the RETURN statement in the function-definition. This distinction is not available in the simplest non-numerical languages (REDUCE, LISP, SNOBOL), where every function returns just one value. When a user wishes to return more than one value, the answer in these languages and in the simplest views of PL/I and ALGOL is to transmit from f to a surrounding or external part p of a program as many extra values as are needed by binding them inside f to global variables which are accessible inside p.

If we want to return two values for a function f in LISP or REDUCE, there is a simple trick available. If these values are x and y, f can be defined to return CONS(x, y). Then, immediately after the call to f is completed in p, and the value is bound to (say) w, we recover x and y by taking successively the CAR and CDR of w. This trick generalizes to n values.

In POP-2, a function can return n values naturally, because the standard happening is that any or all values are placed on the stack as the last step in evaluation. Then, we can pick them off the stack and bind them successively to x, y ... via -> x -> y ..... . There are already multiple-valued functions in POP-2, e.g. DEST(x), which gives both CAR(x) and CDR(x) as values, but we are also allowed to define our own n-valued functions if we wish.

4.8. Arrays

If the FORTRAN array is a standard, its most obvious failing is that it does not permit us to define arrays dynamically, i.e. to write DIMENSION A(N), where N is either read in or calculated at run-time. Fortunately the only other language in our present collection which shares this failing is classical LISP, and very few LISP programmers feel a need to use the language's ARRAY feature anyway. Any good language can be handled by a system in which dynamical declarations of arrays are allowed (ALGOL, POP-2, REDUCE, SNOBOL, PL/I).

FRED goes one step further in abolishing array-declarations altogether, for reasons which are explained in Sections 3.1.2 and 6.3 and illustrated in Section 6.2.

FORTRAN, LISP, POP-2 and PL/I refer to array-elements inside ordinary brackets ( ), which means in practice that any quantity (e.g. SIN) already defined as a function cannot also refer to an array. In FRED, square brackets [ ] are used around indices of an array to relax this restriction. ALGOL and APL also use square brackets. SNOBOL is a lone eccentric in this respect: the indices of an array are surrounded by angular brackets < >.

APL deserves a complete section to itself on the subject of exotic but useful array-handling operations which are not paralleled in any other language. One can only do justice to the subject by reading a suitable reference (e.g. [A5] or [A6]). I look forward to the day when an equally versatile array-handling language designed primarily for conversational non-numerical work will be available. Language-designers may like to regard this as a challenge.

4.9. Communication with machine code

Sometimes, for the sake of efficient programming or to make changes in the standard behaviour of a system, one needs the ability to write a piece of one's program in machine code. In FORTRAN, most people are probably familiar with the freedom to write individual functions or subroutines in this way. For most other languages, sad to say, the manuals either contain no information at all on machine-code insertions or hint unhelpfully that the user has no business to be playing around at that level because the languages themselves are adequate for the computing which he wants to do.

L^6 gives the programmer no means of breaking into the machine-code level, but (or perhaps "because") it provides instructions which come close to being in 1:1 correspondence with machine-code instructions. Nevertheless, it seems that one cannot use L^6 instructions to alter the L^6 system. The language PL360 (Ref. [C11]) may offer greater flexibility in that direction, in the part which is devoted to the processing of machine code, but it is hard to say until one has actually used PL360.

LISP has an explicit branch which reaches down into machine code and which allows machine code to be exploited at any stage of a program. The branch is a function LAP, whose first argument consists of a list of machine-code-like instructions and whose second argument consists of a list of pairs associating each identifier in the first argument with an address in storage. Any machine-code mnemonic for an instruction x may be used inside LAP if it is first associated with the corresponding operation-code which one actually finds inside any word of a stored program where x is the operation to be executed. The function OPDEFINE, which takes a list of quantities (x_i op_i) as its argument, does this job for any x_i not already known to LAP. Without further comment, I define a specimen of LAP for use in IBM 7090 LISP.

```
OPDEFINE (( (XEC 522Q8) (TRA 2Q9) ))
DEFINE ((
(INSET (LAMBDA (U V) (LAP (CONS U (LIST (LIST V))) NIL)))
```

```
(GETCEL (LAMBDA (U) (PROG2 (INSET 12321Q (PLUS U 5Q10)) (GETT) )))
        ))
LAP (( (GETT SUBR 0) (XEC EXEV) (LDQ OCTD) (TRA MKNO))
     ((OCTD . 54Q1) (EXEV . 12321Q) (MKNO . 13645Q)))
```

The purpose of INSET is to insert the value of V into the word in storage whose address is the value of U. GETCEL performs the inverse operation. GETT is a machine-coded function (a SUBR where a function coded in normal S-expressions is an EXPR) of zero arguments which GETCEL must use. Q indicates an octal number. The definitions are listed here only to give an impression of the appearance of a LAP insertion into LISP. The machine-code operations above are made into acceptable S-expressions by being surrounded by brackets.

Present versions of FRED and REDUCE have the same abilities as LISP, but only because they can call on LAP inside LISP. However, it would be nice to see genuine machine-code extensions in every general-purpose language for non-numerical processing.

5. POINTS OF COMPARISON WITHOUT OBVIOUS ANALOGUES IN FORTRAN

5.1. Construction of compound items of information from basic items

The purpose of this subsection is to comment on the handling of the various non-numerical languages of the basic operation whose name in LISP is CONS. Numerical processors cannot offer exact equivalents of this operation, if only because the existence of something analogous to CONS or concatenation implies the existence of pointers for a language to manipulate, while FORTRAN-like languages rely on sequential structures. For the sake of argument, though, one can say that the binding of a value to the array-element A[n+1] in FRED, when the previous maximum value of the index for A in a program has been n, amounts to a crude form of concatenation. Also, one can put the array-concatenating comma in APL under that heading, in the form that it is used in (3.83).

The various non-numerical concatenating operators are given below.

LANGUAGE	DATA-TYPE	OPERATOR	ARGUMENTS
LISP	symbolic	CONS (prefix)	2
POP-2	symbolic	:: (infix), CONS (prefix)	2
REDUCE	symbolic	. (infix), CONS (prefix)	2
SNOBOL	string	blank	>1
PL/I	string	||	>1

5.2. Resolution of compound items into their basic components

In the introductory parts on string-processing in Sections 2 and 3.3, it is already pointed out that there is no unique inverse of CONS in the analysis of strings. If I am given the string 'ABCD' and the information that it has just been produced by concatenation of two smaller strings, there is no way for me to identify those strings except to insist that each string carry around with it a record of its previous history. The number of legitimate pure string-processing problems where such information is needed is so small that language-designers have left such complicated options out of their systems. However, I can easily dissect a string-one character at a time, e.g. if I wish to remove and bind to B the first character of the non-null string labelled by C, I write

```
C LEN(1) . B =
```

in SNOBOL, and

```
B = SUBSTR(C, 1, 1)
C = SUBSTR(C, 2)
```

in PL/I.

For languages based on lists, the process of resolution is central and simple. If I have made the binding equivalent to z = CONS(x, y) in some such language, I recover x by a call to CAR of z (LISP, REDUCE) or HD of z (POP-2). The corresponding operations for y are CDR or TL. In L[6], if the CAR and CDR parts of a block are labelled by A and D, the operations are respectively (x, P, zA) and (y, P, zD).

5.3. Quote

In connection with the introduction to LISP, we have seen the need for a distinction between the case where a name stands for itself and the case where it stands for something else (e.g. a value). In ordinary writing, we distinguish the first case by putting quotation marks around the name. The same distinction actually exists in FORTRAN also, but it tends to lurk below the threshold of consciousness, e.g. we do not often make mental comparisons between the unquoted form ABC and the "quoted" forms 3HABC or 3LABC. The need for the distinction is clearly strongest in a language where symbols are used regularly as pieces of data as well as names. LISP solves the problem of quoting x by writing (QUOTE x) for any legal x. The answers which are offered in other cases to the question of quotation of ABC are given below.

LANGUAGE	DATA-TYPE	QUOTED FORM
LISP	string	$$$ABC$
FORTRAN	string	3HABC or 3LABC
FRED	string	"ABC"

APL, SNOBOL, PL/I	string	'ABC'
REDUCE	symbolic	'ABC'
REDUCE	string	"ABC"
CONVERT	symbolic	(*QUO*ABC)
POP-2	string	/ABC\
POP-2	symbolic inside a list bounded by [and]	[ABC]
POP-2	other symbolic	"ABC"

REDUCE needs only one quotation mark for a symbolic expression, because the interpreter automatically provides a matching quotation mark as soon as it has read exactly one S-expression.

In most symbolic languages, numbers and the symbols for truth and falsity stand for themselves, without quotation.

5.4. Originals versus copies

Whenever a function in a non-numerical language makes implicit or explicit use of the language's analogue of CONS, the effect is that its operands themselves are not changed but that pointers are stored in the new word which CONS picks up from the free-storage list and subsequently manipulated. If you wish, you can say that such operations use the originals of their operands as templates on which to model copies for later manipulation. This prevents the originals from being modified so that we can never subsequently carry out template-like operations on their initial forms. Thus the idea of calculating with copies is the standard idea in LISP, REDUCE and POP-2. The drawback of the idea is that, if we know that we want to make permanent changes to an original, it wastes space to make the changes on a copy while the original remains untouched. For people who have the courage of their convictions, therefore, LISP and REDUCE provide functions RPLACA and RPLACD which make permanent changes to the CAR and CDR partitions of LISP words. In REDUCE or New LISP, an operation to create a copy of the list y in which the first element of y is replaced by x is

CONS(x, CDR y) or x . CDR y

but the corresponding change for the original of y is

RPLACA(y, x)

The operation which creates a copy in POP-2 is

x :: TL(y)

but HD is used as an updater to alter the original of y, as in

x -> HD(y)

In string-processing languages, by contrast, we operate on originals more often than on copies, precisely because "past history" of strings is generally unimportant. A SNOBOL statement like

A 'S' = 'C' (5.80)

where A = 'SOW', produces a change in the original of A to 'COW'. If we want to leave A untouched, we must first insert a binding A = B and then do the string-matching operation on B, not A.

PL/I does not face this question for string-processing, because its standard functions (Section 3.4(b)) are so few in number. A user who builds string-processing operations as complex as (5.80) on the PL/I base must decide for himself whether he wants the operations with the simplest appearance to act on originals or copies. This is the classical libertarian approach, which has something to be said for it if one does not complain about the work involved.

5.5. Property lists and object lists

In a system which relies on list-processing, the problems (especially those which physicists tend to pose – see Ref. [C8]) which one solves often demand that the basic atoms be given all kinds of distinct properties of varying types and lengths and that any of the properties should be available immediately if one makes a suitable reference to the atom. LISP allows an easy solution from the programmer's point of view, as mentioned in Section 3.2, by having a function DEFLIST(x, ind) which stores properties that have the associated indicator "ind", and a function GET(atom, ind) which recovers a requested property of a given atom. Where are the properties stored? The neatest answer is that each atom should possess its own private list of properties, of the rough form (# ind_1 prop_1 ind_2 prop_2 ...). This is the situation for LISP and REDUCE, where DEFLIST and GET operate directly and automatically on the property lists of atoms; we must build such structures at a very early stage if we want to set up a new general list-processing system (e.g. by embedding it in PL/I, following the start provided in Section 3.4(c)). The newest REDUCE offers functions named PUTPROP and GETPROP to replace DEFLIST and GET.

How do DEFLIST and GET know where to find the entry-point # for a given atom? The system must maintain a form of storage S in which each non-numerical atom is located at a fixed and unique place where its # can be found. It is not clear from first principles how S should be organized, but the key requirement is that a function like GET should be able to find the correct entry-point for its first argument with as little searching of S as possible. Although S is a sequential array in some small LISP systems, practical experience suggests that the best form for the widest range of situations is a list of lists of atoms. (The choice raises some extremely interesting active questions in computer sciences, but this is unfortunately not the place to go into them.) Typically, as in LISP, the list is called the object list. Also, in LISP, it is the value of the atom OBLIST, and you can look at it in any LISP system to which you have access by executing

EVAL(OBLIST NIL)

It is an instructive exercise for non-specialists to try to determine what the several atoms in any sublist of the main list have in common.

5.6. Garbage collection

Repeated calls to functions like CONS can eventually exhaust the free-storage list. When this happens, a computation cannot continue unless some of the space formerly occupied by the free-storage list is filled with material (garbage) that can be disposed of to release some more free storage. A word contains garbage if one cannot reach it by following up any chain of pointers beginning from specific places p (e. g. entry-points # on the object list) which are guaranteed to hold live material. Thus, automatic garbage collection is a two-stage business. In the first stage, searches are started from all known places p to follow pointers and mark each word which is found in this way by (say) changing a specially-reserved bit B in each word from 0 to 1. After this stage, all words with the corresponding bit still equal to 0 contain garbage by definition. The second stage is a sequential search through the region which contains the words for use in free storage. For words with B = 1, B is merely reset to 0. The words with B = 0 at the time of the scan are strung together, with a pointer from each word to the next, to form a new free-storage list.

An alternative form of garbage collection is the "reference count" method, in which any word or block contains a field or partition F that records the number of pointers which point to that block from elsewhere. When a new pointer to the block is created, the contents of F are increased by 1, and they are decreased by 1 whenever a pointer is broken or changed. When F = 0, the block is returned to the free-storage list.

Garbage collection is another topic which is of great interest in computer sciences. It is treated in detail in Chapter 2 or Ref. [B4]. Here, I mention the topic only in outline to show a little more of the foundation (novel if your background is only in numerical processing) on which languages and systems for list-processing and string-processing are built.

I leave you with a paradox. Garbage collection is a functional operation like any other machine-coded functional operation. Indeed, the garbage collector in LISP is a function: RECLAIM. Functions require free-storage space in which to manipulate partial and final results of their computations. The garbage collector is called when other functions have found that this free-storage space is full. Therefore, how can the garbage collector compute? Reference [B4] contains no conclusive answers, but at least it keeps the discussion going on a suitably non-trivial level.

5.7. Structures more complicated than arrays

There are two steps in the creation of arbitrarily complicated structures. Firstly, there is the definition of the size and internal partitioning of the basic elements of the structures. Whether we make the definitions by using complicated combinations of arrays of arrays in APL, or RECORDFNS in POP-2, or the block-defining operations GT and Df in L^6, or =ENTR= in CONVERT, or DATA in SNOBOL, or DECLARE with hierarchical forms in PL/I, or DEFINE in RFMS, the consequences are much the same. In the second step, we fill arbitrary partitions or fields in each block with pointers or links to and from other

blocks. APL and CONVERT are left somewhat behind at this step, but POP-2, L^6, SNOBOL and PL/I allow us the freedom to build almost any structure that we want. Reference [B4], Chapter 2, gives some practical examples of complicated structures. The freedom is so great in any one of the last-mentioned four languages that it is difficult to guard against over-enthusiasm. The point is that we have now come a very long way from the condition of being confined to FORTRAN and operations on pure numbers. The limiting factor in the uses of computers and computational structures for partly or wholly non-numerical physical problems is not the set of FORTRAN conventions but rather the programmer's ingenuity in making use of the other languages that are presently available. (Watch out, however, for too high a connectivity in multiply-linked structures! The consequences are stated in Ref. [C16].)

6. EXAMPLES

6.1. The Kepler equation

The Kepler equation in celestial mechanics is $u - L = e \sin u$, or alternatively $u = v + L$, $v = e \sin (L + v)$. For appropriately small e and L, the solution to u can be generated by successive approximations, starting with $u = L$ or $v = 0$. To contrast a conversational with a non-conversational numerical language, and a reasonable one with a better one (for this problem anyway), I give programs in FORTRAN and FRED below. I pad out the number of lines to equal values by writing "No analogy ..." where a statement in one language does not correspond to a statement in the other.

(1) FORTRAN

```
PROGRAM KEPLER (INPUT,OUTPUT)
REAL L
1  FORMAT(2F10.5)
2  FORMAT(1X18HTHE VALUE OF U IS F10.5)
   READ1,E,L
   V=E*SIN(L)
   VTEMP=0.0
3  IF(ABS(V-VTEMP).GE.1.0E-06)GO TO 4
   VALUE=V+L
   PRINT2,VALUE
   GO TO 61
   No analogy in FORTRAN
4  VTEMP=V
   V=E*SIN(L+V)
   GO TO 3
```

```
61 CALL EXIT
   END
```

(2) FRED

```
Four lines with no analogy in FRED
1.0 DEMAND E,L
1.1 V=E*SIN(L)
1.3 VTEMP=0
1.4 DO PART 2 UNTIL ABS(V-VTEMP)<1.0E-06
    No analogy in FRED
1.7 "THE VALUE OF U IS", V+L
    No analogy in FRED
1.8 DONE
2.0 VTEMP=V
2.1 V=E*SIN(L+V)
Three lines with no analogy in FRED
```

The line-by-line differences speak for themselves.

6.2. Ramanujan's number

What is the smallest number that can be expressed as the sum of two N^{th} powers of positive integers in two different ways? Reference [C17] puts the question and gives the answer for N = 3. For N = 2 it is $65 = 1^2 + 8^2 = 4^2 + 7^2$. Suppose that we have to write a program to calculate Ramanujan's number for given N by the most obvious and inefficient method (generation of possibilities p, such that the first p which happens to be generated twice is the number). It is unsporting to use FORTRAN and put the values of p into a fixed array, because we do not know in advance how big the array must be. The best medium of storage is either a list (LISP, REDUCE) or an array which can expand indefinitely (FRED). Below are programs in all three languages, which illustrate firstly the differences in appearance between Old LISP and New LISP code for exactly the same problem, and secondly the added information that a simple program in FRED can give because it stores p-values and other things in elastic arrays.

(1) LISP, or Old LISP

```
DEFINE ((
(RAMANUJAN (LAMBDA (N) (PROG (POWERLIST NEWPOWER
                NEWINTEGER POWERSUM SUMLIST TEMP)
   (SETQ NEWINTEGER 2)  (SETQ POWERLIST (QUOTE (1)))
```

```
J  (SETQ TEMP POWERLIST) (SETQ NEWPOWER (EXPT NEWPOWER N))
H  (COND ((NULL TEMP) (GO G)))
   (SETQ POWERSUM (PLUS NEWPOWER (CAR TEMP)))
   (COND ((MEMBER POWERSUM SUMLIST) (RETURN POWERSUM))
      (T (SETQ SUMLIST (CONS POWERSUM SUMLIST))))
   (SETQ TEMP (CDR TEMP))  (GO H)
G  (SETQ POWERLIST (CONS NEWPOWER POWERLIST))
   (SETQ NEWINTEGER (ADD1 NEWINTEGER))  (GO J)  )))
                              ))
RAMANUJAN(3)
```

The variable SUMLIST is not undefined at its first call: all program-variables automatically receive the initial value NIL. The key to the use of the function is the test (MEMBER POWERSUM SUMLIST). SUMLIST, a list of indefinite length, is the medium of storage.

(2) REDUCE, or New LISP

```
INFIX MEMBER;
LISP PROCEDURE RAMANUJAN N;
   BEGIN SCALAR POWERLIST, NEWPOWER, NEWINTEGER,
                        POWERSUM, SUMLIST, TEMP;
     NEWINTEGER := 2;
     POWERLIST := '(1);
J:   TEMP := POWERLIST; NEWPOWER := NEWINTEGER↑N;
H:   IF NULL TEMP THEN GO TO G;
     POWERSUM := NEWPOWER + CAR TEMP;
     IF POWERSUM MEMBER SUMLIST THEN RETURN POWERSUM
       ELSE SUMLIST := POWERSUM . SUMLIST;
     TEMP := CDR TEMP; GO TO H;
G:   POWERLIST := NEWPOWER . POWERLIST;
     NEWINTEGER := NEWINTEGER + 1; GO TO J;
   END;
RAMANUJAN 3;
```

In REDUCE, at present the first stage is an automatic translation to LISP. The advantage of REDUCE over LISP, as this example shows, is its readability.

(3) FRED

If we use arrays rather than lists for this problem, we have an inexpensive means of recording not only the actual wanted number, but also the constituents of the sums themselves, i.e. if we find by comparison that POWER[I, J] = POWER[K, L] is the number, we also know the values of I, J, K and L at the same instant. The FRED program therefore provides all five items of information in the output.

```
1.0 NEWINTEGER = 3
1.03 POWER[1] = 1
1.1 DEMAND N
1.12 POWER[2] = 2↑N
1.15 POWER[2, 1] = POWER[2] + 1
1.2 SIGNAL = 0
2.0 POWER[NEWINTEGER] = NEWINTEGER↑N
2.1 DO PART 3 FOR J = 1 TO NEWINTEGER-1 WHILE SIGNAL = 0
2.2 TO STEP 6.0 IF SIGNAL = 1
2.3 NEWINTEGER = NEWINTEGER + 1
2.5 TO STEP 2.0
3.0 A = POWER[NEWINTEGER] + POWER[J]
3.1 DO PART 5 FOR L = K - 1 BY -1 TO 1 FOR K = NEWINTEGER-1
                                BY -1 TO 2 WHILE SIGNAL = 0
3.2 TO STEP 6.0 IF SIGNAL = 1
3.3 POWER[NEWINTEGER] = A
5.0 SIGNAL = 1 IF A = POWER[K, L]
5.1 TO STEP 6.0 IF SIGNAL = 0
5.3 J, "↑", N, "+", NEWINTEGER, "↑", N, "=", L, "↑", N, "+", K, "↑", N,
                                                        "=", A
6.0 "THAT'S ALL, FOLKS"
RUN
```

The occurrences of the transfers to the dummy step 6.0 follow from the important and useful rule of FRED: "If a TO statement transfers control out of a PART which is being executed by the DO PART statement numbered n, the control is always transferred to the statement following statement n".

6.3. Case history of arrays in FRED

The point has already been made in Section 3.1.2 that the reason for the freedom of FRED arrays from the need for declarations of size is that the FRED interpreter uses lists to store their elements. If we wish to

add a fifth element to a simple array which already has its first four elements, we merely remove the "end" marker (NIL, in LISP) in the right-hand half of the word whose left-hand half either contains or points to element 4 and replace it by a pointer to the new element. This general property has been put to good use in the FRED program in Section 6.2.

In the LISP notation for lists, a possible "value" for A if A[1] = 10 and A[2] = 20 is ((1 10) (2 20)). At the same time, we have to accommodate values for genuine scalars somewhere. It wastes effort to have two different indicators for these two values, as long as we can use one representation. This is quite practical, because if A = 5 the "value" structure of (NIL 5 (1 10) (2 20)) now holds all of the information about A unambiguously. Hence we have a system in which scalars and one-dimensional array-elements can coexist. Moreover, there is no reason to stop us treating each substructure separately in the same way: for example, if we wish to add A[2, 6] = 26 to the collection, we change the part (2 20) to read (2 20 (6 26)). At each stage we can increase the dimensionality smoothly by one unit, but what is the answer if it is required to define A[1, 3, 5] = 13.5? The substructure must look like (1 10 (3 ? (5 13.5))), where ? stands where A[1, 3] would be if we had a value for it — but this element is not yet defined. For such a situation, we replace ? by NIL, where the meaning of NIL is taken to be "corresponding element not defined". The prescription for a method of array-storage in which elements of different dimensionalities live peacefully together is now complete.

The addition of new elements is simply a process of finding the correct positions for the new elements in the correct levels of the "value" structure by comparisons of indices, followed by some straightforward internal juggling of pointers. Deletion of elements is neither more nor less difficult.

Although our choice of representation looks reasonable, it has a built-in defect which becomes evident as soon as one gets away from the programming of toy demonstration examples. Not many users want to take advantage of the freedom to mix dimensionalities. Most of them prefer to use one-dimensional arrays and to make the arrays as large as possible without causing the system to collapse for want of space in storage. If you have an array of N elements in FORTRAN, the chances are that you want to write DO loops around it. Unfortunately, reference to Fig. 5 shows that a loop in FRED such as

A[J] = A[J] + J FOR J = 1 TO N

takes a chain of J-1 CDR operations in LISP to find A[J], and a further J-1 of them to rediscover the correct place to put the updated value of A[J]. Hence the complete loop calls CDR a total of $N^2 - N$ times. For

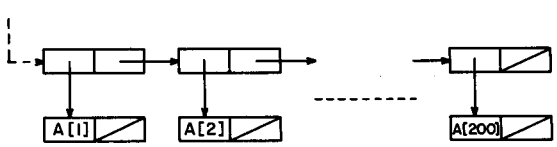


FIG. 5. Storage of a FRED array of 200 elements in a system based on conventional LISP lists.

N = 200, as in Fig. 5, this is rather expensive! When an indignant user presented us with evidence of just this example, we made a small but important change in the representation. The trouble was caused by the fact that early FRED interpreters regarded the point of entry to the array, shown by the broken-line pointer in the top left-hand corner of Fig. 5, as fixed. Our cure was to move this pointer to point to the m^{th} top-level word whenever we entered the array to find the element A[m]. The reward was that each access as part of a loop then required just one call to CDR.

The alert reader will have noticed a new disease hidden in the cure (but perhaps not as quickly as we did). As long as m is increasing, we are doing well, but what happens when (a) m is decreasing, or (b) we have finished one loop and wish to start another one at m = 1? In Fig. 5, we can only use LISP functions (CAR and CDR) to follow pointers downwards and to the right — never upwards or to the left. At this stage, if we are confined to standard LISP, we must give up — the disease is incurable unless we first kill the patient and rebuild him according to completely different rules. However, LISP users with computers in the CDC 6000 series have the ingredients for a remarkable new cure. In the 60-bit word of these machines there is enough space for three full-sized partitions where standard LISP (as in Fig. 5) has only two. In fact, if we implement standard LISP on a CDC6400 or 6600, we must choose to waste one-third of each word in free storage. The earliest builders of 6600 LISP, being economical people, chose instead to make this partition available through non-standard functions, CSR for access (c.f. CAR and CDR) and RPLACS for replacement of contents in any such partition. Our use of this gift was immediate: wherever we had a pointer out of a word w_1 to word w_2 within the top line of a standard structure like Fig. 5, we used the CSR partition of w_2 to point back to w_1. By this means, we have chosen for storage of arrays the "doubly-linked list" (see Section 3.5 and Chapter 2 of Ref. [B4]) in which it is equally easy to go backwards as to go forwards. Figure 6 illustrates the new situation. For multi-dimensional arrays, the scheme is the same; the broken-line pointer is then the means of entry from level N-1 to the level containing the N^{th} index, for every N > 1. The original disease is finally cured.

This section has the dual purposes of being a short commentary on an application of a list-processing language which shows up a limitation of the standard version of that language, and of giving a look behind the scenes at FRED to indicate some of the difficulties involved in the design of an interpreter for a conversational language.

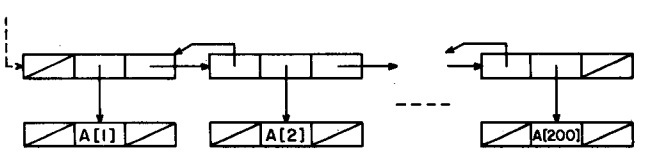


FIG. 6. Revised storage of a FRED array by means of a doubly-linked list.

6.4. Some automatic repairs to FORTRAN programs

Suppose that we are given the exercise of processing a FORTRAN card deck to detect some simple peculiarities for further attention. To make the exercise more concrete, consider the following requirements:

(a) If a card is found which is not a legal comment card ('C' or '*' in column 1, blanks in columns 2-6), or which has a statement-number that is not right-justified to column 5, or which contains a beginning of a statement which does not start with an alphabetic character in column 7, or which is not a legal continuation card, it is to be printed out together with a number which shows where it occurs in the deck.

(b) Any card which is not a comment card and which does not contain in columns 73-80 either blanks or a label x followed by a sequence of decimal digits is to be given the same treatment as in (a). This acts as a check against accidental punching of a statement past column 72. It is assumed that the label x begins in column 1 of a card which precedes the FORTRAN deck and is therefore read in before processing of the main part of the exercise begins.

(c) Let m and n be integers of less than 6 digits. If any card is found to have a statement beginning with "READ(m,n)", a card is punched on which the statement is changed to begin with "READn,".

(d) The same as in (c), except that "WRITE(m,n)" is changed into "PRINTn,".

Programs are given below in SNOBOL, REDUCE or New LISP, PL/I and FORTRAN for this exercise. It is therefore possible to get an idea of the basic operations in each language and to compare them. Not surprisingly, since this is a string-processing problem, SNOBOL wins on the grounds of compactness of its program.

(1) SNOBOL

```
        &ANCHOR = 1
        COMMA = ','
        BLANK = ' '
        IDENTIFY = 'CARD NUMBER = '
        NUMBERS = '0123456789'
        ALF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        SERIAL = TRIM(INPUT)
* NEXT, PATTERNS FOR STATEMENTS, COMMENTS AND
                                        CONTINUATIONS
        STATEMENT1 = ( SPAN(BLANK) | NULL )
                        SPAN(NUMBERS) BLANK POS(6) ANY(ALF)
        STATEMENT = DUPL(BLANK,6) ANY(ALF)
        CONTINUE = DUPL(BLANK,5) NOTANY(BLANK)
        COMMENT = ( 'C' | '*' ) DUPL(BLANK,5)
```

```
      LABEL = SERIAL SPAN(NUMBERS)
      P73.80 = TAB(72) ( LABEL | DUPL(BLANK,8) )
      CARD = 0
  START TEXT = INPUT                              :F(END)
      CARD = CARD + 1
      TEXT COMMENT                                :S(START)
      TEXT ( STATEMENT | STATEMENT1 | CONTINUE )  :F(OUT)
      TEXT  P73.80                                :S(IO)
 OUT  OUTPUT = TEXT
* NOTE HOW THE NEXT STATEMENT IS CUNNINGLY ARRANGED TO
  READ LIKE ENGLISH
      OUPUT = IDENTIFY CARD                       :(START)
IO    TEXT  ( ( LEN(6) . COL6 ) ( BREAK('(') . COMMAND ) '('
.     ( BREAK(COMMA) . DISCARD ) COMMA ( SPAN(NUMBERS). STNO )
.     ')' ) . BEGINNING                           :F(START)
      COMMAND 'READ'                              :S(PCH)
      COMMAND 'WRITE'                             :F(START)
      COMMAND = 'PRINT'
PCH   TEXT BEGINNING = COL6 COMMAND STNO COMMA
.     DUPL(BLANK,2 + SIZE(DISCARD))               :F(ERR)
      PUNCH = TEXT                                :(START)
ERR   OUTPUT = 'IMPOSSIBLE ERROR'
END
```

(2) REDUCE, or New LISP

```
INFIX MATCHES;
SERIAL := READ( );
LISP PROCEDURE REPAIR;
  BEGIN SCALAR CARD, TEXT, ENDTEXT, X, C, BEGINNING;
    X := DUPL(BLANK,5);  CARD := 0;
START: TEXT := READCARD( );
    IF NULL TEXT THEN GO TO EXIT;
    CARD := CARD + 1;
    COMMENT NOTICE THE USE OF "MATCHES" IN INFIX NOTATION
                                                NOW;
    IF TEXT MATCHES (STAR . X) OR TEXT MATCHES ('C . X) THEN
                                                GO TO START
```

```
        ELSE IF STATEMENT TEXT OR STATEMENT1 TEXT OR
                                CONTINUE TEXT THEN GO TO P7380
        ELSE GO TO OUT;

P7380: ENDTEXT := TAB(72, TEXT);
    IF LABEL ENDTEXT OR DUPL(BLANK, 8) MATCHES ENDTEXT THEN
                                                    GO TO IO;
OUT:  PRINTCARD TEXT; PRINT LIST("CARD NUMBER =", CARD);
                                                    GO TO START;
IO:  IF TAB(6, TEXT) MATCHES '(R E A D) THEN GO TO READ
        ELSE IF TAB(6, TEXT) MATCHES '(W R I T E) THEN GO TO PRINT
        ELSE GO TO START;
READ:  C := 0; BEGINNING := FIRST(10, TEXT);  GO TO BOTH;
PRINT: C := 1; BEGINNING := CONC(FIRST(6, TEXT), '(P R I N T));
BOTH: X := STNO(TAB(10 + C, TEXT), 0, 1);
    PUNCHCARD CONC(BEGINNING, CDR X, COMMA . DUPL(BLANK,
                                                    CAR X),
            TAB(69-C-CAR X-LENGTH CDR X, TEXT));  GO TO START;
EXIT:  PRINT "THAT'S ALL, FOLKS";
    END;
```

The LISP function CONC used above concatenates its arguments, lists, into one long list. A function READCARD, which has as its value a list of the 80 characters, including blanks, on an IBM card, is assumed to exist, although in practice one must write it in terms of LISP character-handling functions. The functions PRINTCARD and PUNCHCARD are also assumed.

The New LISP program looks remarkably short, but it does not provide a complete solution to the problem. This is because we must still define a large number of short subsidiary functions (including one for each of the tested patterns STATEMENT, STATEMENT1 and CONTINUE). When this is done, the length of the total program becomes alarmingly great. We now proceed with that task.

```
LISP PROCEDURE MATCHES (X, Y);
    IF NULL X THEN NIL ELSE IF NULL Y THEN T ELSE IF CAR X
            EQUAL CAR Y THEN MATCHES(CDR X, CDR Y) ELSE NIL;
LISP PROCEDURE STATEMENT U;
    BEGIN SCALAR N;  N := 6;
G:  IF N = 0 THEN RETURN LITER CAR U
        ELSE IF CAR U EQ BLANK THEN N := N-1 ELSE RETURN NIL;
    U := CDR U;  GO TO G;
  END;
```

```
LISP PROCEDURE TAB(N,U); IF N=0 THEN U ELSE TAB(N-1,CDR U);
LISP PROCEDURE DUPL(CHAR,NUMBER);
  IF NUMBER=0 THEN NIL ELSE CHAR . DUPL(CHAR,NUMBER-1);
LISP PROCEDURE FIRST (N,U);
  IF N=0 THEN NIL ELSE CAR U . FIRST(N-1,CDR U);
LISP PROCEDURE STATEMENT1 U;
  BEGIN SCALAR P,Q; P := 0;
G: IF Q := DIGIT CAR U THEN P := P+1
    ELSE IF NULL Q AND CAR U EQ BLANK THEN P := P+1
    ELSE IF CAR U EQ BLANK THEN RETURN (P=5 AND LITER CADR U)
    ELSE IF P > 5 THEN RETURN NIL;
  U := CDR U; GO TO G;
 END;
LISP PROCEDURE CONTINUE U;
  BEGIN SCALAR N; N := 5;
G: IF N=0 THEN RETURN NOT (CAR U EQ BLANK)
    ELSE IF CAR U EQ BLANK THEN N := N-1 ELSE RETURN NIL;
  U := CDR U; GO TO G;
 END;
LIST PROCEDURE LABEL U;
 BEGIN
    IF U MATCHES SERIAL THEN GO TO G ELSE RETURN NIL;
G:  U := TAB(LENGTH SERIAL,U);
H:  IF NULL U THEN RETURN T ELSE IF NOT DIGIT CAR U THEN
                                                RETURN NIL;
    U := CDR U; GO TO H;
 END;
LISP PROCEDURE STNO (U,N,M);
    IF NUMBERP M THEN STNO(CDR U, N+1, IF CAR U EQ COMMA
                                        THEN NIL ELSE M)
       ELSE IF CAR U EQ ")" THEN (N-LENGTH M).REVERSE M
       ELSE STNO(CDR U, N+1, CAR U . M);
```

(3) PL/I

```
REPAIR: PROCEDURE OPTIONS(MAIN);
          DECLARE TEXT CHARACTER (80), LSERIAL FIXED,
```

```
            BLANK CHARACTER (1) INITIAL (' '),
            ALF CHARACTER (26) INITIAL
                              ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
            COMMA CHARACTER (1) INITIAL (','),
            NUMBERS CHARACTER (10) INITIAL ('0123456789'),
            CARD FIXED INITIAL (0),
            SERIAL CHARACTER (7) VARYING,
            PLACE CHARACTER (24) VARYING,
            (A, B, C, EIGHTS) FIXED;
        DECLARE PUNCH FILE OUTPUT;
        ON ENDFILE (SYSIN) GO TO EXIT;
SPAN: PROCEDURE (TEST, FORM) RETURNS (FIXED);
        DECLARE TEST CHARACTER (8) VARYING, FORM
                              CHARACTER (26) VARYING,
                              COUNT FIXED INITIAL (0);
G:      IF INDEX(TEST, FORM)¬= 1 THEN RETURN (COUNT);
        COUNT = COUNT + 1;
        IF LENGTH(TEST) = 1 THEN RETURN (COUNT);
        TEST = SUBSTR(TEST, 2);
        GO TO G;
    END SPAN;
DUPL: PROCEDURE (CHAR, NUMBER) RETURNS (CHARACTER (80) VARYING);
        DECLARE CHAR CHARACTER (1), NUMBER FIXED,
          RESULT CHARACTER (80) VARYING INITIAL (' ');

H:    IF NUMBER = 0 THEN RETURN (RESULT);
      RESULT = CHAR || RESULT;
      NUMBER = NUMBER - 1;
      GO TO H;
  END DUPL;
    GET LIST (SERIAL);
    LSERIAL = LENGTH(SERIAL);
    EIGHTS = 8 - LSERIAL;
START: GET LIST (TEXT);
    CARD = CARD + 1;
/* TEST FOR COMMENT, NON-NUMBERED STATEMENT, AND
                              CONTINUATION */
```

```
      IF (SUBSTR(TEXT, 1, 1) = 'C' | SUBSTR(TEXT, 1, 1) = '*') ∧
            SPAN(SUBSTR(TEXT, 2, 5), BLANK) = 5 THEN GO TO START
         ELSE IF ((SPAN(SUBSTR(TEXT, 1, 6), BLANK) = 6) ∧
            INDEX(SUBSTR(TEXT, 7), ALF) = 1) |
              ((SPAN(SUBSTR(TEXT, 1, 5), BLANK) = 5) ∧
                SUBSTR(TEXT, 6, 1) ¬= BLANK) THEN GO TO P7380;
/* NOW FOR NUMBERED STATEMENTS - THE NEXT LINE IS TO SAVE
                                          TIME AND SPACE */
         A = SPAN(SUBSTR(TEXT, 1, 5), BLANK);
         IF ¬ ((A + SPAN(SUBSTR(TEXT, A + 1, 5 - A), NUMBERS) = 5) ∧
              SUBSTR(TEXT, 6, 1) = BLANK ∧
              INDEX(SUBSTR(TEXT, 7), ALF) = 1) THEN GO TO OUT;
P7380:   PLACE = SUBSTR(TEXT, 73);
         IF (SPAN(PLACE, BLANK) = 8) |
           (SUBSTR(PLACE, 1, LSERIAL) = SERIAL ∧
             SPAN(SUBSTR(PLACE, LSERIAL + 1, EIGHTS), NUMBERS)
                                    = EIGHTS) THEN GO TO IO;
OUT:     PUT LIST (TEXT);
         PUT LIST ('CARD NUMBER = ', CARD);
         GO TO START;
IO:      IF SUBSTR(TEXT, 7, 5) = 'READ(' THEN GO TO READ
           ELSE IF SUBSTR(TEXT, 7, 6) = 'WRITE(' THEN GO TO PRINT
           ELSE GO TO START;
READ:    C = 0;
         PLACE = SUBSTR(TEXT, 1, 10);
         GO TO BOTH;
PRINT:   C = 1;
         PLACE = SUBSTR(TEXT, 1, 6) | | 'PRINT'
BOTH:    A = INDEX(SUBSTR(TEXT, 7), COMMA);
         B = INDEX(SUBSTR(TEXT, 7), ')');
         TEXT = PLACE | | SUBSTR(TEXT, A + 1, B - A - 1) | | COMMA | |
                      DUPL(BLANK, A - C - 10) | | SUBSTR(TEXT, B + 1);
         OPEN FILE (PUNCH) OUTPUT;
         PUT FILE (PUNCH) LIST (TEXT);
         CLOSE FILE (PUNCH);
         GO TO START;
```

```
EXIT:    END REPAIR;
```

The signs ¬, | and ∧ represent respectively NOT, OR and AND in PL/I.

(4) FORTRAN (Program written by Mr. George A. Sarvey)

```
      PROGRAM RFORT(INPUT,OUTPUT)
      INTEGER COL(80), PREX, REED(5), ZERO, A, Z, WRIT(6), COMMA, S,
                                                    RPAR, ENN(3)
     1, AST, C, BLK, CARD, PREXP
      DIMENSION IMAGE(80), IDENT(8)
      DATA (REED(I), I=1, 5), ZERO, A, Z, (WRIT(I), I=1, 6), COMMA, RPAR, (ENN(I),
     1I=1,3)/1RR,1RE, 1RA,1RD,1R(,1R0,1RA, 1RZ,1RW,1RR,1RI,1RT, 1RE,1R(,1R,
     2, 1R), 1RE, 1RN, 1RD/
      DATA NINE, AST, BLK, C/1R9, 1R*, 1R , 1RC/
      CARD = 0
      READ 100, (IDENT(I), I=1, 8)
  100 FORMAT(8R1)
  300 READ 101, (COL(I), I=1, 80)
  101 FORMAT(80R1)
          CARD = CARD+1
    *       COMMENT CARD CHECK
 3005 IF(COL(1). EQ. C. OR. COL(1). EQ. AST)GO TO 300
    *       CONTINUATION-CARD CHECKS
          IF(COL(6). EQ. BLK)GO TO 302
    *       CONTINUATION-CARD LABEL CHECK
          DO 1 I = 1, 5
          IF(COL(I). NE. BLK)GO TO 301
        1 CONTINUE
          GO TO 300
  301 ASSIGN 300 TO PREX
          GO TO 400
  302 DO 2 I = 1, 5
          IF(COL(I). NE. BLK)GO TO 303
        2 CONTINUE
          GO TO 3035
```

```
 303 DO 3 J = 1, 5
     IF(COL(J). LT. ZERO. OR. COL(J). GT. NINE)GO TO 304
   3 CONTINUE
*       CHECK ALPHABETIC CHARACTER IN COLUMN 7
3035 IF(COL(7). LT. A. OR. COL(7). GT. Z)GO TO 301
     DO 4I = 1, 5
*       CHECK FOR READ(M, N)
     IF(COL(I + 6). NE. REED(I))GO TO 305
   4 CONTINUE
     N = 11
     GO TO 306
 304 ASSIGN 450 TO PREX
     GO TO 400
*       CHECK FOR WRITE(M, N)
 305 DO 5 I = 1, 6
     IF(COL(I+6). NE. WRIT(I))GO TO 500
   5 CONTINUE
     N=12
*       SHIFT MATERIAL LEFT ON CARD - NOT REQUIRED BUT NICE
 306 DO 6 I=1, 3
     IF(COL(I+N). EQ. COMMA)GO TO 307
   6 CONTINUE
     GO TO 301
 307 S=I+1
     L=N+I+1
     DO 7 K = L, 21
     IF(COL(K). EQ. RPAR)GO TO 308
   7 CONTINUE
     GO TO 301
 308 DO 8 K = L, 21
     COL(K-S)=COL(K)
     IF(COL(K). EQ. RPAR)COL(K-S)=COMMA
   8 CONTINUE
     DO 9 K=22, 72
   9 COL(K - S)=COL(K)
```

```
*       READ NEXT CARD, AND STORE IN "IMAGE"
3085  READ 102, (IMAGE(I),I=1,80)
 102  FORMAT(80R1)
*       CONTINUATION-CARD CHECK
      IF(IMAGE(6).NE. BLK)GO TO 310
      DO 10 K=1,S
  10  COL(72-S+K)=BLK
*       CALL PUNCH FOR FIRST CARD
      ASSIGN 309 TO PREXP
      GO TO 401
*       SHIFT FROM CONTINUATION CARD TO "COL"
 309  DO 11 K=1,80
  11  COL(K) = IMAGE(K)
      CARD=CARD+1
      GO TO 3005
 310  DO 12 K=1,S
  12  COL(72-S+K)=IMAGE(K+6)
      ASSIGN 311 TO PREXP
      GO TO 401
*       MOVE LABEL AND SEQUENCE OF "IMAGE" TO "COL"
 311  DO 13 K=1,8
      COL(K)=IMAGE(K)
  13  COL(72+K)=IMAGE(K+72)
*       MOVE/SHIFT CENTRE OF "IMAGE" TO "COL"
      M = 72-S
      DO 14  K=7,M
  14  COL(K)=IMAGE(K+S)
      CARD=CARD+1
      GO TO 3085
 500  IF(IDENT(1).EQ. BLK)GO TO 304
      DO 15 K=73,80
      IF((IDENT(K-72).EQ. BLK.AND.(COL(K).LT. ZERO.OR. COL(K)
                                          .GT. NINE)).OR.
     1(IDENT(K-72).NE. BLK.AND. IDENT(K-72).NE. COL(K)))GO TO 304
  15  CONTINUE
```

```
450  DO 16 K = 1, 3
     IF(COL(K+6). NE. ENN(K))GO TO 300
 16  CONTINUE
     GO TO 600
*       PRINTING OR PUNCHING OF INTERESTING CARDS
400  PRINT 200, (COL(I), I=1, 80), CARD
200  FORMAT(1H ,80R1,4X, 10HCARD NO. =, I5)
     GO TO PREX
401  PUNCH 201, (COL(I), I=1, 80)
201  FORMAT(80R1)
     GO TO PREXP
600  CONTINUE
     CALL EXIT
     END
```

6.5. Symbolic differentiation

Symbolic differentiation is one of the famous problems of symbolic computation. It is generally believed that this subject was the first to be treated by a program which was a genuine example of symbol-manipulation, in 1953 (Ref. [C18]). Simple examples are given below, in which it is assumed that the only forms allowed in input are in S-expression prefix notation, with only the operators PLUS and TIMES, and with each operator having only two arguments. Of course it is not difficult to extend programs to deal with all realistic cases, since differentiation (unlike symbolic integration) is a simple problem, but the programs here are kept short because the main object is to display the programming languages and not the algorithm. A POP-2 program for differentiation is given on page 36 of Ref. [A16], and a CONVERT program on page 613 of Ref. [A7]. The programs below are in REDUCE and SNOBOL. There is an interesting contrast: the REDUCE program treats the S-expression input as a list and handles it accordingly, but the SNOBOL program processes it as it stands, as a string including brackets.

(1) REDUCE, or New LISP

```
LISP PROCEDURE DIFF (E,X);
  IF ATOM E THEN IF E EQ X THEN 1 ELSE 0
    ELSE IF CAR E EQ 'PLUS THEN LIST('PLUS, DIFF(CADR E,X),
                                     DIFF(CADDR E,X))
    ELSE IF CAR E EQ 'TIMES THEN
        LIST('PLUS, LIST('TIMES, CADDR E, DIFF(CADR E,X)),
                    LIST('TIMES, CADR E, DIFF(CADDR E,X)))
```

```
      ELSE ERROR LIST("ILLEGAL SYNTAX IN DIFF ",E," W.R.T. ",X);
```

(2) SNOBOL

```
      &ANCHOR = 1
      ALF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      NUMBERS = '0123456789'
      SIGNS = '+-'
      LISP = ALF NUMBERS SIGNS '/.*'
      LPAR = '('
      RPAR = ')'
      BLANK = ' '
      NEXT = LPAR BLANK RPAR
      LISPATOM = BREAK(BLANK)∨ ANY(NUMBERS) ∨ ( ( ANY(ALF)
                 ∨ANY(SIGNS) ) SPAN(LISP) ) ∨ ANY(ALF)
      LBAL = ( LPAR BREAK(RPAR) RPAR ) ∨ BREAK(NEXT)
      LSPACE = SPAN(BLANK) ∨ NULL
      LISPNULL = LPAR LSPACE RPAR
      DEFINE('CAR(A)B')                              :(ENDCAR)
CAR   A ( LSPACE LPAR LSPACE ( LBAL . B ) )          :F(ERCAR)
      B = TRIM(B)
      B  LPAR                                        :S(FIN)
      B  LISPATOM . CAR                          :S(RETURN)F(ERCAR)
FIN   CAR = B                                        :(RETURN)
ERCAR OUTPUT = 'ERROR IN CAR ' A
ENDCAR
       PUTCDR = LSPACE LPAR LSPACE *B LSPACE
       DEFINE('CDR(A)B, C, D')                     :(ENDCDR)
CDR    B = CAR(A)
       C = TRIM(A)
       C   PUTCDR = LPAR                           :F(ERCDR)
       C = TRIM(C)
       C   LISPNULL                                :F(LAST)
       C = NULL
LAST   CDR = C                                      :(RETURN)
ERCDR OUTPUT = 'ERROR IN CDR ' A
```

```
ENDCDR
        DEFINE('DIFF(E,X)CARE,CDRE')                    :(ENDIFF)
DIFF    E BREAK('()')                                   :S(NOATOM)
        DIFF = IDENT(E,X) ' 1'                          :S(RETURN)
        DIFF = ' 0'                                     :(RETURN)
NOATOM CARE = CAR(E)
        CDRE = CDR(E)
        DIFF = IDENT(CARE,'PLUS') LPAR 'PLUS ' DIFF(CAR(CDRE),X)
                BLANK DIFF(CAR(CDR(CDRE)),X) RPAR       :S(RETURN)
        CARE = IDENT(CARE,'TIMES') CARE BLANK           :F(ERDIF)
        DIFF = '(PLUS (' CARE CAR(CDR(CDRE)) DIFF(CAR(CDRE),X) ') ('
                CARE CAR(CDRE) DIFF(CAR(CDR(CDRE)),X) '))' :(RETURN)
ERDIF   OUTPUT = 'ILLEGAL SYNTAX IN DIFF ' E 'W.R.T. ' X
ENDIFF
*  NOW WE READ E AND X FROM INPUT
AGAIN   E = NULL
START   Y = TRIM(INPUT)                                 :F(END)
        E = E Y
        E BAL                                           :F(START)
        X = TRIM(INPUT)
* AND NOW, AT LAST, THE DIFFERENTIATION ITSELF
        OUTPUT = DIFF(E,X)                              :(AGAIN)
END
```

The program above shows three examples of function-definitions in SNOBOL, the quantities following the bracketed argument of a function being local variables in that function. The operation *, as in *B in PUTCDR, is the SNOBOL equivalent of the LISP QUOTE for certain variables. The sign ∨ in this program is an alternative to the "OR" sign | on the printer (but the two are the same on punch and teleprinter keyboards).

6.6. Styles in languages

Where do we go from here? Some of the languages which we have seen, such as FORTRAN, are not always understandable to the beginner, because their syntax is not clear from an attempt to make analogies with natural language. These languages probably owe their uneasy mixture of English and mathematical notation to the fact that they were designed early in the history of computation, when the users' pressure for more congenial languages was not so well developed as it is today. One can distinguish two classes of users: one which scorns syntax and looks for a compressed

mathematical notation (e. g. APL), and one which grows happier as the languages (e. g. REDUCE) approach closer to everyday linguistic usage. I expect the languages for these two groups to diverge a little in styles, so that there is more APL-like notation for the APL fanciers, and more REDUCE-like notation for the natural-language fanciers. To the physicist interested in non-numerical programming, there will probably always be a case for a language rich in syntax, which may mean that future developments of special interest will resemble REDUCE, with extensions. For specific uses, e. g. manipulation of Poisson series in mechanics, there may be specific commands (e. g. SERIES, which may declare variables to stand for entire trigonometric series which can be stored in very compact data-structures if that information is known), but the commands will probably still be embedded in a language like REDUCE. The alternative to REDUCE is the full ALGOL 68, if and when the full ALGOL 68 becomes available in working form on a large computer, but the general direction of development which adds new special words to an existing syntax is clearly defined. Our present work in Texas, on a large number of special operations for part of a general "algebra machine", has tended in this direction almost without the designers' being aware of it. Development of a rival APL-like set of languages is not impossible (e. g. for operations in group theory), and there is unlikely to be any conflict, because both forms of language have their advantages on the right occasions. Finally, there will (probably) always be FORTRAN in the centre as a bridge between the two groups. Recall the comment early in Section 1: "... you can do [anything] in FORTRAN". It would be nice to regard as prophecy the words of Mr. W. Yeats ("Things fall apart; the centre cannot hold") in this respect, but the prophecy will probably take a long time to fulfil.

REFERENCES

A. Manuals and descriptions of specific languages

(The first reference listed for each language is the best introductory reference.)

(ALGOL)

[A1] BAUMANN, R., FELICIANO, M., BAUER, F.L., SAMELSON, K., Introduction to ALGOL, Prentice-Hall Inc., Englewood Cliffs, N.J. (1964).
[A2] DIJKSTRA, E.W., Primer of ALGOL 60 Programming, Academic Press, New York (1962).
[A3] NAUR, P. et al., Communs Ass. comput. Mach. 6 (1963) 1.
[A4] Van WIJNGAARDEN, A., Final Draft Report on the Algorithmic Language ALGOL 68, Springer-Verlag, Berlin (1969).

(APL)

[A5] KATZAN, H., APL Programming and Computer Techniques, Van Nostrand-Reinhold, New York (1970).
[A6] IVERSON, K., A Programming Language, John Wiley and Sons, New York (1962).

(CONVERT)

[A7] GUZMAN, A., McINTOSH, H.V., Communs Ass. comput. Mach. 9 (1966) 604.
(More articles by H.V. McIntosh on applications are to be found in Acta Mexicana de Ciencia y Tecnología.)

(FRED, also JOSS and CAL)

[A8] CAMPBELL, J.A., JEFFERYS, W.H., FRED, Texas Univ., Computation Center, Austin, Tex. (1970).
[A9] CAL Reference Manual, Com-Share Inc., Ann Arbor, Mich. (1969).

(LISP)

[A10] WEISSMAN, C., LISP 1.5 Primer, Dickenson Publishing Co., Belmont, Calif. (1967).
[A11] McCARTHY, J. et al., LISP 1.5 Programmer's Manual, MIT Press, Cambridge, Mass. (1965).

(L^6)

[A12] KNOWLTON, K.C., Communs Ass. comput. Mach. 8 (1965) 623; 9 (1966) 616.

(PL/I)

[A13] BATES, F., DOUGLAS, M.L., Programming Language/One, Prentice-Hall Inc., Englewood Cliffs, N.J. (1967).
[A14] FIKE, C.T., PL/I for Scientific Programmers, Prentice-Hall Inc., Englewood Cliffs, N.J. (1970).
[A15] POLLACK, S.V., A Guide to PL/I, Holt Rinehart and Winston Inc., New York (1969).

(POP-2)

[A16] BURSTALL, R.M., COLLINS, J.S., POPPLESTONE, R.J., Programming in POP-2, Edinburgh University Press, Edinburgh (1971).

(REDUCE)

[A17] HEARN, A.C., REDUCE-2 User's Manual, Stanford Artificial Intelligence Project Memo AIM-133, Stanford University, Calif. (1970).
[A18] HEARN, A.C., in Interactive Systems for Experimental Applied Mathematics (KLERER, M., REINFELDS, J., Eds), Academic Press, New York (1968) 79.

(RFMS)

[A19] DALE, A., Remote File Management System User's Manual, University Co-Op, Austin, Tex. (1971).

(SIMULA 67)

[A20] BIRTWISTLE, G., SIMULA BEGIN, Norsk Regnesentral, Oslo, Norway (1970).
[A21] DAHL, O.-J., MYHRHAUG, B., NYGAARD, K., SIMULA 67, Common Base Language, Norsk Regnesentral, Oslo, Norway (1970).

(SNOBOL)

[A22] GRISWOLD, R.E., POAGE, J.F., POLONSKY, I.P., The SNOBOL4 Programming Language, Prentice-Hall Inc., Englewood Cliffs, N.J. (1968).
[A23] FORTE, A., SNOBOL3 Primer, MIT Press, Cambridge, Mass. (1967).

B. General references on programming languages and algorithmic methods

[B1] BARRON, D.W., Programming Languages, Proc. 1970 CERN Computing and Data Processing School, CERN, Geneva (1971) 1.
[B2] ROSEN, S., Programming Systems and Languages, McGraw-Hill Inc., New York (1967).
[B3] SAMMET, J.E., Programming Languages - History and Fundamentals, Prentice-Hall Inc., Englewood Cliffs, N.J. (1969).
[B4] KNUTH, D.E., The Art of Computer Programming, Vol.1, Fundamental Algorithms, Addison-Wesley, Reading, Mass. (1968).
[B5] KNUTH, D.E., The Art of Computer Programming, Vol.2, Seminumerical Algorithms, Addison-Wesley, Reading, Mass. (1970).

C. Further references on types and applications of non-numerical programming

[C1] Proc. Second Symp. Symbolic and Algebraic Manipulation, Association for Computing Machinery, New York (1971).
[C2] Communs Ass. comput. Mach. 9 (1966), August edition.
[C3] Communs Ass. comput. Mach. 14 (1971), August edition.
[C4] CAMPBELL, J.A., Comput. Phys. Communs 1 (1970) 251.
[C5] BARTON, D., FITCH, J.P., Applications of algebraic manipulative programs in physics, Rep. Prog. Phys. 35 (1972) 235.
[C6] CLARK, R.B., MANNY, B.L., PARSONS, R.G., Ann. Phys. 69 (1971) 522.
[C7] AMAREL, S., "On representations of problems of reasoning about actions", Machine Intelligence 3 (MICHIE, D., Ed.), Edinburgh University Press, Edinburgh (1968) 131.
[C8] CAMPBELL, J.A., HEARN, A.C., J. comput. Phys. 5 (1970) 280.
[C9] HEARN, A.C., Bull. Am. phys. Soc. 9 (1964) 436.
[C10] HEARN, A.C., Communs Ass. comput. Mach. 9 (1966) 573.
[C11] WIRTH, N., J. Ass. comput. Mach. 15 (1968) 37.
[C12] JEFFERYS, W.H., Celestial Mech. 2 (1970) 474.
[C13] DROUFFE, J.M., "Automatic generation of programs in high energy phenomenology", Colloquium on Computational Methods in Theoretical Physics, Centre National de la Recherche Scientifique, Marseille (1970) II.22.
[C14] SCONZO, P., LESCHACK, A.R., TOBEY, R.G., Astr. J. 70 (1965) 265.
[C15] GORDON, G., System Simulation, Prentice-Hall Inc., Englewood Cliffs, N.J. (1969).
[C16] DEUTSCH, A.J., "A subway named Moebius", Fantasia Mathematica (FADIMAN, C., Ed.), Simon and Schuster, New York (1958) 222.
[C17] HARDY, G.H., A Mathematician's Apology, Cambridge University Press, Cambridge (1969) 37.
[C18] KAHRIMANIAN, H.G., Analytical Differentiation by a Digital Computer, M.A. thesis, Temple University, Philadelphia (1953).

TRANSLATION OF SYMBOLIC ALGOL I TO SYMBOLIC ALGOL II

G. KUO-PETRAVIĆ, M. PETRAVIĆ
Department of Engineering,
University of Oxford, Oxford

K.V. ROBERTS
Culham Laboratory, Abingdon, Berks,
United Kingdom

Abstract

TRANSLATION OF SYMBOLIC ALGOL I TO SYMBOLIC ALGOL II.

This paper describes the logic of the translator program used for the translation of Symbolic ALGOL I to Symbolic ALGOL II, without referring to any processing language which may be used for this purpose. The syntax analysis which involves a reduction to REVERSE POLISH, and subsequent decoding back to a set of nested procedures in Symbolic ALGOL II, are fully explained.

As has been noted in a previous paper [1], Symbolic ALGOL I has a form which is closely analogous to that of mathematical physics, and it can therefore serve as a language of communication between computational physicists. This should lead to a rapid progress in the interchange of ideas and programs, which has so far been hampered to some degree by the machine dependence of all programs and the lack of a sufficiently flexible method of communication. Symbolic ALGOL I, although legal ALGOL, uses this language in such a way that the actual physics contained in a program can be expressed in a concise, intelligible and general form, in much the same way as in mathematical physics. However, by its very generality this style of writing programs has lost a large factor in speed because of the necessity of repeated procedure calls. Therefore, in order for Symbolic ALGOL I to be a useful language for production calculations, some automatic technique of optimization must be developed. Furthermore, the method should be a relatively simple one, so that it is easy to implement on any machine with its existing compilers and peripheral software. In a series of papers [2, 3] we have set out to show how a gain in speed of a few orders of magnitude may be achieved by a two-stage process of optimization, involving first the use of the STAGE-2 Macro-Processor [4] and then ALGOL itself. In this paper we shall be concerned only with the first stage of the optimization, which is the automatic translation of Symbolic ALGOL I into a second style of ALGOL termed Symbolic ALGOL II.

The aim of the translator program should be to accept, as input, differential equations in their mathematical form or in a form which is closely analogous to it, such as Symbolic ALGOL I. It is obvious that some text manipulation and syntax analysis is required before Symbolic ALGOL II, which consists of nested procedure calls, can be produced. The ability to manipulate text is satisfied by almost any high-level language such as FORTRAN or ALGOL, but another approach is to use a string processor such as STAGE 2 [4]. In this paper we shall only describe the logic of the

process leading to the production of Symbolic ALGOL II, which could be implemented by a wide range of languages. For completeness, however, we should mention that the STAGE-2 Macro-Processor [4], as implemented on the Culham KDF9, was in fact used in the translator program. More details about the program in STAGE 2 are contained in Ref. [3].

We shall now proceed to illustrate our method by way of an example, which is the right-hand side of the magnetic equation used in the TRINITY program [1]. In Symbolic ALGOL I this equation reads:

```
B = dT * (Curl ( Cross (v, B) ) + eta * Delsq (B) );
```

The first step is to replace the operator names by symbols such as α, β, γ, etc. We note that the operators are of two kinds; the prefix operator like Curl and Delsq which operates on its right-hand side only, and the infix operator like Cross which acts between two operands. The latter is written in Symbolic ALGOL I as a procedure with two arguments. It is desirable at this stage to replace Cross (v, B) by ((v)α(B)), where α is any symbol, to bring it in line with other normal operators, e.g. */. For the sake of illustration, let us take here a more complex example:

```
- - - - * Cross ( v + Curl (B), Curl ( Cross (v, B) ) ) + - - - -
- - - - * Cross < v + Curl (B), Curl ( Cross (v, B) ) > + - - - -
- - - - *          ( (v + Curl (B) )α(Curl (Cross (v, B) ) ) ) + - - - -
```

We set up a counter which counts brackets, taking opening bracket (= +1 and closing bracket) = -1. Taking first the second half of the line we count to the first unbalanced closing bracket and replace it by > . Similarly, the first unbalanced opening bracket counting backwards in the first half of the line is replaced by <. The name Cross is coded into a symbol such as α and the following substitution made:

```
< →( (
, →) α (
> →) )
```

Returning to the case of the magnetic equation, we would then have:

```
B + dT * ( Curl ( ( ( (v) α (B) ) ) + eta * Delsq (B) )
```

Prefix operator names like Curl and Delsq are then coded. They are recognized by the fact that they appear before an opening bracket (and after an operator or another opening bracket (. We then have

```
B + dT * ( β ( ( (v) α (B) ) ) + eta * γ (B) )
```

where Curl has been coded into β and Delsq into γ. In order to take into account the operator precedence implied in this statement by the configuration of its brackets and the nature of its operators, this statement is turned into a one-dimensional REVERSE POLISH string which then has built-in operator precedence. The rules for output in REVERSE POLISH

are: (i) Operands are written into the REVERSE POLISH string in a backward sequence. A dummy operand denoted by Ω is written to the string when a prefix operator appears. (ii) Operators are stacked on an operator stack. If the operator on top of the stack is of higher or equal priority to the incoming operator, then the operator on stack is output. (iii) Brackets are treated as special operators. An opening bracket (goes on stack without comparison, and a closing bracket) causes the output of all operators down to the previous opening bracket (and removes this opening bracket (from stack.

The resultant REVERSE POLISH string reads:

$$+ * + * \gamma B\ \Omega\, eta\, \beta\alpha B\ v\ \Omega\ DT\ B$$

where a blank space immediately following an operand is inserted to act as break character for operands.

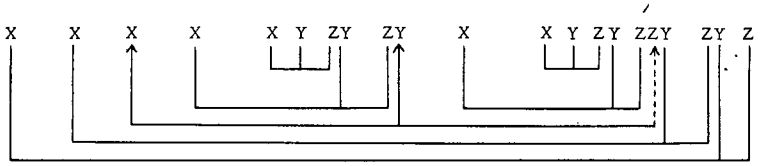
As the operator symbols can be made to point to the addresses for the operator names, it is easy in general to retrieve the operator names. However, some rules regarding the output of brackets and commas have to be formulated. The operators and operands in the REVERSE POLISH string are examined sequentially. An operand is recognized by the fact that it is followed by a blank character. The output format is determined by the following rules:

	Output	Add to test string
operator preceded by operator	operator (	X
operator preceded by operand	, operator (	YX
operand preceded by operator	operand	
operand preceded by operand	, operand)	YZ

The configuration of brackets and commas built up in this process must be remembered in order that correct sets of opening bracket, comma, and closing bracket (,) should be present when the output is completed. This is achieved by adding to a test string the letter X whenever an opening bracket (is output, Y whenever a comma , is output, and Z whenever a closing bracket) is output. In our example we have:

$$+ * + * \gamma B\ \Omega\ eta\ \beta\alpha B\ v\ \Omega\ dT\ B$$

sum(mult(sum(mult(Delsq(B, Ω), eta), Curl(Cross(B, v), Ω)), dT), B)



Whenever a Z is added to the test string, a triad of XYZ is completed and can be removed from the string. We then tentatively assign another closing

bracket, that is to say, add another Z to the remainder of the test string and test if a sequence of XYZ exists at its end. If yes, a closing bracket) can be definitely output at this point and another XYZ deleted from the string. When no more sequence of XYZ can be found at the end of the test string, the last Z is removed. It may be seen from the example that the closing bracket in dotted line has been assigned in this way. If this were absent, there would then be an unbalance of brackets and commas when the end of the line is reached. At this point the test string should become a null string.

As a consequenceof the use of REVERSE POLISH, all operands appear in reverse order to that in the original statement. While this presents no difficulty to the arithmetic unit of a computer, it is undesirable in Symbolic ALGOL II as we do not want to define operators like Cross in different order from their conventional definition. We, therefore, perform an interchange of operands within each triad of opening bracket, comma and closing bracket (,) . A counter which counts opening bracket (= +1, comma , = 0, and closing bracket) = -1 enables us to pick out the appropriate triad:

counter 1 2 3 4 5 5 4 33 4 5 5 44 322 1 0

sum(mult(sum(mult(Delsq(B, Ω), eta), Curl(Cross(B, v), Ω)), dT), B)

sum/mult(sum(mult(Delsq(B, Ω), eta), Curl(Cross(B,v),Ω)), dT) $B\

sum⟨B:mult(sum(mult(Delsq(B,Ω), eta),Curl(Cross(B, v),Ω)), dT)⟩

When the counter registers 1 and the character is an opening bracket (, this is replaced by / . When the counter again registers 1 and the character is a comma , , this is replaced by $. Finally, when the counter registers 0 and the character is a closing bracket) , this is replaced by \. We then simply interchange the operands delimited by the characters / $ and $ \ respectively. Operands within this triad have then been interchanged, and the corresponding brackets are inactivated by replacing the character / by ⟨ , $ by : , and \ by ⟩. The whole interchange process is repeated again, and this time the next level down will be interchanged. This continues until we obtain:

sum⟨B:mult⟨DT:sum⟨Curl⟨Ω:Cross⟨v:B⟩:mult⟨eta:Delsq⟨Ω:B⟩⟩⟩⟩⟩.

Finally, by replacing ⟨ → (, : → , , and ⟩ →) , and deleting the dummy operand Ω, we obtain the desired result:

Equate (B, sum(B, mult(DT, sum(Curl(Cross(v, B)), mult(eta, Delsq(B)))))).

The procedure Equate has to be created separately after the recognition of the = sign in the input equation.

In Fig. 1(a), the input to the translator program TRANSTAG is listed, as obtained from a Symbolic ALGOL I version of the three-dimensional MHD code TRINITY [1]. In Fig. 1(c), the corresponding output is listed. However, with a trivial modification, TRANSTAG can be made to handle input as shown in Fig. 1(b) where the equations are written in an entirely mathematical form. Here the user has to define ⟨x⟩ ≡ Cross, ⟨D⟩ ≡ dot, etc., although in principle, symbols such as ∇x, ∇^2 may be created on the teletype thus making the definitions unnecessary.

```
M NEW.RHO=RHO-DT*DIV(RHO*V);
 'FOR' C1=1 'STEP' 1 'UNTIL' 3 'DO'
M V=(RHO*V+DT*(-GRAD(RHO*TEM+DOT(B,B)/2)-DIV(RHO*TEN(V,V)-TEN(B,B))
M   +NU*DELSQ(RHO*V)))/NEWRHO;
 'FOR' C1=1 'STEP' 1 'UNTIL' 3 'DO'
M B=B+DT*(CURL(CROSS(V,B))+ETA*DELSQ(B));
M TEM=TEM+DT*(-DIV(TEM*V)+KAPPA*DELSQ(TEM)
M     +(2-GAMMA)*SAV(TEM)*DIV(V)+(GAMMA-1)*ETA*DOT(CURL(B),CURL(B))
M     /SAV(RHO)+(GAMMA-1)*NU*(DOT(CURL(V),CURL(V))+DIV(V)↑2));
```

(a)

```
M  SCALAR EQUATION 1;
M  DRHO/DT =-DIV(RHO*V);
M  VECTOR EQUATION 2;
M  DV/DT=(DIV((B<T>B)-RHO*(V<T>V))-GRAD(RHO*TEM+(B<D>B)/2.0)
M        +NU*DELSQ(RHO*V))/RHO;
M  VECTOR EQUATION 3;
M  DB/DT=CURL(V<X>B)+ETA*DELSQ(B);
M  SCALAR EQUATION 4;
M  DTEM/DT=-DIV(TEM*V)+KAPPA*DELSQ(TEM)
M           +(2-GAMMA)*SAV(TEM)*DIV(V)+(GAMMA-1)*ETA*(CURL(B)<D>CURL(B))
M           /SAV(RHO)+(GAMMA-1)*NU*((CURL(V)<D>CURL(V))+DIV(V)↑2));
```

(b)

```
EQUATE( NEW RHO,DIFF(RHO,MULT(DT,DIV(MULT(RHO,V)))))-(000600)
'FOR' C1=1 'STEP' 1 'UNTIL' 3 'DO'-(000700)
EQUATE( V,QUOT(SUM(MULT(RHO,V),MULT(DT,SUM(DIFF(DIFF(BLANK,GRAD(SUM(MUL
  (004200)
T(RHO,TEM),QUOT(DOT(B,B),RNUM(2)))))),DIV(DIFF(MULT(RHO,TEN(V,V)),TEN(B,
  (004300)
B)))),MULT(NU,DELSQ(MULT(RHO,V))))))),NEWRHO))-(004400)
'FOR' C1=1 'STEP' 1 'UNTIL' 3 'DO'-(004500)
EQUATE( B,SUM(B,MULT(DT,SUM(CURL(CROSS(V,B)),MULT(ETA,DELSQ(B)))))))-(00
  5600)
EQUATE( TEM,SUM(TEM,MULT(DT,SUM(SUM(SUM(SUM(DIFF(BLANK,DIV(MULT(TEM,V))
  (009800)
),MULT(KAPPA,DELSQ(TEM)))),MULT(MULT(DIFF(RNUM(2),GAMMA),SAV(TEM)),DIV(V
  (009900)
))),QUOT(MULT(MULT(DIFF(GAMMA,RNUM(1)),ETA),DOT(CURL(B),CURL(B))),SAV(R
  (010000)
HO))),MULT(MULT(DIFF(GAMMA,RNUM(1)),NU),SUM(DOT(CURL(V),CURL(V)),EXP(DI
  (010100)
V(V),2)))))))-(010200)
```

(c)

FIG. 1. (a) Input to the translator program TRANSTAG as obtained from a Symbolic ALGOL I version of the three-dimensional MHD code TRINITY.
(b) Modified input.
(c) Corresponding output to input shown in (a).

On the KDF9 at Culham Laboratory, TRANSTAG took 18 minutes to process the four equations of TRINITY. This is not extremely fast; however, such translation need only be performed once for a given problem and thus this time would be quite tolerable.

ACKNOWLEDGEMENTS

The authors wish to express their thanks to Mr. W. Holman for assistance in STAGE-2 programming, and to Dr. P.C. Poole for the implementation of STAGE 2 at Culham and for many helpful discussions.

REFERENCES

[1] ROBERTS, K.V., BORIS, J.P., The solution of partial differential equations using a Symbolic Style of ALGOL, J. comput. Phys. 8, 1 (1971) 83.

[2] PETRAVIĆ, M., ROBERTS, K.V., KUO-PETRAVIĆ, G., The automatic optimization of Symbolic ALGOL programmes, to be published in J. comput. Phys.

[3] KUO-PETRAVIĆ, G., PETRAVIĆ, M., ROBERTS, K.V., The Translation of Symbolic ALGOL I to Symbolic ALGOL II by the STAGE 2 Macro-Processor, A User's Guide, PDN 4/71, Culham Laboratory.

[4] WAITE, W.M., The mobile programming system STAGE 2, Communs Ass. comput. Mach. (1970) 415.

AUTOMATIC OPTIMIZATION OF SYMBOLIC ALGOL PROGRAMS: I. GENERAL PRINCIPLES*

M. PETRAVIĆ, G. KUO-PETRAVIĆ
Department of Engineering Science,
Oxford University, Oxford

K.V. ROBERTS
Culham Laboratory, Abingdon, Berks,
United Kingdom

Abstract

AUTOMATIC OPTIMIZATION OF SYMBOLIC ALGOL PROGRAMS: I. GENERAL PRINCIPLES.

The symbolic style of programming referred to as Symbolic ALGOL I [1] appears to have a number of advantages when applied to the solution of sets of nonlinear partial differential equations. Programs written in that style are clear, elegant and concise and their modular structure enables large parts of the programs to be used over and over again for many different problems. Such programs, however, tend to be slow because they involve a large number of nested procedure calls at execution time.

Finite difference methods in several dimensions require in general that a relatively small number of equations be solved a large number of times and much is gained if these nested procedure calls are executed only once. This is achieved by a generator or translator program, written in ALGOL, which processes input written in a related style named Symbolic ALGOL II. Usually, only finite difference equations in very compact symbolic form are input, while output is completely explicit and can be in a number of computer languages. Of greatest interest are ASSEMBLER code modules automatically produced in this way. They are competitive in speed with fully hand-optimized FORTRAN versions and are produced effortlessly and error-free, so that complex sets of equations can readily be programmed or alterations made. For production runs, these modules can be incorporated into a FORTRAN control program.

1. INTRODUCTION

This paper describes how ALGOL 60 can be used as a powerful macro-processor which enables the symbolic expressions of classical vector analysis to generate efficient target code automatically by the use of controlled side effects. The target languages produced so far have been IBM-360 ASSEMBLER code, FORTRAN, ALGOL and ICL KDF9 USERCODE, but it appears that any language might be generated in a similar way. The target code can be optimized by physical symmetry declarations; for example, if $V_\theta = 0$ (no rotation) and $\partial f/\partial z = 0$ for all functions f (no z-dependence), then appropriate declarations can be used to suppress terms in which such quantities occur as products. The method can be used for generalized orthogonal curvilinear co-ordinates and an example will be given. Finally, it would seem that the method might be extended without difficulty to other kinds of symbolic formalism.

A previous paper [1] showed how ALGOL could be used for the symbolic solution of problems in computational physics, especially those in which

* For Part II, see Ref.[6].

sets of partial differential equations are solved by finite difference methods using a discrete mesh. A vector equation such as

$$\frac{\partial \vec{B}}{\partial t} = \text{Curl}\,(\vec{V}\times\vec{B}) + \eta\nabla^2\vec{B} \tag{1}$$

can be programmed symbolically in the closely similar form

$$\text{AB[C1, Q]} := \text{CURL(CROSS(V, B))} + \text{ETA} \times \text{DELSQ(B)}; \tag{2}$$

a style of programming which has been termed Symbolic ALGOL I [1]. Here the left-hand side of Eq. (2) is an array element representing the magnetic field B in the C1-direction (C1 = 1, 2 or 3) at the mesh point Q, while most of the identifiers on the right-hand side are symbolic operators or functions which are closely analogous to their counterparts in Eq. (1) and are represented by real procedures. Details of the choice of co-ordinate system, the number of dimensions, the boundary conditions and the difference scheme are excluded from Eq. (2) and are dealt with at a lower level just as in the familiar symbolic notations of mathematical physics.

Symbolic ALGOL I (SA/I) enables complex problems to be coded in a concise form which is virtually system-independent and should be readily intelligible to physicists because it is close to the mathematical language which they normally use. By way of example, Table I shows the partial differential equations which are used in the 3D magnetohydrodynamic TRINITY code [1], while Table II shows the same equations programmed in SA/I. The differences are fairly minor and are partly due to the restricted class of symbols currently available on computer input devices such as the teletype or card punch.

The advantages of symbolic notation are clear enough. The ALGOL procedure-operators are neat and concise and have the same formal properties as their mathematical counterparts, so that the manipulation of statements and the construction of new expressions are quick, intelligible and easy to check for errors. A typical example is the operator CURL, represented in a Cartesian co-ordinate system by the short procedure:

$$\underline{\text{real procedure}}\ \text{CURL(A)};\ \underline{\text{real}}\ \text{A};\ \text{CURL} := \text{RP(DEL(RP(A)))} - \text{RM(DEL(RM(A)))}; \tag{3}$$

Here RP and RM are rotation operators which rotate the 1, 2, 3-components of vectors or tensors in either the positive (RP) or negative (RM) directions (Fig. 1), while DEL is a finite difference operator. These rotation operators are reciprocal to one another so that

$$\text{RP(RM(A))} = \text{RM(RP(A))} = \text{A} \tag{4}$$

while the property

$$\text{RP(RP(A))} = \text{RM(A)} \tag{5}$$

is also often used in three dimensions. The use of vector and tensor operators ensures that statements are independent of the co-ordinate system (covariant), the components being hidden and appearing only at execution time.

TABLE I. 3D MHD EQUATIONS USED IN THE TRINITY CODE

Continuity equation	$\frac{\partial \rho}{\partial t} = - \nabla \cdot \rho \vec{v}$
Momentum equation	$\frac{\partial (\rho v_i)}{\partial t} = - \frac{\partial}{\partial x_j} (p_{ij}) + \nu \nabla^2 \rho v_i$
Magnetic equation	$\frac{\partial \vec{B}}{\partial t} = \vec{\nabla} \times (\vec{v} \times \vec{B}) + \eta \nabla^2 \vec{B}$
Temperature equation	$\frac{\partial T}{\partial t} = - \vec{\nabla} \cdot (T \vec{v}) + (2 - \gamma) T \vec{\nabla} \cdot \vec{v} + \kappa \nabla^2 T$ $+ (\gamma - 1) \eta j^2/\rho + (\gamma - 1) \nu [(\nabla \times \vec{v})^2 + (\vec{\nabla} \cdot \vec{v})^2]$
Pressure	$p_{ij} = \rho T \delta_{ij} + \rho v_i v_j + \frac{B^2}{2} \delta_{ij} - B_i B_j$
Current	$\vec{j} = \vec{\nabla} \times \vec{B}$

All these properties combined with modularity and portability of programs [2] make SA/I a powerful tool for the quick and error-free development of large and complex physics or engineering programs. However, SA/I executes quite slowly because of the great number of nested procedure calls and is therefore not too useful for 2D and 3D production runs, although this depends on how well the ALGOL 60 compiler has been written. Its main application to date lies in the testing of prototype programs on a coarse mesh over a few time steps. In this way, standard test results are obtained for comparison with future better-optimized and faster versions of the same program, written, for example, in ordinary ALGOL, FORTRAN or ASSEMBLER code [1].

The aim of the present paper is to carry the theory of Symbolic ALGOL one stage further. We shall show that by a further slight transformation of a vector expression such as Eq.(2) it can be made to generate optimized code automatically. In this new style of programming, which is termed Symbolic ALGOL II (SA/II), Eq.(2) in fact becomes

```
EQUATE(B, SUM(B, MULT(DT, SUM(CURL(CROSS(V, B)), MULT(ETA, DELSQ(B))))));        (6)
```

and the statements of Table II are replaced by those of Table III. The reason for this transformation is to replace the arithmetic operators +, -, ×, /, :=, which occupy a privileged position in high-level languages, by their generalized counterparts SUM, DIFF, MULT, QUOT which are real procedures, and EQUATE which is a procedure. Once this has been done, these procedures can, of course, be given any interpretation that we choose, and they can in particular be made to generate optimized code in any desired programming language by means of side effects as explained in Section 2.

TABLE II. 3D MHD EQUATIONS PROGRAMMED IN SYMBOLIC ALGOL I

```
procedure INVOKE DIFFERENCE EQUATIONS;

begin

    CONTINUITY EQUATION: DT:= 2× DELTA T; C1:=C2:=1;

    Q:=1+I+1+(J+1)×PI+(K+1)×PI×PJ;

    NEW RHO:=RHO-DT× DIV(RHO× V);

    MOMENTUM EQUATION:  DT:=2× DELTA T/(1 + NU/EPS);

        for C1:=1,2,3 do

        AV[C1,Q]:=(RHO×V+DT×(-DIV2(P) + NU × DELSQ(RHO×V)))/NEW RHO;

        ARHO[Q]:=NEW RHO;

    MAGNETIC EQUATION:    DT:=2× DELTA T/(1 + ETA/EPS);

        for C1:=1,2,3 do

        AB[C1,Q]:=B+DT×(CURL(CROSS(V,B)) + ETA × DELSQ(B));

    TEMPERATURE EQUATION: DT:=2×DELTA T/(1 + KAPPA/EPS); C1:=1;

        ATEM[Q]:=TEM+DT×(-DIV(TEM× V) + KAPPA × DELSQ(TEM)

                + (2 - GAMMA)× SAV(TEM)× DIV(V)+(GAMMA - 1)×(ETA ×

                SQM(CURL(B))/SAV(RHO)+NU×(SQM(CURL(V))+DIV(V)↑2 )));

end;

real procedure P;

P:=if C1=C2 then (RHO×(TEM+V×V)+0.5×DOT(B,B)-B×B) else (RHO×V×V2-B×B2);
```

Note: The differences in the treatment of p and $\vec{j}$ between Tables I, II and III are not essential.

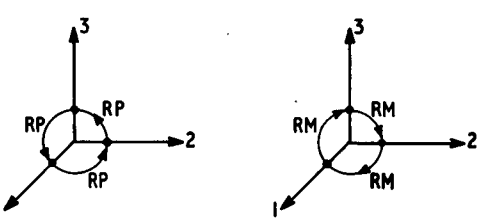


FIG.1. Positive and negative rotation operators.
The operators RP, RM rotate vector components cyclically in the positive and negative directions, respectively, and satisfy the symbolic relations $R_+^3 = R_-^3 = R_+R_- = R_-R_+ = 1$, from which $R_+^2 = R_-$ etc. (here $R_+ \equiv$ RP, $R_- \equiv$ RM).

TABLE III. 3D MHD EQUATIONS PROGRAMMED IN SYMBOLIC ALGOL II

```
CONTINUITY EQUATION:
EQUATE(NEWRHO, DIFF(RHO, MULT(DT(1), DIV(MULT(RHO, V)))));

MOMENTUM EQUATION:
   for C1:=1,2,3 do
EQUATE(V, QUOT(SUM(MULT(RHO, V), MULT(DT(2), DIFF(SUM(DIV
(DIFF(TEN(B, B), MULT(RHO(TEN(V, V)))), MULT(NU, DELSQ(MULT(RHO, V)))),
GRAD(SUM(MULT(RHO, TEM), MULT(RNUM(0.5), DOT(B, B)))))), NEWRHO));

MAGNETIC EQUATION:
   for C1:=1,2,3 do
EQUATE(B, SUM(B, MULT(DT(3), SUM(CURL(CROSS(V, B)), MULT(ETA, DELSQ(B))))));

TEMPERATURE EQUATION:
EQUATE(TEM, SUM(TEM, MULT(DT(4), SUM(DIFF(SUM(MULT(MULT(DIFF
(RNUM(2.0), GAMMA), SAV(TEM)), DIV(V)), MULT(KAPPA, DELSQ(TEM))), DIV(MULT
(TEM, V))), SUM(QUOT(MULT(DIFF(GAMMA, RNUM(1.0)), MULT(ETA, SQUARE
(CURL(B)))), SAV(RHO)), MULT(DIFF(GAMMA, RNUM(1.0)), MULT(NU, SUM
(SQUARE(CURL(V)), EXP(DIV(V), INUM(2)))))))));
```

Note: Equations (5) and (6) have been incorporated into the main equations although they can be defined separately. The four calls of the procedure DT take into account the Dufort-Frankel factors [1].

The languages generated so far have been IBM-360 ASSEMBLER code, optimized ALGOL and FORTRAN, and ICL KDF9 USERCODE which is similar to REVERSE POLISH and therefore has a theoretical as well as a practical interest. The transformations from Table I and Table II to Table III obey prescribed rules and we have in fact carried them out automatically by two separate methods. One method uses the STAGE-2 Macro-Processor [3, 4], and the other uses ALGOL 60 as a string processor and syntax analyser [5]. In practice, these transformations are not difficult to perform by hand.

An SA/II generator program looks very much like the corresponding SA/I calculational program except that some of the auxiliary statements must also be changed from form (2) to form (6), so that, for example, CURL becomes

```
real procedure CURL(X); real (X); CURL := DIFF(RP(DEL(RP(X))), RM(DEL(RM(X))));    (7)
```

The purpose of (7) is, however, rather different from that of (3) because instead of actually calculating numerical values directly, the program now works out which programming instructions are needed to calculate these values and then generates the instructions, either printing them or punching them out on cards or placing them in a file for subsequent execution. Clearly, there is a close correspondence between the two processes of cal-

culation and code generation – for example, in both SA/I and SA/II the storage locations are evaluated by manipulating symbolic operators – and so CURL, DEL and other operators appear virtually unaltered. The automatically-generated modules can be incorporated into control programs which are written in FORTRAN, ALGOL or any other convenient language, as discussed in Section 6.

An SA/II generator program is not a compiler, and it is shorter, more flexible and easier to write than most compilers. We shall endeavour to explain the relation in Section 6. Since it is coded in ALGOL and has at least as many nested procedure calls as SA/I, it may also not be as fast as some compilers. However, the point is that each statement in Table III is executed only once, instead of many millions of times as in a normal run. Code generation from Table III actually occupies about 10 seconds on the IBM 360/91, which is comparable with ordinary job overheads and much less than the duration of a typical production run which may last for several minutes or even hours. The only real requirement is that the generated ASSEMBLER code should be efficient, and it turns out in practice to be slightly more efficient than the corresponding FORTRAN written by a good programmer and compiled with the IBM FORTRAN H Option 2 Compiler.

The basic idea of SA/II is quite straightforward and will be explained in Section 2, namely that side effects of typed procedures can be used to generate code. Some complications arise in a practical program because it is desirable to make the generated code as efficient as possible. As we have already mentioned, it should be possible in a problem in which $V_\theta = 0$ or $\partial f/\partial z = 0$ to declare these symmetry conditions to the generator program and to have it automatically eliminate all terms which are then known to be identically zero. This can be achieved by making use of the fact that the real procedures in a statement such as (6) pass numerical values to one another. The simplest version of SA/II outlined in Section 2 treats these values as dummies, but they can be employed for a variety of purposes including optimization. A zero value is assigned to basic terms or products which are known to be identically zero in the physical problem, and this value is used to control the way in which the code is generated. Unnecessary brackets and signs are eliminated in a similar fashion.

Code generation is carried out by a real procedure TRIPOP (triple operator), and operators such as SUM, DIFF, MULT, QUOT simply call TRIPOP to generate the appropriate output. Although TRIPOP is quite short, its working is fairly complex, and to avoid burdening the present discussion it will be described in detail elsewhere [6]. A brief account is, however, given in Section 3.

Since ALGOL is likely to be useful for other types of symbolic manipulation, it may be appropriate, after some examples have been given, to list those features of the language that we have found to be important, and this is done in Section 8. It will, however, become evident almost from the beginning that recursion and call by name play a significant part, together with a number of other facilities which are not available in a language such as FORTRAN. A more practical point is that although ALGOL allows identifiers to have arbitrary length, some compilers (including that of the IBM 360) distinguish only the first six characters. To make this paper clearer, we have in some cases used long English-language identifiers, but in the program itself [7] the identifiers are restricted to six characters or less to avoid possible clashes.

2. CODE GENERATION

The generation of code by an expression such as

$$\text{SUM(X, MULT(Y, Z))} \tag{8}$$

is simple to explain. We first need a procedure PRINT (⟨string⟩) which will output an arbitrary string of characters, taking into account any special requirements to leave gaps at the beginning and end of the line. In terms of this we define the algebraic symbols:

```
procedure PLUS; PRINT('+');
procedure STAR; PRINT('*');
```
(9)

and the symbols to be used as identifiers in the generated code:

```
real procedure X; begin PRINT('X'); X:=1; end;
real procedure Y; begin PRINT('Y'); Y:=1; end;
real procedure Z; begin PRINT('Z'); Z:=1; end;
```
(10)

Finally, there are two arithmetic operators:

```
real procedure MULT(A, B); real(A, B);
    begin real CALL; CALL:=A; STAR; CALL:=B; MULT:=1; end;
```
(11)

and

```
real procedure SUM(A, B); real(A, B);
    begin real CALL; CALL:=A; PLUS; CALL:=B; SUM:=1; end;
```
(12)

2.1. Explanation

Exactly what happens is illustrated by the tree shown in Fig. 2. When the statement 'CALL:=A' is encountered, the real procedure X is invoked by name from SUM, causing a symbol 'X' to be generated. The local variable CALL in SUM is set equal to 1 but this value is disregarded. A '+' symbol is next produced. Finally, MULT is entered by 'CALL:=B' which generates the symbols 'Y', '*', 'Z' in the same way, so that we get

$$X+Y*Z \tag{13}$$

as required.

Several points may be noticed:

(a) Typed procedures are used to generate the code.

(b) The actual generation takes place by means of side effects; the statement

```
CALL:=A;
```

causes the string of symbols associated with the formal parameter A to be constructed, however complex an expression A may represent.

(c) A precise order is forced on these side effects, independently of the order in which the ALGOL compiler writer may choose to evaluate the terms in an arithmetic expression such as P+Q or P×Q.

(d) It is immaterial whether real or integer procedures are used, but we have chosen real procedures for compatibility with SA/I.

(e) At this stage, the values associated with the real procedures have no significance; only the side effects are important. In Section 3, we shall, however, indicate how a suitably-chosen assignment of values can be used to remove expressions which are known to be identically zero.

2.2. Generation of indices

The variable indices defining a particular component of a vector variable, and the mesh point at which it is to be evaluated, are specified by global variables whose values are controlled by the cyclic rotation operators RP(X), RM(X) and the displacement operators EP(X), EM(X). The action of these operators relies on the ALGOL "call by name" facility [8, 9]. In the example

```
real procedure RP(X); real X;
    begin C1 := CP[C1]; RP := X; C1 := CM[C1]; end;
```

the procedure X is made to use the value of C1 defined within the procedure RP, because it is evaluated at the time the statement RP := X; is executed and not at the time of the call to RP, as would be the case if the "call by value" alternative were used.

2.3. Generation of REVERSE POLISH

The reader might be excused for wondering what exactly has been achieved so far; starting from the algebraic expression (13), we have converted it automatically or by hand into the SA/II form (8) and then proved that this is capable of reproducing the original expression. The real advantages are, first, that the generated code can represent a considerable expansion of the original which may be in symbolic vector or tensor

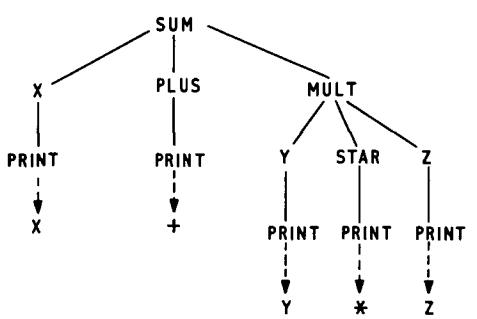


FIG. 2. Generation of code by the expression SUM(X, MULT(Y, Z)).

form, and second, that by making small changes in the basic procedures such as PLUS, STAR, X, Y, Z, MULT, SUM we can generate the code in a lower-level and therefore more efficient language.

To illustrate this second point we show how to convert (8) into the REVERSE POLISH form

$$X, Y, Z, *, +, \tag{14}$$

which is in one-to-one correspondence with the ASSEMBLER language (or 'USERCODE') of the ICL KDF9 on which Symbolic ALGOL was first developed. To do this we simply modify the five procedures (9) and (10) in order to add an extra comma, e.g.

```
procedure PLUS; PRINT('+, '); (15)
```

and then reverse the order of the second and third statements in the arithmetic operators (11) and (12), e.g.:

```
CALL:=A; CALL:=B; STAR; MULT:=1; (16)
```

which then enables (8) to generate (14).

2.4. Brackets

The question of brackets has so far not been taken into account in generating algebraic expressions such as (13). Thus at present

$$X \times (Y+Z) \tag{17}$$

would be converted by

$$\mathrm{MULT}(X, \mathrm{SUM}(Y, Z)) \tag{18}$$

into

$$X * Y+Z \tag{19}$$

which is wrong, although the corresponding REVERSE POLISH string is still correct:

$$X, Y, Z, +, *, \tag{20}$$

The simplest way of correcting this is to enclose the output from each arithmetic operator in a pair of brackets by means of two procedures OPENB, CLOSEB, so that, for example, (12) becomes

```
real procedure SUM(A, B); real(A, B);
begin real CALL; OPENB; CALL:=A; PLUS; CALL:=B; SUM:=1; CLOSEB; end; (21)
```

Then expressions (8) and (18) generate the correct output

$$(X+(Y*Z)) \tag{22}$$

and

$$(X * (Y+Z)) \tag{23}$$

respectively.

A complicated expression will now produce a large number of unnecessary pairs of brackets, but although these make the expression difficult for a human to understand, they should not trouble a FORTRAN or ALGOL compiler; in fact, they might well shorten the compilation time since questions of operator precedence need no longer be resolved. However, in order to make the generated code more elegant and intelligible, we have implemented a method for removing the unnecessary bracket pairs; this is described in Ref.[6].

Another difficulty which occurs with the KDF9 is that the nesting store or arithmetic stack has finite depth (in practice about 13), so that a sufficiently long string of identifiers unrelieved by operators could cause it to overflow. To monitor this situation (which has not yet occurred in the problems that we have treated), one can introduce a level counter and print out a warning to the user when overflow is detected. In most cases he can then permute the order in the SA/II expression to produce a better pattern. For example,

$$\text{SUM(MULT(Y, Z), X)} \tag{24}$$

instead of (8) leads to

$$\text{Y, Z, *, X, +,} \tag{25}$$

which requires a smaller stack. Overflow is more likely to occur with the IBM 360 which has only four floating-point registers, and here we perform automatic dumping and restoring of registers when necessary, as explained in Ref.[6].

3. PROCESSING THE EQUATIONS

The central procedure of the SA/II generator program is EQUATE, which for ALGOL output is

```
procedure EQUATE(X, Y); real X, Y;
    begin real CALL;
        NEXTLINE; PRINT := false;
        if X = 0 then go to EXIT;
        PRINT := true; SIGN := 0;                    (26)
        CALL := X; ASSIGN;
        if Y = 0 then TEXT(1, '0');
        SEMICOLON;
    EXIT;
    end;
```

A similar version is used for any other target code, but the explanation is simpler for this particular example and may give some idea of the power of the symbolic method. ASSIGN generates the assignment symbol '.=' for the IBM 360.

Suppose that (26) is called by the statement

$$\text{EQUATE(B, SUM(B, MULT(DT, CURL(CROSS(V, B)))));} \qquad (27)$$

as in Table III but without the resistive diffusion term. Because this is a vector equation it will be called three times with C1 = 1, 2, 3 and we shall suppose that B_z has been declared identically zero. The tree of real procedure calls is indicated in Fig. 3.

The auxiliary call to NEXTLINE does any editing that is required, e.g. ruling a line across the output page, and printing is then switched off. The left-hand side X of the expression, in this case B_x, B_y or B_z, is next tested to see if it is identically zero, which, as explained in Section 1, will be represented by a zero value. If so, there is no point in generating

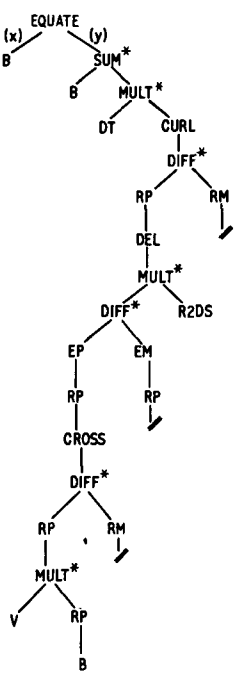


FIG. 3. Procedure tree for the magnetic equation (27).
Each arithmetic operator indicated by * calls the TRIPOP operator recursively to control the optimization process. Branches broken off with // are similar to the full branch shown on the diagram. Terminal symbols V, B, DT, R2DS generate the code, while RP, RM manipulate the global component variable C1, and EP, EM manipulate the mesh location.

anything, and the right-hand side is skipped (C1 = 3). Note that since X is a real procedure, the statement if X = 0 then ..... involves further operations at a lower level[1].

Assuming that the component is not identically zero, we switch printing on again, make sure that no + sign will be printed, and reference X once more, subsequently calling the procedure ASSIGN. This series of actions will generate some expanded form such as

$$B1(/Q/).= \tag{28}$$

depending on the hardware representation and on the way in which array components are being referenced. Note that the numerical value in (28) is that of the formal parameter V.

The whole of the testing and generation of the right-hand side is now contained in the deceptively simple statement

$$\text{if } Y = 0 \text{ then } TEXT(1, '0'); \tag{29}$$

Some possible cases for the right-hand side Y are:

(a) Constant. Since printing is now switched on, the numerical representation of the constant is generated.

(b) Non-subscripted variable. The character string corresponding to this variable is generated.

(c) Array variable. If this variable has been declared to be identically zero, it will not be printed and the value 0 will be returned, causing the second part of the if statement to print '0'. If it is not identically zero, it will print its own character representation, including any necessary vector or tensor components.

(d) Arithmetic expression. The outermost arithmetic operator calls TRIPOP which switches off the printing and tests the whole of the expression, TRIPOP being called again recursively by each of the internal arithmetic operators, including those in CURL and CROSS. If it is finally determined to be identically zero, then '0' is printed. Otherwise this outermost TRIPOP initiates a second scan which prints all those sub-expressions which were found to be not identically zero the first time they were tested.

As an example, Table IV shows ALGOL 60 target code generated for the IBM 360 for Maxwell's two equations

$$\frac{\partial \vec{E}}{\partial t} = \text{Curl}\,\vec{H} \qquad \frac{\partial \vec{H}}{\partial t} = -\,\text{Curl}\,\vec{E}$$

in orthogonal curvilinear co-ordinates using a mesh of size (14×10×42). The source statements were:

---

[1] Since X is real it is perhaps not good practice to test whether or not it is exactly zero but we have found no difficulty on the ICL KDF9 or IBM 360. The procedure values are always set to integers or sums or differences of small integers and it is hard to see why trouble should arise with any computer. If it does, one can simply test for $X < \epsilon$ where $0 < \epsilon \ll 1$.

```
for C1 = 1, 2, 3 do
    EQUATE(EFIELD, SUM(EFIELD, MULT(DCT(8), CURL(HFIELD))));
for C1 = 1, 2, 3 do
    EQUATE(HFIELD, DIFF(HFIELD, MULT(DCT(7), CURL(EFIELD))));
```
(30)

In this case no symmetry conditions were imposed. DCT7 and DCT8 are time-step factors which will usually be the same.

TABLE IV. MAXWELL'S EQUATIONS IN 3D ORTHOGONAL CURVILINEAR CO-ORDINATES GENERATED IN IBM 360 ALGOL

```
'COMMENT'----------------------------------------;
HFIELD1(/Q/) . = HFIELD1(/Q/)-DCT7*((EFIELD3(/Q+ 14/)*H3(/+2/)-EFIELD3(/Q- 1
        4/)*H3(/-2/))*R2DQ2-(EFIELD2(/Q+ 140/)*H2(/+3/)-EFIELD2(/Q- 140/)*
        H2(/-3/))*R2DQ3)*RHPHM1;
'COMMENT'----------------------------------------;
HFIELD2(/Q/) . = HFIELD2(/Q/)-DCT7*((EFIELD1(/Q+ 140/)*H1(/+3/)-EFIELD1(/Q- 14
        Q/)*H1(/-3/))*R2DQ3-(EFIELD3(/Q+ 1/)*H3(/+1/)-EFIELD3(/Q- 1/)*
        H2(/-1/))*R2DQ1)*RHPHM2;
'COMMENT'----------------------------------------;
HFIELD3(/Q/) . = HFIELD3(/Q/)-DCT7*((EFIELD2(/Q+ 1/)*H2(/+1/)-EFIELD2(/Q-
        1/)*H2(/-1/))*R2DQ1-(EFIELD1(/Q+ 14/)*H1(/+2/)-EFIELD1(/Q- 14/)*
        H1(/-2/))*R2DQ2)*RHPHM3;
'COMMENT'----------------------------------------;
EFIELD1(/Q/) . = EFIELD1(/Q/)+DCT8*((HFIELD3(/Q+ 14/)*H3(/+2/)-HFIELD3(/Q- 1
        4/)*H3(/-2/))*R2DQ2-(HFIELD2(/Q+ 140/)*H2(/+3/)-HFIELD2(/Q- 140/)*
        H2(/-3/))*R2DQ3)*RHPHM1;
'COMMENT'----------------------------------------;
EFIELD2(/Q/) . = EFIELD2(/Q/)+DCT8*((HFIELD1(/Q+ 140/)*H1(/+3/)-HFIELD1(/Q- 14
        Q/)*H1(/-3/))*R2DQ3-(HFIELD3(/Q+ 1/)*H3(/+1/)-HFIELD3(/Q- 1/)*
        H3(/-1/))*R2DQ1)*RHPHM2;
'COMMENT'----------------------------------------;
EFIELD3(/Q/) . = EFIELD3(/Q/)+DCT8*((HFIELD2(/Q+ 1/)*H2(/+1/)-HFIELD2(/Q-
        1/)*H2(/-1/))*R2DQ1-(HFIELD1(/Q+ 14/)*H1(/+2/)-HFIELD1(/Q- 14/)*
        H1(/-2/))*R2DQ2)*RHPHM3;
```

Note: In this example, PI = 14 and PJ = 10, hence the displacements in the 1, 2, 3-directions are (1, -1), (14, -14) and (14 x 10, -14 x 10) respectively.

4. PHYSICAL CONSTANTS AND VARIABLES

Provision is made for handling non-subscripted variables which may be constants or functions only of the time, and scalar, vector and tensor functions. A typical 'declaration' specified by the user is, for example,

```
real procedure B; B:=VECTOR(5,1,'B',1,1,0);                    (31)
```

This calls the real procedure VECTOR which in the ALGOL 60 output version reads

```
real procedure VECTOR(ORDINAL NUMBER, LENGTH, S, V1, V2, V3);
    integer ORDINAL NUMBER, LENGTH, V1, V2, V3; string S;
    begin
        VECTOR:= if C1=1 then V1 else if C1=2 then V2 else V3;
        if not PRINT then go to EXIT;
        SIGN IT; TEXT(LENGTH, S);
        COMPONENT; SHIFT;
    EXIT:
end;                                                           (32)
```

The parameter ORDINAL NUMBER is not being used here; it controls the actual storage region used by B which is important in some ASSEMBLER language versions and is retained for consistency. LENGTH gives the length of the identifier string 'B' which will be used in target statements, instructions and comments, in this case 1. The last three parameters V1, V2, V3 define whether or not the x, y, z components are identically zero. In this case, the user has specified V3 = 0 so that B_z is taken to be identically zero. Because VECTOR returns the value 0 whenever C1 = 3, the z-component of the magnetic equation will be suppressed altogether (Section 8), and all terms in which B_z occurs as a product will be suppressed on the right-hand side of any other equation. Otherwise, when PRINT = true, the procedure VECTOR will proceed as follows:

SIGN IT	Examine the value of a global variable SIGN and output either a preliminary '+', '-', or no sign.
TEXT(LENGTH, S)	Output the string S of length LENGTH, in our example 'B'.
COMPONENT	Output the current value of the component C1, e.g. '2'.
SHIFT	Output '[' or '(/' followed by the signed numerical value of the current displacement from the local mesh origin, and finally a closing bracket, ']' or '/)'.

A typical output for ALGOL on the IBM 360 is

$$+B2(/Q+73/) \tag{33}$$

and similarly for FORTRAN except for the absence of the slashes. Several other versions have been developed; for example, the displacements can be handled symbolically so that the code does not have to be regenerated when the mesh size is changed, or one might generate 'BY' instead of 'B2' for clarity.

The operators RP, RM act on the global variable C1 as determined by algebraic and analytic operators such as DOT, CROSS, DIV, CURL and therefore enable the correct component label to be calculated. Similarly, vector translation operators EP, EM, which are called whenever a space derivative occurs, are used to calculate the position on the lattice relative to the central point Q of the local mesh 'molecule'. In (33) this appears as a numerical displacement of 73 words within the region of core store occupied by the array B2.

A scalar is handled in a precisely similar way by the real procedure SCALAR which requires only a single value 0 or 1 and does not need to call COMPONENT, and correspondingly for constants on the one hand, and tensors on the other.

4.1. Addressing of variables

We have already mentioned that the generated code can be in any chosen computer language. Whatever the language, however, we obviously have to be able to print some form of address at which the value of a particular variable defined on a specific mesh point is stored. The addressing can be completely symbolic, e.g. in ALGOL, via an array

$$B[C1, Q+DX-DY], \tag{34}$$

it can be coded numerically as in KDF9 USERCODE

$$V34P6M15 \tag{35}$$

(location 34 of variable block 6 modified by register 15), or it can be partly numerical and partly symbolic as in IBM-360 ASSEMBLER code

$$\left.\begin{array}{ll} 0052(,DISPQ) & BY(I,J-1,K) \\ 0312(PLUSDZ,DISPQ) & BZ(I,J,K+1) \end{array}\right\} \tag{36}$$

Here the displacement in bytes defines both the variable and the position in the xy-plane relative to the centre of the molecule, DISPQ is a general register which contains the current centre, while PLUSDZ is a register which shifts one mesh unit in the z-direction. The comments on the right-hand side give the FORTRAN notation. A fuller example of IBM-360 ASSEMBLER code produced in this way is shown in Table V. Several other choices are possible. In any case, the address will depend on the variable in question, on the co-ordinate direction on to which a vector variable is being projected, and on the mesh point at which the expression should be

TABLE V. IBM-360 ASSEMBLER CODE GENERATED FOR THE SECOND COMPONENT OF THE MAGNETIC EQUATION

Operation code	Location	Variable	Level	
			00	001
LE	0,0308(,DISPQ)	B2+0+0	01	002
LE	2,CONSTANT+008	DT	02	003
LE	4,0296(PLUSDZ,DISPQ)	V2+0+0	03	004
ME	4,0312(PLUSDZ,DISPQ)	B3+0+0	03	005
LE	6,0300(PLUSDZ,DISPQ)	V3+0+0	04	006
ME	6,0308(PLUSDZ,DISPQ)	B2+0+0	04	007
SER	4,6		03	008
LE	6,0296(MINSDZ,DISPQ)	V2+0+0	04	009
ME	6,0312(MINSDZ,DISPQ)	B3+0+0	04	010
SER	4,6		03	011
LE	6,0300(MINSDZ,DISPQ)	V3+0+0	04	012
ME	6,0308(MINSDZ,DISPQ)	B2+0+0	04	013
AER	4,6		03	014
ME	4,CONSTANT+016	RECDS2	03	015
LE	6,0324(,DISPQ)	V1+1+0	04	016
ME	6,0340(,DISPQ)	B2+1+0	04	017
STE	0,STORAGE+000		05	018
LE	0,0328	V2+1+0	05	019
ME	0,0336(,DISPQ)	B1+1+0	05	020
SER	6,0		04	021
LE	0,0260(,DISPQ)	V1-1+0	05	022
ME	0,0276(,DISPQ)	B2-1+0	05	023
SER	6,0		04	024
LE	0,0264(,DISPQ)	V2-1+0	05	025
ME	0,0272(,DISPQ)	B1-1+0	05	026
AER	6,0		04	027
ME	6,CONSTANT+016	RECDS2	04	028
SER	4,6		03	029
LE	6,CONSTANT+004	ETA	04	030
LE	0,0564(,DISPQ)	B2+0+1	05	031
AE	0,0052(,DISPQ)	B2+0-1	05	032
AE	0,0308(PLUSDZ,DISPQ)	B2+0+0	05	033
AE	0,0308(MINSDZ,DISPQ)	B2+0+0	05	034

TABLE V (cont.)

Operation code	Location	Variable	Level	
AE	0,0340(,DISPQ)	B2+1+0	05	035
AE	0,0276(,DISPQ)	B2-1+0	05	036
STE	2,STORAGE+004		06	037
LE	2,=E'+6.0000'+00'		06	038
ME	2,0308(,DISPQ)	B2+0+0	06	039
SER	0,2		05	040
MER	6,0		04	041
ME	6,CONSTANT+020	REDSSQ	04	042
AER	4,6		03	043
LE	2,STORAGE+004		02	044
MER	2,4		02	045
LE	0,STORAGE+000		01	046
AER	0,2		01	047
STE	0,0308(,DISPQ)	B2+0+0	01	048

Note: The code is efficient, with no extra address-manipulation instructions. Out of the 48 instructions, four are concerned with stack overflow. Note that comments are also generated automatically.

evaluated. These three quantities are defined in all the Symbolic ALGOL programs [1] by the variable name and by the global integer variables C1 and Q.

Because the generator program does not evaluate the physical expressions itself, it always has to leave the central point of the difference scheme undetermined and refer to it as Q, or else to point to the register in which the current value of Q will be kept during the subsequent execution run. These registers can be referred to directly (M15) or symbolically (DISPQ). The variable name, on the other hand, is known at the time of the generation run and so the numerical value of C1 (or, if the variable name is coded numerically, some arithmetic combination of that code number and the value of C1) can be output. Table V shows an example of IBM-360 ASSEMBLER code produced in this way.

Although in ALGOL or FORTRAN we could represent a scalar variable by an array with as many subscripts as there are dimensions n, and a vector variable by an (n+1)-dimensional array, this is often not desirable. The use of multidimensional arrays requires more registers and address calculations and slows down execution. One-dimensional arrays are therefore to be preferred for the representation of physical variables when the execution count is high.

Decisions must thus be made about whether to have separate arrays for different vector components of the same variable or, if not, in which order they are to be stored in a single array. Even for scalar variables it has to be decided in which way a three-dimensional set of values is going to be mapped into a one-dimensional array. These strategic decisions cannot all be made automatically and the best solution will depend on particular circumstances like the type of the mesh, the symmetries of the problem, the order in which the mesh is scanned, the total number of mesh points, the size and type of core and backing storage available and so on. In practical terms this means that even when the output language has been selected, small changes to the generator program will be necessary from problem to problem. In most cases, however, these changes will be confined to one procedure (SHIFT), and will consist in replacing one simple function of the component C1, the lattice displacement vector (K1, K2, K3), and the core storage 'distances' (DELTA1, DELTA2, DELTA3) between elements one lattice spacing apart by another simple function of the same integer variables. Since any SA/II program will in this way be mesh-dependent, we give a brief description of the mesh that the TRINITY program assumes [1].

4.2. Finite difference mesh used for TRINITY

The mesh is assumed to be Cartesian, of up to three dimensions, and equispaced. The numbers of mesh points in the x, y and z directions are denoted by PI, PJ and PK respectively, so that the total number of mesh points is SIZE = PI× PJ× PK. This includes six guard planes introduced to enable the same difference expressions to be used on the physical boundaries of the volume as inside it. The mesh points are counted in the x-direction, starting along the intersection of the first y- and the first z-plane numbered by integers I, J and K. The numerical relation is

$$Q = I + (J-1) \times PI + (K-1) \times PI \times PJ \tag{37}$$

For output in ALGOL or FORTRAN we choose separate arrays for each vector component in which the values are packed with increasing number Q. All the arrays are then of the same size equal to PI × PJ × PK and are functions only of one index Q.

This way of mapping the three-dimensional mesh on to one-dimensional arrays implies that two mesh points which are adjacent in the y-direction correspond to array elements PI storage locations apart and that, if the points are adjacent in the z-direction, the corresponding elements will be PI× PJ locations apart. The single mesh index changes by DELTA3 = PI×PJ if the shift is in the z-direction. It is the vector displacement operators EP and EM that cause, on a single application, a shift by one mesh point in the direction determined by the current value of C1. The number of shifts in the x, y and z directions is denoted in the program by the global integers K1, K2 and K3 from which, knowing PI and PJ, the corresponding change in Q can be calculated.

It is sometimes desirable to interleave the variable arrays so that the variable values at one mesh point Q are stored sequentially in the memory, e.g. in the order

$$\text{RHO, V1, V2, V3, B1, B2, B3, TEM} \tag{38}$$

followed by the same sequence for the next mesh point (Q+1) and so on. Adjacent values of the same variable are then, say, 32 bytes apart. This enables physical data planes to be transferred readily to and from the backing store as a single block and can be done either in ASSEMBLER code or, in FORTRAN, by the use of EQUIVALENCE statements. In all cases the SA/II program can readily be adapted to calculate the correct word or byte position in the store.

A similar situation exists when the core store is too small to hold the complete set of physical data. The largest size of mesh on which TRINITY has been run has $60\times80\times48$ points and requires two IBM-2301 drums to accommodate the 7 Mbytes of data. All the calculation then takes place within three sectors of a rotating quadrupole buffer, the fourth sector being used to transfer data to and from the core store in parallel with the calculation. The SHIFT procedure has been adapted so that it always refers to the correct areas of buffer storage.

5. ORTHOGONAL CURVILINEAR CO-ORDINATES

Symbolic vector algebra and analysis on a Cartesian lattice in SA/I have been described elsewhere [1] and few changes are required for SA/II. It may, however, be of interest to demonstrate how the method has been extended to generalized orthogonal curvilinear co-ordinates, with spherical polar co-ordinates as a special case.

The generalized definitions for divergence and curl can conveniently be written as

$$\operatorname{div}\vec{A} = \frac{1}{h_1 h_2 h_3}\sum_i \frac{\partial}{\partial q_i}\left(h_i^+ h_i^- A_i\right) \tag{39}$$

$$\operatorname{curl}\vec{A} = \sum_i \frac{\vec{a}_i}{h_i^+ h_i^-}\left(\frac{\partial\left(h_i^- A_i^-\right)}{\partial q_i^+} - \frac{\partial\left(h_i^+ A_i^+\right)}{\partial q_i^-}\right) \tag{40}$$

where $(\vec{a}_1, \vec{a}_2, \vec{a}_3)$ are unit vectors, $\vec{h} = (h_1, h_2, h_3)$ are scale factors, and +, - denote positive and negative cyclic rotation respectively.

These are translated into SA/II as

```
real procedure DIV(A); real A; DIV:=MULT(DOT(DEL(MULT(A, HPHM)), R2DQ), RH1H2H3);
```
(41)

```
real procedure CURL(A); real A;
 CURL:=MULT(DIFF(RP(MULT(DEL(RP(MULT(A, H, ))), R2DQ)),
                                RM(MULT(DEL(RM(MULT(A, H))), R2DQ))), RHPHM);
```
(42)

where

```
real procedure DEL(X); real X; DEL:=DIFF(EP(X), EM(X));
```
(43)

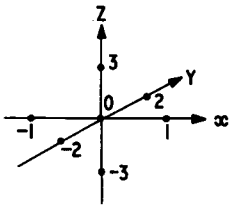


FIG. 4. Subsidiary mesh used to store the scale factors.

This leaves the real procedures to be defined

$$\begin{array}{lll} H & \rightarrow (h_1,\ h_2,\ h_3) & \text{(vector function)} \\ RH1H2H3 & \rightarrow 1/(h_1h_2h_3) & \text{(scalar function)} \\ RHPHM & \rightarrow 1/(h^+h^-) & \text{(vector function)} \\ R2DQ & \rightarrow 1/(2\cdot DQ) & \text{(vector)} \end{array} \qquad (44)$$

The mnemonic 'R' means reciprocal and signifies that division is avoided in the interests of efficiency, and for similar reasons $h_1h_2h_3$ and h^+h^- are defined separately instead of being constructed from $\vec{h}$.

When using the leapfrog scheme [1] we find it useful to store the scale factors and quantities that are constructed from them on a subsidiary mesh centred on the point Q, which may either contain 7 points if only one displacement $\pm\Delta q_i$ occurs at a time (Fig. 4) or 27 if they occur together. Then, for example, RHPHM becomes a real procedure which generates the code

$$RHPHM1[I],\ RHPHM2[I],\ RHPHM3[I] \qquad (45)$$

with I in the range (-3, 3) as in Table IV.

The generated target module is therefore still independent of the co-ordinate system although it does depend on the symmetry. To run the target program we must reload the variables on the subsidiary mesh whenever they alter. In spherical polars the scale factors are

$$h_r = 1,\ h_\theta = r,\ h_\varphi = r\sin\theta \qquad (46)$$

so that we can minimize the amount of recalculation by assigning the variables in order of $q_1 = \varphi$, $q_2 = \theta$, $q_3 = r$ with the innermost scan over q_1, although this is not necessarily the best choice on other grounds.

Some further improvements can be made if the co-ordinate system is specified at generation time; for example, h_r might be automatically suppressed since it is known to be unity.

6. THE PROBLEM PROGRAM AND THE GENERATOR PROGRAM

The problem program may be in any desired combination of languages, and by no means all of it need be constructed automatically. One convenient approach is to use FORTRAN for the more straightforward control sections

of the program, and for setting up core storage, and to use automatically-generated ASSEMBLER-code modules to handle the solution of the partial differential equations themselves as well as any complexities due to boundary conditions. Changes to the physics can then be made reliably and quickly, simply by generating a new module and linking it to the remainder of the program. The automatically-generated modules operate on variables in the COMMON storage area and communicate with the rest of the program either via COMMON or through argument lists. To switch to a new type of computer system, it is only necessary to alter the generator program so that it outputs the appropriate ASSEMBLER code.

An alternative approach is to use a combination of ALGOL 60 control program and ASSEMBLER-code modules. This approach has advantages if the local ALGOL system is sufficiently powerful, but in most cases FORTRAN is at present to be preferred.

One therefore starts by developing a strategy for the problem which takes into account its modular structure, the layout of core and backing storage, the communication between program modules, the choice of language and programming style for each module and so on. Those parts of the program whose execution count is low should be optimized according to considerations other than those of CPU efficiency; for example, they should be made intelligible or easy to write and to debug. Portability should also be planned right from the start, so that low-level modules designed for one computer system can be rapidly replaced by those written for another.

6.1. The generator program

An ASSEMBLER code replacement for an SA/I module of the prototype program is constructed by incorporating the corresponding SA/II module into the generator program which is then run (Fig. 5). We visualize a series of related physical problems, run either on one computer at a single laboratory or on a range of different computers at several laboratories. If the generator program has been constructed properly, only a small part of the work has to be repeated for each new situation. This saves time and makes it easier for one person to understand a range of programs, since they share a family resemblance like members of related biological species.

The generator program ought to be highly modular since there are a number of requirements which may well need to be changed independently. Examples are:

(a) Computer system on which the code is generated
  (i) ALGOL character representation
  (ii) System output procedures
(b) Computer system for which the code is generated
  (i) Character representation
  (ii) Language
  (iii) System facilities
(c) Number of dimensions and co-ordinate system
(d) Mesh and storage layout
(e) Symbols used for variables and constants
(f) Numerical methods and difference schemes
(g) Physical equations
(h) Boundary conditions
(i) Physical coefficients (e.g. thermal conductivity)

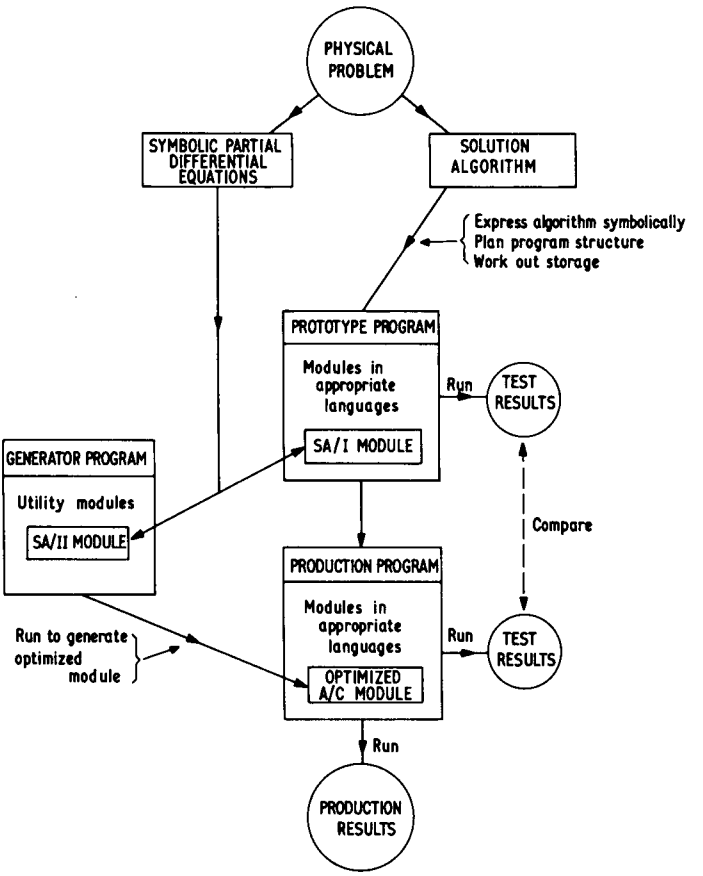


FIG.5. Programming strategy.
The method of solution is first tried out by writing a prototype program which is used to produce sets of test results. Modules of the prototype are next reconstructed either by hand or with the SA/II generator program to make them more efficient. Test results from the production program are then compared with those from the original tests to make sure that the updating has been carried out correctly.

Figure 6 shows the relation between the modules which have been developed so far, while Table VI provides a list. In Section 7 we shall go through this list briefly, indicating what the various modules do. A detailed description together with a program listing and test runs will be published elsewhere [6].

6.2. Relation to compilers and macro-generators

In mathematics or theoretical physics it is always possible for an author to devise a new notation or extension to the accepted 'language' and, having defined it for the reader, to use it throughout a paper or a course of research. This flexibility has been largely unavailable in computing science,

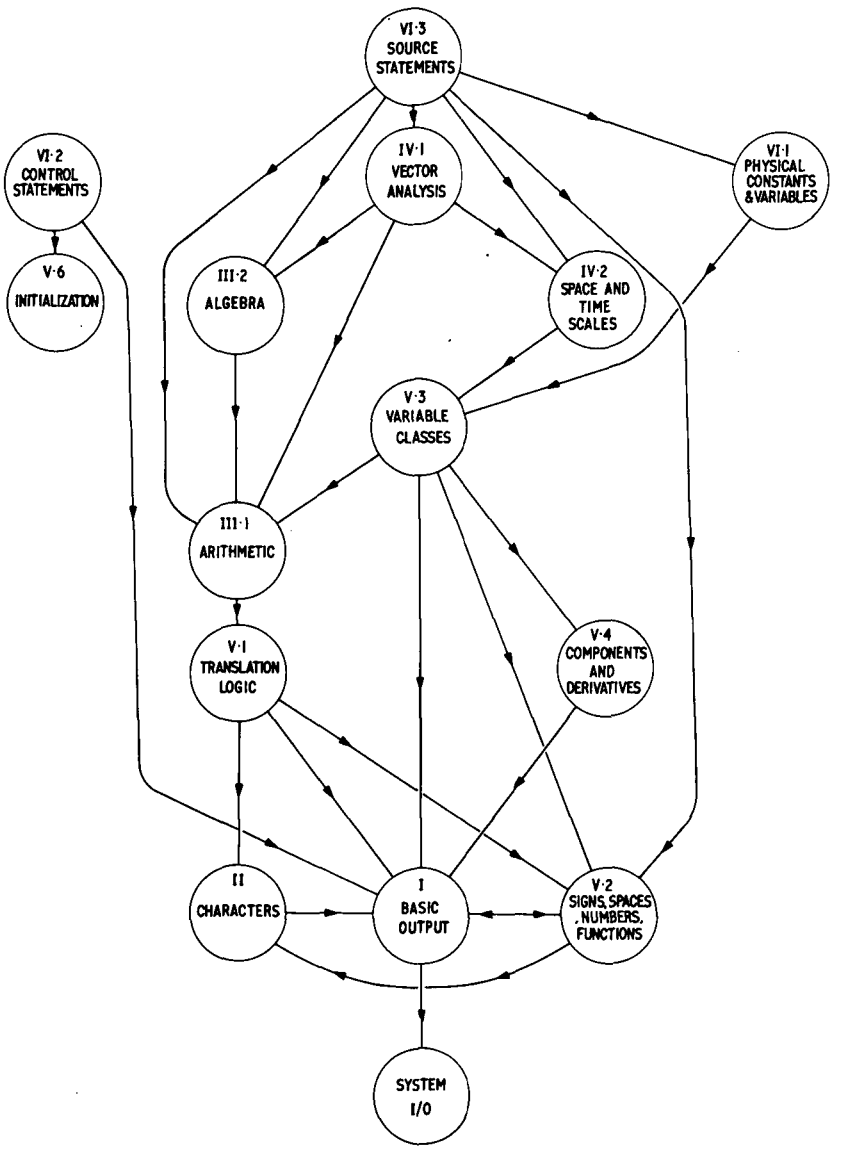


FIG. 6. Relation between the modules of the generator program.
An arrow indicates that one module makes use of procedures belonging to the other. Note that all output is channelled through a single short module BASIC OUTPUT so that only this module has to be changed to run the program on another computer.

where it has been customary to use rather standardized and limited languages such as FORTRAN, ALGOL or PL/I which are constructed either by manufacturers or by international committees and translated by compilers which are expensive and time-consuming to write and to maintain. The only degree of freedom left to the individual has been the ability to define sets of library subroutines or procedures which in effect become additions to the standard notation. There is no limit on the extensions which can be achieved in this way, but programs tend to run slowly if they make considerable use of procedure calls, as in SA/I.

Macro-processors such as STAGE 2 [4] enable any string of symbols to be given a meaning. The user is free to define sets of macros which convert any string into any other string and eventually into the code of some high- or low-level language whose efficiency depends solely on his own ingenuity. Thus the full flexibility of mathematics is achieved provided that the character set is wide enough.

SA/II appears to lie somewhere in between. The formal structure of the input string is constrained by the syntax of ALGOL so that there is usually an excessive number of brackets, as in Table III, but the manipulations that can be carried out are quite general and it is remarkably easy for the individual user to make alterations or additions to the 'language' by changing the basic procedures of the generator program. An SA/II generator program is also quite short, usually only a few hundred ALGOL statements. Thus we have effectively at our disposal an ultra-high-level 'language' compiler which is difference-scheme and problem-dependent but is also easily changed by any user.

7. STRUCTURE OF THE GENERATOR PROGRAM

The current version divides logically into 6 main modules and 13 submodules as shown in Table VI.

I. BASIC OUTPUT. In transferring the generator program to a new computer system the first task is to rewrite this module, which forms a link to the standard ALGOL output procedures of the computer on which the generator is being run. (This need not of course be the same as the computer for which the optimized code is being produced.) The ICL KDF9 version occupies about 25 cards and it can usually be rewritten for another system in a few hours. The module defines the output channel and sets up standard formats, and contains procedures which enable the output to be manipulated in a straightforward system-independent way, e.g.

BLANKS(N)	Output N blank spaces
LINE	Start a new line
OUTNUM(L, F, X)	Output the value of an arithmetic expression X in format F, length L
TEXT(L, S)	Output a string S of length L.[2]

The complete set is enough to generate code and comments in any language.

[2] The simpler procedure PRINT discussed in Section 2 did not contain the parameter L, which helps the output procedures to organize the layout of the line.

TABLE VI. LIST OF GENERATOR PROGRAM MODULES AND SUBMODULES

I.	BASIC OUTPUT
II.	CHARACTERS
III.	MATHEMATICS A
	III.1. ARITHMETIC
	III.2. ALGEBRA
IV.	MATHEMATICS B
	IV.1. CARTESIAN ANALYSIS LEAPFROG
	IV.2. SPACE AND TIME SCALES
V.	OUTPUT ORGANIZATION
	V.1. TRANSLATION LOGIC
	V.2. SIGNS, SPACES, NUMBERS AND FUNCTIONS
	V.3. VARIABLE CLASSES
	V.4. COMPONENTS AND DERIVATIVES
	V.5. COMMANDS AND REGISTER CHECKS
	V.6. INITIALIZATION
VI.	PROBLEM DEFINITION
	VI.1. PHYSICAL CONSTANTS AND VARIABLES
	VI.2. CONTROL STATEMENTS
	VI.3. SOURCE STATEMENTS

II. CHARACTERS. This also contains no submodules. It is made up of a dozen or so simple statement procedures which enable one to refer to symbols like (, + ; = etc. by symbolic names: OPENB, COMMA, PLUS, SEMICOLON, EQUALS. Although this somewhat slows down code generation, it makes the program much easier to read and quicker to write, particularly in view of the awkward way in which string quotes are often represented in ALGOL. Examples are:

```
procedure EQUALS; TEXT(1,'=');
procedure CLOSEB; TEXT(1,')');
procedure OPENSB; TEXT(1,'[');
procedure MINUS; TEXT(1,'-');
```
(47)

The first argument gives the length of the string, in this case 1.

III. MATHEMATICS A. The first mathematics module comprises the submodules

ARITHMETIC

ALGEBRA

The first submodule contains a set of procedures

SUM(X, Y)	MULT21(X, Y)
DIFF(X, Y)	QUOT(X, Y)
MULT(X, Y)	SUM3(X, Y, Z)

which are all dealt with by a single fairly complex procedure TRIPOP (triple operator) explained elsewhere [5], e.g.

real procedure SUM3(X, Y, Z); real X, Y, Z; SUM3 := TRIPOP(5, X, Y, Z, 1, 1); (48)

The first argument of TRIPOP is an operation code, the second to fourth arguments are the operands, and the last two specify the first or second indexes (in the case of a second-rank tensor). Procedure SUM3 is convenient when handling 3D scalar products and divergences while MULT21 is used for handling tensor contractions. The others deal with the arithmetic operations +, -, x, /.

The ALGEBRA submodule contains the rotation operators RP and RM and the vector-algebraic operators DOT and CROSS.

IV. MATHEMATICS B. In the simplest version, the second mathematics module contains the two submodules

1. CARTESIAN ANALYSIS LEAPFROG
2. SPACE AND TIME SCALES

As its name implies, the first of these submodules depends on the mesh geometry and on the difference schemes used (although not on the physical problem or on the computer system). The procedures EP, EM, DEL, GRAD, DIV, CURL, SAV, DELSQ are fairly direct translations of the SA/I versions already published [1]. The other submodule deals with the constants Δt, $2\Delta S$ and $(\Delta S)^2$ and can be also readily extended.

V. OUTPUT ORGANIZATION. This module depends on the output language. It may contain up to six submodules of which number 5 is omitted in the ALGOL target version.

V.1. TRANSLATION LOGIC. Contains most of the logic needed to generate the output code and to eliminate expressions which are identically zero, as well as unnecessary brackets. It therefore deserves a more detailed description which is given in Ref. [6].

V.2. SIGNS, SPACES, NUMBERS AND FUNCTIONS. A number of standard utilities are provided here, some of which depend on the output language or format. For example, a line overflow in FORTRAN requires that a continuation symbol should be punched in column 6; overflow in ASSEMBLER language is handled in a different way, while at the end of an ALGOL statement a semicolon is required. Signs, integer and real numbers and elementary mathematical functions are also provided for. The most important procedure is EQUATE which is discussed in Section 3.

V.3. VARIABLE CLASSES. Contains procedures which deal with constants, scalars, vectors and tensors, of which (32) is an example.

V.4. COMPONENTS AND DERIVATIVES. Contains the procedures COMPONENT and SHIFT which generate the code for referencing storage locations, including any subsidiary calculations that are needed.

V.5. COMMANDS AND REGISTER CHECKS. (IBM-360 ASSEMBLER code only). Contains procedures REGISTER REGISTER, REGISTER STORAGE which issue IBM-360 instruction mnemonics, and STORE IF OVERFLOW which determines whether or not the required register is already in use, if so copying it into a reserved location in core store.

V.6. INITIALIZATION. Contains a procedure START which initializes the variable of the generator program.

VI. PROBLEM DEFINITION. This module is provided by the user and consists of three submodules which have been kept as simple and as close to the physics as possible:

1. PHYSICAL CONSTANTS AND VARIABLES
2. CONTROL STATEMENTS
3. SOURCE STATEMENTS

Table III gives the source statements for TRINITY while (31) is an example of a variable declaration. Typical initialization statements are

$$\text{NDIM} := 3;\ \text{PI} := \text{PJ} := \text{PK} := 8; \tag{49}$$

(use an $8 \times 8 \times 8$ mesh in three dimensions).

8. CONCLUDING REMARKS

A satisfactory solution to the slowness of programs written in Symbolic ALGOL I has been found. Using a style known as Symbolic ALGOL II, it has been possible to translate finite difference equations automatically into fully explicit codes in a number of target languages. The best of these codes are fully competitive in speed with hand-optimized FORTRAN and are fast enough to make the solution of time-dependent magnetohydrodynamic equations in three space dimensions a feasible proposition. In a recent exercise, a 3D plasma code in rotating spherical co-ordinates was designed and written in about four days using the SA/II method. The same translator program can be used for other systems of fluid equations and also for problems in two dimensions. Although the advantages of the method are smaller when applied to simpler problems, repetition of effort can still be avoided. More important, the use of well-tested procedures reduces programming errors, in particular those of a numerical nature which are often impossible to detect except experimentally through a comparison with another calculation. Though at first sight trivial, this may prove to be one of the important attractions of the method.

The underlying principle is that instead of writing a problem program by hand, one constructs a generator problem which writes it automatically.

Because this generator program is built up from prefabricated modules and is also highly symbolic it can be developed and altered very quickly.

In essence we are using ALGOL as a powerful macro-generator which is capable of substituting one expression into another as well as performing many subsidiary calculations. Because a value is associated with each substitution, extra information can be carried along which allows some optimization to be done. A further extension might be to relate this value to a generalized variable type (e.g. logical, integer, real, complex, quaternion, matrix or whatever), so that any necessary conversions can be carried out and the basic operators SUM, DIFF, MULT, etc. can be interpreted in the appropriate way in each case. This is close to the procedure followed in mathematics, which allows operators such as +, -, × to be freely generalized to new object classes. Other interesting possibilities are to apply the SA/II technique to other kinds of program such as operating systems and compilers, and to other types of computer such as the CDC STAR.

The features of ALGOL 60 that appear to be necessary or useful for this kind of work are [8, 9]:

(a) Call by name. Needed for the symbolic substitution of one expression into another.

(b) Parameterless typed procedures. Just as in mathematics, a function need have no explicitly-indicated arguments (FORTRAN does not allow this).

(c) English-language identifiers of any length, with blanks ignored. Can be used to make programs more intelligible and to avoid bulky comments.

(d) Elimination of the unnecessary word 'CALL'.

(e) Ability to have several statements on one line. Both (d) and (e) make programs more concise and attractive.

(f) No overhead on procedure declarations. Often these declarations are only one card long, and one line in the compiler listing, instead of several pages as in FORTRAN.

(g) Block structure for variable scopes. Global variables can be passed into a procedure implicitly without the need for a bulky COMMON deck or argument lists.

(h) Recursion. Typed procedures can be substituted into one another without restriction as required by mathematical physics.

(i) Side effects. Available also in other languages, but mentioned here as being crucial to the whole method.

ACKNOWLEDGEMENTS

The early planning of SA/II was carried out in collaboration with Dr. J.P. Boris. We should like to thank Dr. F. Hertweck and his colleagues at the Institut für Plasmaphysik, Garching, Federal Republic of Germany, for making available to us the excellent facilities of the IBM-360/91 Computing Centre at the Institute. We should also like to thank Mr. R.S. Peckover for many discussions on Symbolic ALGOL, and Dr. N.K. Winsor for providing an improved compiler.

REFERENCES

[1] ROBERTS, K.V., BORIS, J.P., The solution of partial differential equations using a symbolic style of ALGOL, J. comput. Phys. 8 (1971) 83.

[2] ROBERTS, K.V., PECKOVER, R.S., "Symbolic programming for plasma physicists", in Proc. Fourth Conf. Numerical Simulation of Plasmas, U.S. Naval Research Laboratory, Washington, D.C., November 1970, Office of Naval Research, Department of the Navy (July 1971) 165.

[3] KUO-PETRAVIĆ, G., PETRAVIĆ, M., ROBERTS, K.V., The Translation of Symbolic ALGOL I to Symbolic ALGOL II by the STAGE 2 Macro-Processor, Culham Laboratory preprint CLM-P 275.

[4] WAITE, W.M., The mobile programming system STAGE 2, Communs Ass. comput. Mach. 13 (1970) 415.

[5] KUO-PETRAVIĆ, G., PETRAVIĆ, M., ROBERTS, K.V., The translation of Symbolic ALGOL I to Symbolic ALGOL II using ALGOL 60, to be submitted for publication in Comput. Phys. Communs.

[6] PETRAVIĆ, M., KUO-PETRAVIĆ, G., ROBERTS, K.V., "Automatic optimization of Symbolic ALGOL programs, II. Code generation", to be published.

[7] PETRAVIĆ, M., KUO-PETRAVIĆ, G., ROBERTS, K.V., The Symbolic ALGOL II generator program, to be submitted for publication in Comput. Phys. Communs.

[8] HIGMAN, B., A Comparative Study of Programming Languages, McDonald/Elsevier Computer Monographs (1967).

[9] BAUMANN, R., FELICIANO, M., BAUER, F.L., SAMUELSON, K., Introduction to ALGOL, Prentice-Hall (1964).

THE DELPHI-SPEAKEASY SYSTEM*

S. COHEN

Argonne National Laboratory,
Argonne, Ill., United States of America

Abstract

THE DELPHI-SPEAKEASY SYSTEM.

This paper describes a system of modular computer programs designed to provide investigators in the physical sciences with more convenient access to the power of a modern computer. Constructed within the framework of a special storage technique, the routines of the system resemble true mathematical operators. A new problem is programmed by bringing together the needed operators with a simple directive program. To make the construction of this control program even more direct, a new interpretative computer language called SPEAKEASY has been designed. The power of this language to provide quick results has already proved its worth. The growth of this language is greatly enhanced by a recently added facility, used at execution time, by which it can be dynamically linked to user-contributed routines from logically separate libraries.

INTRODUCTION

The growth in capabilities of hardware and software systems in modern computers has tended to an increased isolation of the normal research scientist. To properly use a computer, the scientist is expected to know ever more about the detailed structure of a tool which should be an aid in his research. This burden of additional knowledge can only be met by sacrificing time which might better be spent in his own research discipline.

Historically each new computer study is carried out in much the same way. The entire problem is analysed in the context of the available computer system. Then a large efficient computer program is constructed to carry out the specific calculation. Existing subroutines are used primarily to evaluate special functions needed during the calculation. This seemingly straightforward approach to computer studies is common practice today and is very little advanced over that used several years ago. Each project in effect starts anew and involves the time-consuming and frustrating process of debugging a massive and detailed computer program. Such an approach is hardly able to meet the needs of a scientist who knows little of the advanced facilities available in the computer and has only a transient interest in the details of program construction. Nor is such an approach likely to evolve into one that can exploit the increased capability of new machines.

The DELPHI-SPEAKEASY system [1] is a project aimed at providing researchers in the physical sciences with a more direct approach to the power of a modern computer. It does so by providing users with a highly modular library of interlinkable routines that can be joined together into an operational program. Built into the structure of this system is the

* Work performed under the auspices of the US Atomic Energy Commission.

concept of an evolutionary increase in its capability for application to new problems. Each new module added to the system enhances the overall power of the system. Ease of use is a prime consideration in all decisions made in specifying the system.

A systematic development such as this provides an interface between a growing computer complex and a relatively untrained user. Modules of the system can be designed to exploit the power of the machine without requiring the user to acquire a detailed understanding of their operation. The long-term growth capability cannot be overemphasized. Each problem successfully carried out within the system provides future users with new capabilities. It therefore becomes easier and easier to carry out new problems. Proper design specifications of modules also ensure that calculations are carried out efficiently.

NAMED-STORAGE MANIPULATORS

Each module of the DELPHI system has been designed as a "clean" subroutine, i. e. one intended to resemble a mathematical operator as closely as possible. A special storage technique called NAMED STORAGE [2] was developed to aid in their construction. Sets of data, each with its associated structure-defining information, are stored in a special dynamic-storage array and assigned a reference name. The task of the NAMED-STORAGE subsystem is to retain the information assigned to it and to make the information available to the executing program on demand. All reference to such data is by its assigned name.

Each set of data stored in NAMED STORAGE carries information about the kind of data (e. g. floating point, integer, alphabetic or the like) in addition to the structure of the defined object (e. g. 10×10 matrix, 300-component array, a scalar or some other). It is therefore possible to construct general routines to operate on named quantities to produce new named results. Such routines are designed for operations on sets of data rather than on individual elements. Data-directed routines combined with the dynamic-storage scheme provide powerful execution-time capabilities.

Each routine can be written to carry out a class of mathematical algorithms. As an example, the FORTRAN calling sequence for a routine for multiplication could be

```
CALL MULT ('A', 'B', 'C')
```

where literal variables are indicated by enclosing them in apostrophes. The routine MULT can be designed to include all that is normally meant by the operation multiply. The statement above means multiply the object called A by the object called B and create a resultant called C. If A is a 2×3 real matrix and B is a 3×5 real matrix, then this subroutine should define a real 2×5 matrix named C with elements arrived at by the usual rules of matrix algebra. The operation of the routine MULT is data-directed in much the way that the mathematical operation of multiply is dependent on its operands. The operation of multiply can be fully and completely programmed once and for all within this single subroutine.

Since complete information about the structure of the operands is available, extensive checking is feasible. In the above example, an attempt to multiply a 2×3 matrix by a 4×5 matrix results in the calling of a standard error routine.

Routines such as the one described above are referred to as NAMED-STORAGE MANIPULATORS. Manipulators, since they are written only once, can be written with extreme concern about efficiency and completeness.

Since manipulators are completely self-contained, one can write new manipulators that use other manipulators internally. For greater efficiency, these composite manipulators can later be replaced by more compactly written routines. The evolutionary process of system development will eventually lead to a complete set of efficient routines.

A large number of manipulators of the type described above make up the DELPHI system. Each is a mathematical operator – a programmed algorithmic specification for a specific task. The value of such a set of routines should be apparent. In programming a new problem, one is able to draw freely on a library of well-tested operators. The task of writing a new program is greatly eased since most of the trivial parts of the computations are carried out within the standard packages. The user can concentrate on the logic of the calculation at a level closer to that of the problem of interest. Because of the extensive checking facilities built into the manipulator routines, it is also unlikely that an error will pass undetected.

Each manipulator is in some sense trivial. It carries out a single straightforward operation. Often the operation is so simple that one questions the need for a special routine for this purpose. It should be understood that the library of manipulators is of value only insofar as it is complete. The manipulators can be viewed as an extension of the simple built-in functions of most systems. One rarely thinks about the operations inside such routines as SIN, ABS, etc. when writing new programs. The intent of the DELPHI system is to provide a much larger library of operations to be used in a similar way.

SPEAKEASY

The existence of a complete library of manipulators means that a program applying them consists entirely of calls to manipulators. A section of such a program might be

FORTRAN statement	Meaning
Call MULT ('A', 'A', 'D')	D = A * A
Call MULT ('A', 'B', 'C')	C = A * B
Call ADD ('C', 'D', 'E')	E = C + D
Call PRINT ('A', 'B', 'E')	PRINT (A, B, E)

The meaning of each statement is shown on the right. A computer program could be written to accept statements such as the righ-hand ones and to generate the calls shown on the left. Such a program has been written and has resulted in the creation of a new computer language called SPEAKEASY.

```
SPEAKEASY 3C 10:39 AM 10/12/71

INPUT...M=MATRIX(3,3:1 2 3 4 5 9.5 10 2)
INPUT...PRINT M

      M (A 3 BY 3 MATRIX)
      1     2     3
      4     5     9.5
      10    2     0

INPUT...M=M+TRANSPOSE(M)
INPUT...T=1/M
INPUT...PRINT M T

      M (A 3 BY 3 MATRIX)
      2      6      13
      6      10     11.5
      13     11.5   0

      T (A 3 BY 3 MATRIX)
      .82399   -.93146    .38006
     -.93146    1.053    -.34268
      .38006   -.34268    .099688

INPUT...M*T

    M*T  (A 3 BY 3 MATRIX)
      1  0  0
      0  1  0
      0  0  1

INPUT...EIGENVALUES(M)

    EIGENVALUES(M)  (A VECTOR WITH 3 COMPONENTS)
      24.621  .49696 -13.118

INPUT...EIGENVECTORS(M)

    EIGENVECTORS(M)  (A 3 BY 3 MATRIX)
     -.50014 -.65278 -.56898
      .64579 -.71891  .25714
      .5769   .23884 -.78111
```

FIG. 1. The first part of a sample SPEAKEASY run illustrating one of its capabilities in matrix algebra. Note in particular the freedom from format specifications.

The power of this language is that it provides a quick and concise means of connecting the manipulative routines into a workable program.

SPEAKEASY is operational (in several versions) on computers of the IBM-360 series. Great effort has been expended in providing user-oriented decisions throughout the language. The notation itself closely resembles that of usual mathematics and is therefore easily and quickly learned. The intent is always to provide the scientist with the answers to his problems quickly and in a form as easily read as possible.

The result of having this powerful and easily used language available is already apparent. Major problems can and have been solved in a single day. This should not be surprising; a modern computer is after all a very fast and sophisticated device. Properly used, the vast storehouse of information available to such a machine can be harnessed to the needs of

```
INPUT...A=ARRAY(3,3:1 2 3 4 5 9.5 10 2)
INPUT...PRINT A

      A (A 3 BY 3 ARRAY)
      1     2     3
      4     5     9.5
      10    2     0

INPUT...A=A+TRANSPOSE(A)
INPUT...R=1/A
    WARNING--
    IN STAT. " R=1/A  " DIVISION BY ZERO.
INPUT...PRINT A R

      A (A 3 BY 3 ARRAY)
      2      6      13
      6      10     11.5
      13     11.5   0

      R (A 3 BY 3 ARRAY)
      .5          .16667    .076923
      .16667      .1        .086957
      .076923     .086957   0

INPUT...A*R

    A*R  (A 3 BY 3 ARRAY)
      1  1  1
      1  1  1
      1  1  0

INPUT...SUM A
    SUM A  =  73
INPUT...AVERAGE A
    AVERAGE A  =  8.1111
```

FIG. 2. The part of the SPEAKEASY run in which the array algebra is illustrated. Note the differences between the algebra of arrays and that of matrices (Fig. 1).

specialists such as research scientists. After all, the demands of the scientist usually are rather clearly formulated requests well within the capacity of the machines. The difficulty encountered in using most computer languages is that the scientist is forced to spend most of his effort on the mundane parts of the calculation – the trivial loops, the input and output formats, the choice of dimension statements, etc. Errors are more likely in these microscopic details of programs because the scientist is probably thinking more about the physical problem and not about the artificial steps he must take to solve it.

SPEAKEASY is a much more direct computer language. It bridges the gap between a scientist and the computer by forcing the computer to do much of the detailed work and by letting the scientist concentrate on the real scientific problem. The chances for error are greatly reduced in this language because of its concise and natural form.

Figures 1 to 4 are a sample run of the current production version of SPEAKEASY. Both the input and output are shown here (the input information is labelled on the left by INPUT ...). Realize that this is the complete run. Aside from standard job-control cards, the dozen or so cards indicated in these figures are all that were needed to produce the results.

```
INPUT...X=GRID(0,10,(.2))
INPUT...Y=SIN(X)*EXP(-X)
INPUT...TABULATE(X,Y)

      X    Y            X    Y            X    Y            X    Y

      0    0            2.6   .038288     5.2 -.0048736     7.8   4.0914E-4
      .2   .16266       2.8   .020371     5.4 -.0034903     8     3.3189E-4
      .4   .26103       3     .007026     5.6 -.0023343     8.2   2.5838E-4
      .6   .30988       3.2 -.0023795     5.8 -.0014066     8.4   1.9217E-4
      .8   .32233       3.4 -.0085282     6    -6.926E-4    8.6   1.3521E-4
      1    .30956       3.6 -.012091      6.2 -1.6862E-4    8.8   8.8166E-5
      1.2  .28072       3.8 -.013688      6.4   1.9365E-4   9     5.0859E-5
      1.4  .24301       4   -.013861      6.6   4.2381E-4   9.2   2.2521E-5
      1.6  .20181       4.2 -.01307       6.8   5.5033E-4   9.4   2.0495E-6
      1.8  .16098       4.4 -.011683      7     5.9909E-4   9.6 -1.1807E-5
      2    .12306       4.6 -.0099884     7.2   5.9254E-4   9.8 -2.0322E-5
      2.2  .089584      4.8 -.0081982     7.4   5.4934E-4   10  -2.4699E-5
      2.4  .061277      5   -.0064612     7.6   4.844E-4

INPUT...TOTALINTEGRAL(Y:X)
    TOTALINTEGRAL(Y:X)  =  .4967
INPUT...AVERAGE Y
    AVERAGE Y  =  .048696
INPUT...AVERAGE X
    AVERAGE X  =  5
INPUT...WHEREVER (ABS(Y) LT .001) Y=0
INPUT...TABULATE (X Y)

      X    Y            X    Y            X    Y            X    Y

      0    0            2.6   .038288     5.2 -.0048736     7.8   0
      .2   .16266       2.8   .020371     5.4 -.0034903     8     0
      .4   .26103       3     .007026     5.6 -.0023343     8.2   0
      .6   .30988       3.2 -.0023795     5.8 -.0014066     8.4   0
      .8   .32233       3.4 -.0085282     6     0           8.6   0
      1    .30956       3.6 -.012091      6.2   0           8.8   0
      1.2  .28072       3.8 -.013688      6.4   0           9     0
      1.4  .24301       4   -.013861      6.6   0           9.2   0
      1.6  .20181       4.2 -.01307       6.8   0           9.4   0
      1.8  .16098       4.4 -.011683      7     0           9.6   0
      2    .12306       4.6 -.0099884     7.2   0           9.8   0
      2.2  .089584      4.8 -.0081982     7.4   0           10    0
      2.4  .061277      5   -.0064612     7.6   0
```

FIG. 3. Part of a SPEAKEASY run showing the tabular capabilities and some of the operators on functions of one variable.

LINKULES

The power of a language such as SPEAKEASY resides in its ability to meet the needs of users. As long as modifications of the processor remain totally under the control of the originators, growth is limited. A rather recent addition to the processor has provided capabilities that should eliminate such constraints.

During processing, if SPEAKEASY encounters a word that is not defined in either the user's program or in the standard SPEAKEASY processor, several added libraries are searched. These libraries contain compiled subprograms and represent additional manipulative routines

```
INPUT...VOCABULARY
    *** STANDARD WORDS ***
    AFAM      MFAM      VFAM      RECLASS   VECTOR    MATRIX    SYMMAT    ASYMMAT
    ARRAY     ARRAY2D   DIAGMAT   GRID      INTEGERS  ROWMAT    COLMAT    ***
    UNITMAT   VEC       MAT       SMAT      AMAT      A1D       A2D       DMAT
    VARIABLE  INTS      ROWARRAY  COLARRAY  **        UMAT      GOTO      IF
    EXECUTE   WHERE     WHEREVER  LOC       LOCS      SIGNIFIC  SETNULL   SETINFIN
    ACCURACY  SPACE     COLWIDTH  DOMAIN    MARGINS   ONERROR   AUTOPRIN  FREE
    TIME      READ      WRITE     PUNCH     EIGENVAL  EIGENVEC  NORM      DIAGELS
    TRACE     DET       INVERSE   TRANSP    TRANSPOS  SIN       COS       TAN
    COT       ASIN      ACOS      ATAN      ACOT      LOG       EXP       SQRT
    ABS       SIGN      SINH      COSH      GAMMA     LOGGAMMA  REALPART  IMAGPART
    INTPART   FRACPART  CONJUGAT  REAL      IMAG      WHOLE     FRAC      CONJ
    RANKED    RANKER    NOROOTS   TOTALINT  INTEGRAL  DERIV     ROOTS     INTERP
    ORDERED   ORDER     NOZEROS   TOTINT    INTEG     DERIVATI  ZEROS     INTERPOL
    MIN       ROWMIN    COLMIN    MAX       ROWMAX    COLMAX    NOELS     NOROWS
    NOCOLS    LOCMIN    LOCMAX    SUM       SUMROWS   SUMCOLS   SUMSQ     SUMSQROW
    SUMSQCOL  PROD      PRODROWS  PRODCOLS  LENGTH
    *** LINKULES***
    AVERAGE   BATFLAP   BESSEL    CLEBSCH   COULOMB   CUMPROD   CUMSUM    EIGENSYS
    ERF       ERFC      FCPOT     FCTRPOT   FS3SIDES  GEIGEN    HOWBOUND  LOWERTRI
    MASSAMU   MASSMEV   MAXOF     MAXOFCOL  MAXOFROW  MELD      MINOF     MINOFCOL
    MINOFROW  NEUMANN   NINEJ     OBJECT    GRAPH     QLEG      RACAH     RANDOM
    REAL8     REIDPOT   RHOMBUS   RKCOUL    SETGAUSS  SETJACOB  SETLAGUE  SETLEGEN
    SIMEQ     SIXJ      SPHBES    SPHBESN   SSDIFFEQ  SSDIFF2   SSGRID    THREEJ
    TRIANGLE  UPPERTRI  VDISTORT  XLOC
     ***  AVAILABLE DOCUMENTS ***
    ARGUMENT  BESSEL    COULOMB   CUMPROD   CUMSUM    EIGENSYS  ERF       ERFC
    GEIGEN    HENCEFOR  HOWBOUND  INDEX     LOWERTRI  MASSAMU   MASSMEV   MAXOF
    MELD      MINOF     NEUMANN   NOECHO    NOLIST    OBJECT    QLEG      RANDOM
    SETGAUSS  SETJACOB  SETLAGUE  SIMEQ     SPHBES    SPHBESN   UPPERTRI
```

FIG.4. The current SPEAKEASY vocabulary, which is available on demand from the processor. This figure shows the standard words and linkules available at Argonne. Note the variety of the words in the vocabulary.

which are not in the basic program. If the reference name of such a subprogram is identical to the unknown word, then that subprogram is loaded and execution control is transferred to it. A completion control is returned to the SPEAKEASY processor.

Realize that the compilation of each such subprogram, referred to as a LINKULE, is entirely independent of the SPEAKEASY processor. LINKULES may be added, deleted and replaced without involving the basic processor at all. Each user or group of users may have a private set of linkules. New operations can be added to the language by adding them to the communal linkule library. Growth in power of the SPEAKEASY language now comes mainly from the user community and has become more rapid and varied than it would otherwise have been.

The concept of dynamic linking is so powerful that most of the standard operations of SPEAKEASY are now being transferred to linkule libraries. In this way the size of the basic SPEAKEASY processor can be decreased to the benefit of all users.

AVAILABILITY

The production versions of SPEAKEASY are available for use in installations having an IBM-360 system with Model-50 or larger machines. Special interactive versions for IBM-2250 display stations are also available. Copies of these programs and descriptive manuals for the language are available from the author.

REFERENCES

[1] COHEN, S., The DELPHI-SPEAKEASY system, Comput. Phys. Communs 2 (1971) 1.
[2] COHEN, S., Named Storage with Manipulator Routines, Argonne National Laboratory Rep. No. ANL-7021 (1964).

INTRODUCTION TO LISP

D. LURIE*
CERN, Geneva

Abstract

INTRODUCTION TO LISP.

1. Preliminary; 2. The concept of a list; 3. Machine representation of lists; 4. S-expressions; 5. Basic LISP functions; 6. LISP programming; 7. LISP program structure – property lists; 8. A short LISP program for symbolic differentiation.

1. Preliminary

LISP (an acronym for LIST PROCESSOR) is a programming language designed for the manipulation of a particular type of data structure known as a "list" or more generally an "S-expression". The language was developed about ten years ago by a group at M.I.T. led by McCarthy [1]. LISP differs from FORTRAN or ALGOL in that it is designed for symbol manipulation but not – or at least not primarily – for numerical computation. Its key areas of application are those which call for large amounts of symbolic computation. In physics these include quantum field theory and group theory [2-5]. Other domains of application in which LISP has proved its usefulness are differential and integral calculus, algebra (including non-commutative algebras), mathematical logic, game theory, graph theory, electrical circuit analysis and artificial intelligence.

Our aim is to introduce the reader to the basic concepts and programming techniques of LISP. For this purpose, the original form of the language – called "LISP 1.5" – is not too convenient. The syntax of LISP 1.5 differs considerably from that of other higher-level programming languages and the beginner is therefore faced with a double burden – mastering the basic techniques of list programming and at the same time coping with an unfamiliar syntax. To avoid this we shall use, instead of LISP 1.5, a recently developed LISP-like language whose syntax is very close to that of ALGOL. This language, called REDUCE [6], is the programming language for a general system for algebraic problem solving developed by Hearn [7-9]. In the text, we shall refer to the language simply as LISP or REDUCE[1].

The organization of this introduction is as follows. In Sections 2, 3 and 4, we introduce the concept of a list and an S-expression and discuss how these structures are represented within the computer. In Section 5, we present the five basic LISP functions CAR, CDR, CONS, ATOM and EQ. Section 6 is devoted to the techniques of LISP programming: several simple LISP programs are explained and the concept of a "lambda expression" is introduced. In Section 7, we summarize some general features of LISP

---

* Permanent address: Department of Physics, Technion-Israel Institute of Technology, Haifa, Israel.

[1] The REDUCE system itself is written in LISP 1.5 so that REDUCE can be regarded as a third-level language.

program structure and discuss the manipulation of "property lists". The final section, Section 8, describes a simple LISP program to carry out symbolic differentiation.

There is far more to LISP than can be covered here. The present introduction will have served its purpose if it enables the reader to turn to the more technically oriented literature on the subject and begin writing his own LISP programs. The primary source of information on LISP 1.5 is the "LISP 1.5 Programmer's Manual" [1] but Refs [10-12] are also extremely useful. For further information on REDUCE the reader should consult Refs [6-9].

2. The concept of a list

Lists can be described either at the machine level – in terms of how lists are represented inside a computer – or at the user level in terms of a convenient paper and pencil notation. At the user level we may define a list as an ordered sequence of elements enclosed within a pair of parentheses. An example is the list of three elements

$$(A\ B\ C) \tag{1}$$

Elements of a list can be either (a) atoms or (b) lists. Atoms are the irreducible building blocks of lists – they are elements which cannot be analysed further. In (1), the three elements A, B, and C are all atoms. An example of a list containing both atoms and lists is

$$((A\ B)\ C\ (D\ E)\ F) \tag{2}$$

This contains four elements: (A B), C, (D E) and F. Lists can be nested within lists down to any level; we might have, for example,

$$(((A\ B))\ (C\ (D\ E))\ F) \tag{3}$$

Lists therefore exhibit a hierarchical structure: we can speak of the levels of a list. One way to display this hierarchical structure graphically is by means of a tree with branches leading down to the various levels. In Fig. 1, for example, we display a possible tree representation for the

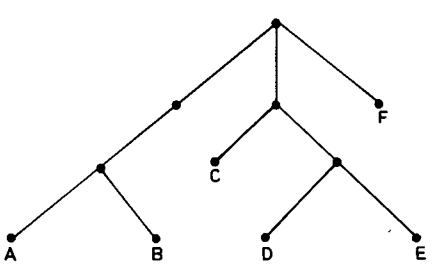


FIG. 1. A possible tree representation of the list (((A B)) (C (D E)) F).

list (3). A more convenient representation is in terms of a binary tree where each vertex has exactly two branches pointing downwards. We shall introduce the binary tree representation in Section 3 when we discuss the way lists are structured inside the machine.

A list that contains no elements is called the null list and is denoted by the symbol ( ).

In LISP an atom can be either a number – integer, real or floating point – in which case we speak of a numeric atom, or it can be a string of capital letters and decimal digits, e.g.

A

SIGMA2

AVERYLONGSTRINGOFCHARACTERS

X2Y4Z6

with the restriction that the first character must be a letter. Atoms of this type are called literal atoms. The maximum allowed length of an atom depends on the particular LISP implementation in use; in the CDC 6000 implementation, for example, the maximum allowed length is 30 characters.

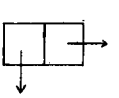
To every atom the LISP system associates a property list consisting of a sequence of the atom's attributes, each one followed by the corresponding value of the attribute. For example, an atom PROTON might have a property list of the form

(PNAME PROTON SPIN ONEHALF ISOSPIN ONEHALF)

The attributes (also called "indicators") are here PNAME (an acronym for PRINT NAME), SPIN and ISOSPIN, and are each followed by the corresponding values. In this way we can use the property list as a convenient structure on which to "hang" any information about the atom that we might need. We shall discuss property lists in more detail in Section 7.

3. Machine representation of lists

It is clear that if lists consisted only of linear arrays of atoms, say (A B C) for example, these could be stored in sequential memory locations in much the same way as FORTRAN arrays. It is the hierarchical structure of lists – the fact that list elements can themselves be lists – which forces one to adopt a more flexible, more complex storage arrangement. Without entering into details of computer hardware, we shall represent a computer word as a cell or rectangle containing a pair of addresses called "pointers":



To each list element – whether it is an atom or a sublist – we now associate a cell in which the first pointer designates where in the machine's

memory the corresponding element is to be found, while the second pointer designates where the continuation of the list is located.

As an example, consider the machine-level representations of the lists (1) and (3). These are displayed in Figs 2 and 3, respectively. For atomic list elements, e. g. A, B or C, the first pointer in the corresponding rectangle points to the property list of the atom. On the other hand, if the element is a sublist, e. g. (A B), the pointer designates the address of the sublist. The end of a list or a sublist is represented by $\rightarrow \emptyset$, a pointer to some fixed location in the machine. Since at the end of a list the continuation is simply the list containing no elements we can identify the symbol $\rightarrow \emptyset$ with the null list symbol ( ) introduced previously.

The type of structure exhibited in Figs 2 and 3 is in fact just a binary tree. Each rectangle represents a node of the tree and each node is connected to two adjacent nodes by a pair of branches represented by the two pointers. The machine-level representation of a list is also referred to as a "linked list": the pointers serve as links between computer words

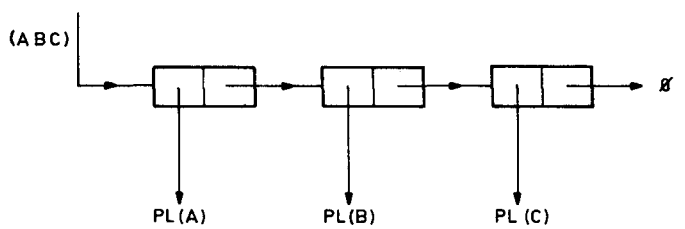


FIG. 2. Machine-level representation of the list (A B C).

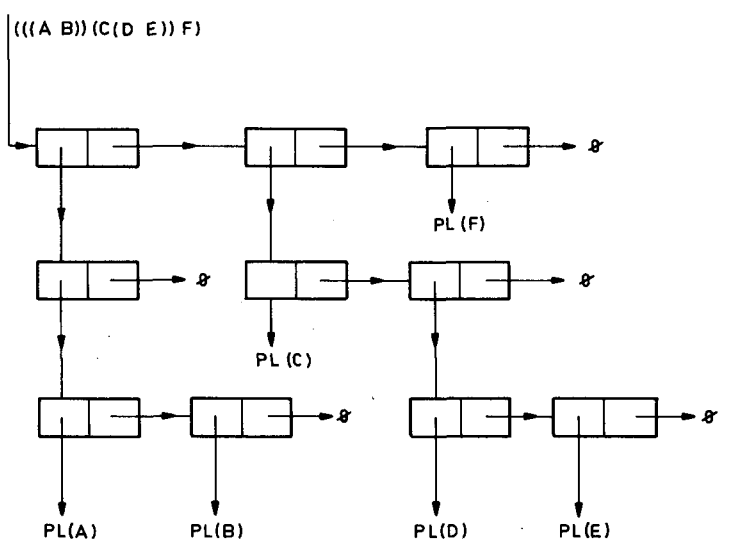
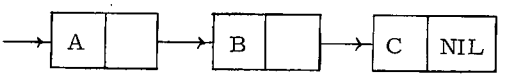


FIG. 3. Machine-level representation of the list (((A B)) (C (D E)) F).

which can be, and generally are, scattered randomly throughout the memory as a result of the complex transformations to which list structures are subjected in the course of a computation.

For the sake of convenience, we shall frequently use an abbreviated notation for the machine-level representation in which (a) we suppress pointers to atomic property lists and simply write the corresponding atom in the first address-field of the rectangle, and (b) replace $\rightarrow \not{0}$ by the atomic symbol NIL. Thus (A B C), for example, will be simply represented as



but it should be borne in mind that this is simply an abbreviation. It is not the atoms A, B, C, and NIL but pointers to the corresponding property lists that are stored in the cells which make up the list structure.

NIL is unique in that it is the only element of a list structure which is both a list and an atom at the same time: we have the identity

$$NIL = ( )$$

4. S-expressions

In addition to lists, LISP can also process a somewhat more general type of data structure known as an S-expression ("symbolic expression"). An S-expression is either

(1) an atom, e.g. A
(2) a dotted pair of atoms, e.g. (B.C)
(3) a dotted pair of S-expressions, e.g. (A.(B.C)) or ((A.(B.C)).(D.E))

S-expressions are best understood by referring to the machine-level representation: A dotted pair is represented by a cell containing a pair of pointers, the first pointing to the first element of the pair and the second pointing to the second element. An example is shown in Fig. 4 where we exhibit the machine-level representation of ((A.(B.C)).(D.E)).

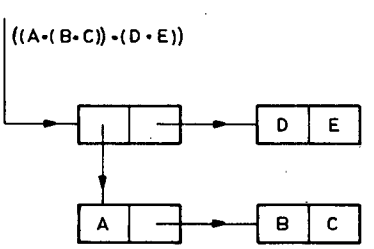


FIG. 4. Machine-level representation of the S-expression ((A.(B.C)).(D.E)). The atom names represent pointers to the corresponding atomic property lists.

How are S-expressions related to lists? We have seen that in a list structure the second pointer in each cell always points to a list, never to an atom except for the list terminator NIL. When we remove this restriction and allow the second pointer to point to atoms other than NIL we get the more general concept of an S-expression. Thus lists are particular instances of S-expressions and can always be represented as such. For example, the list (A) is just the dotted pair (A. NIL); the list (A B) is the dotted pair (A. (B. NIL)). By the same token, S-expressions like (A. B) or (A. (B. C)) are not lists since they feature second pointers to atoms other than NIL. The S-expression displayed in Fig. 4 has two such pointers – to the atoms C and E. If the corresponding pointers were to be replaced by NIL we would get a list ((A B)D).

5. Basic LISP functions

The LISP system contains five basic functions from which, in principle, all other list processing functions can be built up. The first of these functions is

(I) CAR(X)

where the variable X is any S-expression other than an atom: thus X must be a dotted pair. The value of CAR(X) is defined to be the left part of the dotted pair representing the S-expression. In particular, it follows that if the dotted pair happens to be a list then the value of CAR(X) is the first element of the list. Use of CAR(X) when X is an atom is illegal.

To illustrate the use of this function, say we wish to call CAR(X) for the argument (A. (B. C)). We write

CAR('(A. (B. C))); (4)

and the system returns the value A. As a further illustration the function calls

CAR('(A B C)); (5)

and

CAR('((A B) C (D E) F)); (6)

return the values A and (A B) respectively since these are the first elements of the corresponding lists.

Note the single quotation-mark preceding the argument of the function in (4), (5) and (6). This is used to inform the LISP system that the S-expression which follows the quotation-mark stands for itself. This is an extremely important concept in LISP and should be clearly understood. Consider the case of FORTRAN. In FORTRAN a string of symbols like A or SIGMA2 normally stands for a variable having a numerical value at any given time during the computation or the name of a function or subroutine which operates on numerical data. If we wish SIGMA2, for example, to stand for itself we must indicate this by writing it as a Hollerith constant, i. e. 6HSIGMA2. The situation in LISP is analogous except for the fact

that the data is now primarily symbolic rather than numeric. Literal strings will normally be interpreted by the LISP system as standing either for variables having S-expressions as their values or for functions which operate on S-expressions (like CAR in the present example). To indicate that the string stands for itself we use the quotation-mark. The meaning of the quotation-mark in LISP is essentially the same as in ordinary language: London refers to a city but "London" refers to the word itself.

To recapitulate, the LISP system attempts to evaluate every non-quoted literal string that it encounters during execution of a program. So the rule is: Quote any S-expression which one wishes to shield from evaluation, *i.e. write 'A – not A – if A stands for itself.* Note, however, that for numeric atoms, quoting is not necessary as these are always interpreted as standing for themselves.

The second basic LISP function is

(II) CDR(X)

where X is again any S-expression other than an atom. The value of CDR(X) is defined to be the right part of the dotted pair. In particular, if X is a list then the value of CDR(X) is the list derived from X by removing its first element. Use of CDR(X) is illegal if X is an atom. As an illustration, the function calls

CDR('(A.(B.C))); (7)

CDR('(A B C)); (8)

CDR('((A B) C (D E) F)); (9)

return the values (B.C), (B C) and (C (D E) F) respectively.

The third basic function is the "constructor" function

(III) CONS(X, Y)

also denoted [6]

X.Y

This acts as the inverse to CAR and CDR in the sense that it reconstructs a list or an S-expression from the components that have been obtained by application of CAR and CDR. The arguments X and Y are both S-expressions and the value of CONS(X, Y) is the dotted pair (X.Y). As an example, the function call

CONS('A,'(B C)); (10)

returns the value (A.(B C)) or in list notation (A B C). As a further example, the function call

CONS('(A.B),'(C)); (11)

returns ((A.B).(C)) or ((A.B) C).

The fourth function is a predicate

(IV) ATOM(X)

which tests if the argument X is an atom. If so, then ATOM(X) returns the value T (true), if not, it returns the value NIL (false). The LISP system uses the two special atomic symbols T and NIL to denote truth and falsehood, respectively. Examples of the use of ATOM(X) are the calls

ATOM('A); (12)

ATOM(NIL);[2] (13)

ATOM('(A B C)); (14)

which return the values T, T and NIL respectively.

The last basic LISP function is the predicate

(V) EQ(X, Y)

which tests whether X and Y are the same atom. If so, EQ(X, Y) returns T, if not, it returns NIL. Normally, EQ(X, Y) should not be used if either X or Y or both are non-atomic as it may return a wrong result.

The first two functions – CAR and CDR – are called "selector" functions since they can be applied repetitively to select any component of a given list. For example, if we wish to "project out" the atom B from the list ((A B) C (D E)) we simply form

CAR(CDR(CAR('((A B) C (D E)))));

giving us the value B. The LISP system provides a convenient abbreviated notation for such combinations of CAR and CDR. We simply concatenate the sequence of middle letters and sandwich it between C and R. Thus CAR(CDR(CAR(X))) can be written as

CADAR(X)

and similarly CAR(CAR(CDR(CDR(X)))) can be abbreviated to[3]

CAADDR(X)

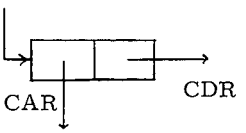
The functions CAR and CDR have a simple interpretation in terms of the machine-level representation. In the list (A B C), for example, the CAR

---

[2] Note that special atoms like NIL and T need not be quoted when they appear in programs. The reason is that they have themselves as values. This is explained in the LISP 1.5 Programmer's Manual [1], p.22.

[3] Functions like CADR, CDAR, CADAR, etc. are very useful in LISP programming. The possibility of defining these mnemonic abbreviations explains why "CAR" and "CDR" are preferred to the more descriptive terms "head" and "tail". The terms CAR and CDR were originally coined by McCarthy as abbreviations for "contents of address part of register" and "contents of decrement part of register", respectively.

is the pointer to the property list PL(A) while the CDR is the pointer to the continuation, i.e. the list (B C). Similarly, for the list ((A)) (Fig. 5), the CAR of ((A)) is (A), represented by the pointer to the lower cell in Fig. 5 while the CDR is the null list ( ) represented by the pointer NIL in the upper cell. The NIL pointer in the lower cell is the CDR of the sublist (A) while the CAR of (A) is the pointer to the property list of the atom A. In general, the CAR and CDR of a list or sublist are just the pair of pointers contained in the corresponding cell:



We have stated that the constructor function, CONS(X, Y), acts as the inverse of CAR and CDR. It is important, however, to understand the action of CONS at the machine level. Given an S-expression (X. Y) stored in some cell, the constructor CONS(X, Y) = (X. Y) is not a pointer to the same cell. Instead, a new storage cell is allocated to represent CONS(X, Y). In this new cell, the first pointer is set to point to X and the second to point to Y. This is illustrated in Fig. 6a. As a further example, suppose we have a list (A B C) stored in the machine. Figure 6b illustrates the action of

$$\text{CONS('A, '(B C))} \rightarrow \text{(A B C)}$$

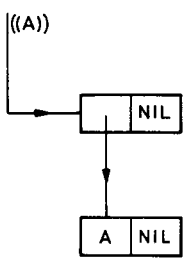


FIG. 5. Machine-level representation of ((A)).

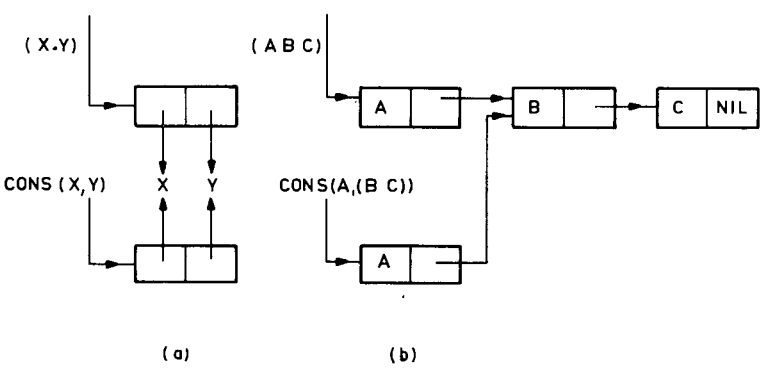


FIG. 6. Action of CONS function at machine level.

Again, the result is not the original pointer to (A B C) but a pointer to a new cell. Each call to CONS during execution of a program results in the allocation of a new word of storage. The latter is obtained from a special list of all available words called the "free storage list". When a word is required following a call to CONS, the CAR of the free storage list is allocated and the free storage list is reset to the CDR of what it was previously. Clearly some means of returning words to the free storage list is necessary since the free storage list can be quickly exhausted during program execution. It is one of the advantages of LISP that this task is not the programmer's responsibility but is performed automatically by a special routine known as the "garbage collector". Called whenever the free storage list runs out of words, the garbage collector marks all words that form part of active list structures – i.e. which are needed by the computation – and then returns all unneeded words to the free storage list from which they can again be allocated when needed.

6. LISP programming

As was stated at the beginning of the preceding section, once the five basic functions CAR, CDR, CONS, ATOM and EQ are made available to the user, precoded in machine language, all other list processing functions can, in principle, be defined as LISP programs in terms of the five primitive functions.

Consider, for example, the very important predicate function NULL(X) which tests whether X is the atom NIL – i.e. the null list – and returns the value T if it is and NIL if it is not. A short program for NULL(X) in the REDUCE language is

```
LISP PROCEDURE NULL(X);
    IF ATOM(X) THEN EQ(X,NIL) ELSE NIL;
```

The syntax used here is essentially the same as that of ALGOL and the meaning of the program should be immediately apparent. If X is an atom then NULL(X) is true if EQ(X, NIL) is true; otherwise NULL(X) is false (NIL). The program is prefaced by a procedure declaration which defines NULL as a LISP function with a single argument X. X plays the role of a "formal parameter" which must be replaced by an "actual" or effective parameter when the function NULL is called[4]. To call NULL for an actual parameter X = (A B), for example, the user writes

```
NULL('(A B));
```

and upon execution the system should return the value NIL since (A B) is not the null list.

A further point concerns the use of parentheses enclosing arguments of functions. REDUCE allows the user to suppress these parentheses

[4] The terms "formal parameter" and "actual parameter" are taken from FORTRAN or ALGOL terminology but it should be borne in mind that these "parameters" are in fact list variables.

in the case of functions of a single argument. Thus the program for NULL(X) could equally well be written in the form

```
LISP PROCEDURE NULL X;
    IF ATOM X THEN EQ(X, NIL) ELSE NIL;
```

Use of this facility improves the readability of LISP programs and we shall make use of it throughout the remainder of this paper.

We have used the above simple program for NULL(X) as an illustration of some elementary features of LISP programming, but in practice the NULL predicate is used so often that it would be extremely inefficient to code it in LISP. In fact, NULL together with over one hundred other frequently used list processing functions are precoded – usually in machine language – and made available to the user as part of the LISP system[5]. For the purpose of this introduction, however, we shall proceed as though this were not the case and, rather than provide the reader with a formal syntactic description of REDUCE as given in Ref. [6], we shall use REDUCE to code several commonly-used LISP functions. This will illustrate both the syntax of the language and, more important, the basic techniques used in handling lists.

Among these techniques, the most important is that of recursion. The FORTRAN language makes no provision for this; it relies exclusively on iteration for the performance of repetitive operations. In practice, this is largely true also of ALGOL, although ALGOL does in fact allow recursion to be used. For handling lists, however, recursion is far more convenient than iteration and LISP programming techniques rely very heavily on this tool.

Consider the following example of a recursive program to append a list Y to a list X:

```
LISP PROCEDURE APPEND(X, Y);
    IF NULL X THEN Y ELSE CONS(CAR X, APPEND(CDR X, Y));
```

Let us call APPEND for the actual parameters X = (A B) and Y = (C D). We write

```
APPEND('(A B), '(C D));
```

and upon execution the system should return the value (A B C D). It is worth tracing through the action of the program to see how the recursive mechanism works. On the first call to APPEND, X = (A B) and Y = (C D) and the procedure returns the value (we omit quotes) CONS(A, APPEND((B), (C D))). We have therefore a second call to APPEND for X = (B), Y = (C D) and the procedure is re-entered for this new set of actual parameters.

---

[5] LISP also provides functions for doing arithmetic on numeric atoms. We shall refer briefly to some of these in Section 8.

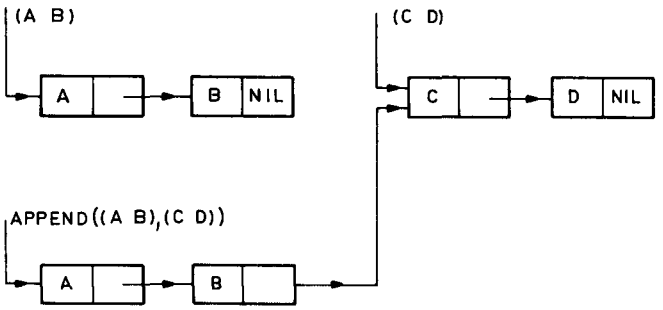


FIG. 7. Result of APPENDing (A B) to (C D) at machine level.

Proceeding in this fashion we have the following sequence (we suppress quotation-marks for the readability)

APPEND((A B), (C D)) → CONS(A, APPEND((B), C D)))
→ CONS(A, CONS(B, APPEND(NIL, (C D))))
→ CONS(A, CONS(B, (C D)))
→ CONS(A, (B C D))
→ (A B C D)

Note that on the final call to APPEND, X is NIL; hence the procedure returns the value Y, i.e. (C D). This accounts for the third line above. It is instructive to look at the result of appending (A B) to (C D) at the machine level. This is shown in Fig. 7. The original lists (A B) and (C D) still exist in the machine. On the other hand, (C D) has grown a new list of heads to form the list (A B C D), as a result of the CONSing operations carried out during the execution of APPEND.

We see that recursion has here its usual mathematical meaning: a function is defined in terms of itself. The procedure is well defined because at each step one works down the list X by means of CDR until one eventually reaches the null list at which point the procedure prescribes the value Y. The LISP system of course must keep track of the intermediate results along the way so that it can work its way back, as in lines 4 and 5 above, but this is not the programmer's responsibility.

A second example is the function SUPERREVERSE which reverses all levels of a list, i.e.

SUPERREVERSE('((A B) C (D E) F)) → (F (E D) C (B A))

A short LISP program to do this uses the APPEND function defined above:

```
LISP PROCEDURE SUPERREVERSE X;
   IF ATOM X THEN X
   ELSE APPEND(SUPERREVERSE CDR X, LIST SUPERREVERSE CAR X);
```

where the function LIST X defined by

```
LISP PROCEDURE LIST X;
    CONS(X, NIL);
```

serves to transform X to (X). The logic of the program for SUPERREVERSE is that if X is an atom (including the null list) then SUPERREVERSE(X) is just X itself. Otherwise it is the result of appending (a) the list obtained by applying SUPERREVERSE to CAR X and making this a list, to (b) the list obtained by applying SUPERREVERSE to CDR X. Notice that in this example recursion works on both the CAR and the CDR of the main list, i.e. on both the main list and on all its sublists. This is necessary in order that all levels of X be reversed. A program which would reverse only the top level of a list X would be

```
LISP PROCEDURE REVERSE X;
    IF NULL X THEN NIL
    ELSE APPEND(REVERSE CDR X, LIST CAR X);
```

Here recursion works only on the CDR of the main list as we work our way down it. As an exercise we urge the reader to apply REVERSE and SUPERREVERSE to a few examples to get a feeling for the operation of the programs.

LISP programs can also be written using iteration. As an example, consider a program to find the first atom of a list:

```
LISP PROCEDURE FIRSTATOM X;
    BEGIN
    G: IF ATOM X THEN RETURN X;
       X := CAR X;
       GO TO G
    END;
```

Note the special syntax here: A label – in this case, G – can only be used within a "compound statement" – a series of statements enclosed between a BEGIN and an END[6]. The iteration proceeds by computing successive CARs of X by means of the assignment statement X := CAR(X). When the first atom is found, RETURN X transfers control out of the loop and sets X as the value of the procedure.

---

[6] A precise syntactic definition of a compound statement is given in Ref. [6]

⟨compound statement⟩ : : = BEGIN ⟨compound tail⟩
⟨compound tail⟩ : : = ⟨unlabelled compound tail⟩
| ⟨label⟩ : ⟨compound tail⟩
⟨unlabelled compound tail⟩ : : = ⟨statement⟩ END
| ⟨statement⟩ ; ⟨compound tail⟩

This notation for syntax definition is due to J. W. Backus. For example, the notation ⟨a⟩ : : = ⟨b⟩ | ⟨c⟩ : ⟨d⟩ means that the syntax element a is either the syntax element b or the syntax element c followed by a colon followed by the syntax element d. Note the use of recursion in the above definition.

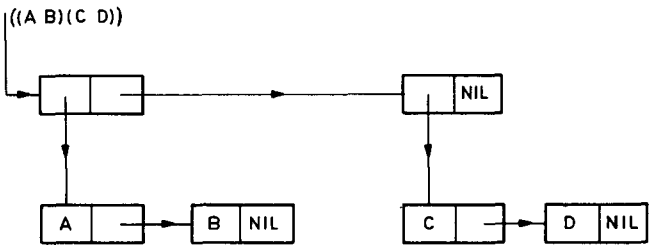


FIG. 8. Operation of SUPERREVERSE on the list ((A B) (C D)).

When should one use recursion and when should one use iteration? By and large, recursion is more convenient when dealing with lists. Iteration is only convenient when a program proceeds sequentially down a list or a sublist without backtracking. For example, in the above program FIRSTATOM, the iteration proceeds by successively computing CARs until the first atom is found. Under these conditions, a recursive program, i.e.

```
LISP PROCEDURE FIRSTATOM X;
    IF ATOM X THEN X ELSE FIRSTATOM CAR X;
```

does not represent any real gain as far as convenience is concerned. Similarly, one could just as easily have written an iterative program for the REVERSE function; the reader might do this as an exercise. If, however, the same operations are to be carried out on the main list and all its sublists, as in SUPERREVERSE for example, then recursion proves to be far more convenient. To see this, consider the operation of SUPERREVERSE on the list ((A B) (C D)) represented in Fig. 8. An iterative program would have to take into account the fact that once the sublist (A B) had been reversed it would be necessary to backtrack to the main list prior to reversing (C D). But the pointers are unidirectional – we cannot use CAR or CDR to help us backtrack – so that additional book-keeping variables would be required to enable the program to find its way back to the main list. Recursion does this for us automatically but an iterative program to do the same job would look extremely clumsy. The reader might try to write an iterative program for SUPERREVERSE to convince himself of this.

Iteration and recursion can also be used jointly. A good example is the following program for SUPERREVERSE which is partly iterative and partly recursive:

```
LISP PROCEDURE SUPERREVERSE X;
    BEGIN SCALAR A,B;
      L: IF NULL X THEN RETURN A;
         B := CAR X;
         IF ATOM B THEN GO TO M;
         B := SUPERREVERSE B;
```

```
    M: A := CONS(B,A);
       X := CDR X;
       GO TO L
  END;
```

Note the statement SCALAR A,B; Its function is to declare two list variables A and B for use by the program. When SUPERREVERSE is called, these variables are automatically initialized to NIL by the system and are then reset by the assignment statements during execution. This program provides a nice illustration of the points made above: recursion is used on the successive CARs of the main list – thus allowing for automatic backtracking once a sublist is reversed – while iteration is used to work down the main list and down each sublist by taking successive CDRs.

There is a particularly useful class of LISP functions which take functions as their arguments. The most important such function is MAPLIST defined by

```
LISP PROCEDURE MAPLIST(X,FN);
    IF NULL X THEN NIL
    ELSE CONS(FN X,MAPLIST(CDR X,FN));
```

MAPLIST is a function of two arguments. The first, X, is a list variable while the second argument FN is a function of a single argument. The value of MAPLIST as defined by the above procedure is a new list composed of the elements obtained by applying FN successively to X, CDR X, CDDR X, CDDDR X, and so on. As an illustration of the use of MAPLIST let us suppose that we wish to input a list ((A B) (C D E)) and output a list composed of the first atoms of the two sublists. We write

```
MAPLIST('((A B) (C D E)), FUNCTION FIRSTATOM);
```

and the system will return the value (A C). Observe that FIRSTATOM appears here preceded by the word FUNCTION; this informs the LISP system that it is dealing with a functional argument, not an ordinary variable, and ensures that the functional argument will be processed correctly[7].

Let us take another example: Suppose we wish to CONS an atom V on to every element of a list, say (A B C). We might first define a new function CONSV by

```
LISP PROCEDURE CONSV X;
    CONS('V,CAR X);
```

and then call MAPLIST with the functional argument CONSV:

```
MAPLIST('(A B C), FUNCTION CONSV);
```

[7] A detailed explanation of how this works is given in Ref.[10], p.64.

This returns the value ((V.A) (V.B) (V.C)). But if CONSV is to be used only once in the course of a larger program we might wish to avoid giving it an explicit name. How, then, are we to refer to it in the call to MAPLIST? LISP allows us to do this by means of a device called a "lambda expression" [13,14]. Instead of defining a new function as above we simply write

```
MAPLIST('(A B C), FUNCTION (LAMBDA (X); CONS('V,CAR X)));
```

and the system again returns the value ((V.A) (V.B) (V.C)). The key point here is that the LISP interpreter recognizes the lambda expression

```
(LAMBDA (X); CONS('V,CAR X))
```

as a function of a single argument X whose operative part – the so-called "body" of the lambda expression – is CONS('V,CAR X). Because X is enclosed in parentheses immediately following the word LAMBDA, the interpreter knows that the symbol X in CONS('V,CAR X)) is the variable which must be paired to the actual argument, in this case the list (A B C). This process of pairing variables with arguments is called "lambda binding".

Lambda expressions are treated by the system in exactly the same way as functions defined by LISP PROCEDURE declarations. For example, instead of calling APPEND for the actual arguments (A B) and (C D):

```
APPEND('(A B),'(C D));
```

we could equally well write

```
(LAMBDA (X,Y); APPEND(X,Y))('(A B),'(C D));
```

In the latter case the lambda expression

```
(LAMBDA (X,Y); APPEND(X,Y))
```

is recognized as a function of two variables X and Y and the call is executed by first binding X to the first argument (A B), binding Y to the second argument (C D), and then applying the body of the lambda expression, in this case APPEND(X,Y), to the effective arguments (A B) and (C D). Lambda expressions with three or more arguments can also be defined. However, only single-argument functions can be used as functional arguments for MAPLIST.

Owing to the special syntax of REDUCE, lambda expressions are primarily a convenient programming device. In LISP 1.5, however, lambda expressions play a crucial role. Indeed all user-defined functions in LISP 1.5 are programmed as lambda expressions which are represented in the machine in the form of lists [1]. What the REDUCE system does is to translate LISP PROCEDURE function definitions into LISP 1.5 lambda expressions.

To conclude this section, consider the following LISP program which takes as input data a list of n atoms and outputs a list of the n! permutations of the input list. The program consists of four function definitions – MAPCON, MAPCONS, DELETE and PERMUTE – followed by a call to PERMUTE for the input list (A B C). The value returned is the list ((A B C) (A C B) (B A C) (B C A) (C A B) (C B A)).

```
LISP PROCEDURE MAPCON(X,FN);
    IF NULL X THEN NIL
    ELSE APPEND(FN X,MAPCON(CDR X,FN));

LISP PROCEDURE MAPCONS(V,U);
    MAPLIST(U, FUNCTION (LAMBDA (X); CONS(V,CAR X)));

LISP PROCEDURE DELETE(V,U);
    IF NULL U THEN NIL
    ELSE IF EQ(CAR U,V) THEN DELETE(V,CDR U)
    ELSE CONS(CAR U,DELETE(V,CDR U));

LISP PROCEDURE PERMUTE X;
    IF NULL X THEN LIST NIL
    ELSE MAPCON(X,FUNCTION(LAMBDA (J);
                    MAPCONS(CAR J, PERMUTE DELETE(CAR J,X))));

PERMUTE('(A B C));
```

The function MAPCON in the above program is similar to MAPLIST except that the resulting list is obtained by APPENDing rather than by CONSing.[8] The second function MAPCONS(V,U) CONSes an S-expression V onto a list U; this is essentially the function we considered earlier except that V is now a variable. DELETE(V,U) deletes the atom V from a list of atoms U. All three functions are called by the function PERMUTE which produces the list of permutations from the input list. The logic in the definition of PERMUTE is fairly tricky and we urge the reader to trace it through carefully.

7. LISP program structure – property lists

A complete LISP program normally consists of a number of function definitions prefixed by LISP PROCEDURE declarations and a number of calling statements in which system functions and/or user-defined functions are applied to initial data. A small-scale example is provided by the permutations program at the end of the preceding section, which consists of four function definitions and one call to a user-defined function, i.e. PERMUTE. Such a function call is referred to as a "top level" call – it is on the same level of the program as the function definitions for MAPCON, MAPCONS, DELETE and PERMUTE. The call to MAPCON in the definition of PERMUTE is a second-level call and the call to MAPCONS is third level. Large LISP programs may involve hundreds of functions that call each other on different levels.

---

[8] In the definition of MAPCON given in Ref. [1] a concatenation operation is used instead of APPEND. At the user level the effect is the same as in our definition but there is a difference at the machine level.

The top level of a program can also contain – in addition to function definitions and function calls – assignment statements which set values to variables that are "global" to the entire program. Variables which appear in the variable list of a LISP PROCEDURE definition, i.e. X and Y in

```
LISP PROCEDURE APPEND(X, Y);
    IF NULL X THEN Y
    ELSE CONS(CAR X, APPEND(CDR X, Y));
```

are local to the relevant function; they are bound when the function is called but their bindings cannot be retrieved outside the scope of the function call. Global variables are used when it is necessary to transmit information between functions at all levels of the program. Thus if we wished to refer throughout the program to a certain fixed list, say (A B C), we could either refer each time to '(A B C) or alternatively we could define a global variable G and give it the value (A B C) by means of the top-level assignment statement

```
G := '(A B C);
```

G can then be used in place of (A B C) throughout the program.

As in all programming languages, the user can select whatever names he pleases for his LISP functions subject to certain restrictions: it is illegal to use (1) names whose first character is a number, (2) names of functions already present in the LISP system, and (3) "reserved words". The latter are words which are used by the syntactical constructions of the REDUCE language such as IF, THEN, ELSE, BEGIN, END, RETURN, LAMBDA and FUNCTION or the special atomic symbol NIL. A further set of reserved words [6] is DO, FOR, STEP, UNTIL and WHILE which are used in forming DO loops[9] as in ALGOL.

In Section 2, we referred to the fact that for every atom read into the machine, the LISP system creates a property list which stores information about the atom. These properties are of two types:

(1) Properties represented by single elements that convey information just by their presence or absence. These are called flags.
(2) Properties represented by two elements. The first element is then called an indicator and the second is a pointer to the corresponding value.

A simple example should illustrate this concept. In Fig. 9, a possible property list for an atomic symbol "PROTON" is displayed. Although details may vary from one implementation to another the structure exhibited in Fig. 9 is fairly typical. The property list is characterized by having the special constant -1 as its first element; this is called the "atom.head". This is followed in our example by one flag, FERMION, to indicate that the atomic symbol represents a fermion. There is then a sequence of indicators – NAME, SPIN, ISOSPIN – and after each indicator a pointer

[9] In practice, the use of DO loops is rather infrequent in list processing and they can of course always be avoided by using GO TO statements.

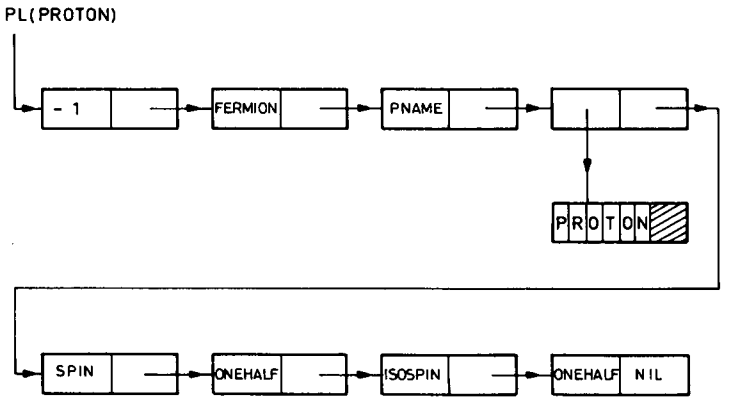


FIG. 9. A property list for the atomic symbol PROTON.

to the corresponding value. Bear in mind that flags are themselves atomic symbols having their own property lists. The same applies to indicators and their values; the exception here is the value of PNAME which is simply a string of characters – "PROTON" – representing the print name of the atom. The computer word or words in which the print name is stored is located in a special area of memory called the full word space.

LISP provides a number of functions for accessing information on atomic property lists or for adding new properties to these lists. The function call

FLAG(X,'FERMION);

places the flag FERMION on the property list of each atom in the list X. In particular, if X is the one element list ('PROTON) the flag FERMION is placed on the property list of PROTON. To test for the presence of a flag, LISP also provides a predicate function FLAGP; the calling statement

FLAGP('A,'FERMION);

returns T if the atom A has the flag FERMION on its property list and NIL otherwise. Note that whereas the first argument of FLAG is a list, the first argument of FLAGP is an atom.

Three other functions, PUT, DEFLIST and GET are designed to handle indicators. PUT is a function of three arguments, PUT(A,I,V), which places the indicator I and its value V on the property list of the atom A. Thus in our example, if we call

PUT('PROTON,'MASS,'M);

the system will add the indicator MASS and its associated value M to the property list. To put different values of a given indicator onto different atoms, one uses the function DEFLIST(X,I) which takes two arguments: the first, X, is a list of atom-value pairs and the second, I, is the indicator to be used. Thus the call

DEFLIST('((PROTON ONEHALF) (RHO ONE) (PION ZERO)),'SPIN);

places the spins $\frac{1}{2}$, 1 and 0 on the property lists of PROTON, RHO and PION respectively. Finally, the function GET(A,I) allows one to retrieve the value corresponding to the indicator I on the property list of the atom A. If the indicator is absent, GET returns the value NIL, otherwise it returns the corresponding value. In our example, the function call

```
GET('PROTON,'SPIN);
```

returns the value ONEHALF.

The functions PUT, DEFLIST and GET allow the user to have direct access to property lists but it should be borne in mind that the user is often accessing these property lists indirectly as a result of the various manipulations performed by the LISP system. The most important use to which property lists are put by the system is in associating function names to the corresponding functional definitions. We have noted earlier that in LISP 1.5 functional definitions are represented by lambda-expressions which are stored in the form of lists inside the machine. These lambda expressions are linked to the corresponding function names through the property lists. When a function is called, the LISP interpreter searches the property list of the atomic symbol representing the function name, retrieves the associated lambda expression and applies it to the arguments of the function. The system also uses property lists to store the bindings of global variables.

8. A short LISP program for symbolic differentiation

In this section, we present a LISP program which takes as its input an algebraic expression – suitably represented in the form of a list – and computes its derivative. To keep the program fairly short, only the following differentiation rules are implemented:

$$(1)\quad \frac{dx}{dx} = 1$$

$$(2)\quad \frac{dy}{dx} = 0 \text{ for } y \neq x$$

$$(3)\quad \frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$$

$$(4)\quad \frac{d(u.v)}{dx} = u.\frac{dv}{dx} + v.\frac{du}{dx}$$

In a practical differentiation routine, one would of course have to implement rules to handle differences, quotients, powers, sines and cosines, etc. Here our purpose is simply to illustrate how list-processing techniques can be applied to carry out algebraic manipulations.

The first task is to find a suitable way to represent algebraic expressions as lists. The most convenient way to do this is to use the so-called Polish

prefix notation (invented by the Polish mathematician Łukasiewicz) in which operators are written to the left of their arguments. Here are some examples:

Algebraic expression	List representation
x + y	(PLUS X Y)
x . y	(TIMES X Y)
x . (y + z)	(TIMES X (PLUS Y Z))
x + y + z	(PLUS X (PLUS Y Z))

The rule is that algebraic operators and variables become LISP atoms and that the operators PLUS and TIMES are allowed to have only two operands at a time.

A listing of the differentiation routine is given at the end of this section. It accepts as input an algebraic expression represented as a list in Polish prefix form. The output is the differentiated expression, also in Polish prefix form. The heart of the program is the function DERIV which performs the actual differentiation. The four other functions SIMPLIFY, SPLUS, STIMES and COLLECT are included in order to simplify the Polish prefix expressions produced by DERIV. The program is essentially a REDUCE version of parts of a LISP 1.5 differentiation algorithm given by Weissman [11]; the latter includes additional facilities for handling differences, quotients and powers as well as functions for translating from ordinary mathematical notation to prefix notation and back again.

Let us consider each of the five functions in turn. As we have indicated, the actual work of differentiating a Polish prefix expression is done by the function DERIV(E, X). Here E is the algebraic expression that is to be differentiated with respect to X. The structure of the algorithm for DERIV is simply a step-by-step implementation of the four differentiation rules stated above. Line 2 of the program tests if the argument E is an atom; if so, rules 1 and 2 are respectively applied in lines 3 and 4 of the program. Thus a call to DERIV('X, 'X) will return the value 1, while a call to DERIV('Y, 'X) will return 0. If the argument E is not an atom then, in accordance with our input convention, it must be a list of the form

$$(\text{operator} \quad \text{argument}_1 \quad \text{argument}_2)$$

where "operator" stands for either PLUS or TIMES and where the two arguments are either atoms or are themselves lists of this form. The program therefore checks the CAR of E to decide if it is PLUS or TIMES. If it is PLUS then rule 3 is applied in line 6; if it is TIMES then rule 4 is applied (lines 8, 9, 10). The program uses the auxiliary function

$$\text{LIST}(A, B, \ldots, N) = \text{CONS}(A, \text{CONS}(B, \ldots \text{CONS}(N, \text{NIL}) \ldots))$$

generalizing the LIST function of a single argument introduced in Section 6.

As a simple example, consider the top-level call

DERIV('(TIMES X X), 'X);

Since the CAR of the list is TIMES, lines 8, 9 and 10 yield the partially evaluated expression

LIST('PLUS, LIST('TIMES, 'X, DERIV('X, 'X)), LIST('TIMES, 'X, DERIV('X, 'X)))

with two recursive calls to DERIV('X, 'X). Since DERIV('X, 'X) is 1 (line 3), the value returned at the top level is

(PLUS (TIMES X 1) (TIMES X 1))

As a second example, we take the top-level call

DERIV('(PLUS (TIMES 3 (TIMES X X)) (TIMES 2 X)), 'X);

This computes the derivative of $3x^2 + 2x$ in Polish prefix notation. We leave it to the reader to verify that the value returned in this case is

(PLUS (PLUS (TIMES (TIMES X X) 0)
(TIMES 3 (PLUS (TIMES X 1) (TIMES X 1))))
(PLUS (TIMES X 0) (TIMES 2 1)))

From these examples it should be clear that the Polish prefix expressions returned by DERIV are badly in need of simplification. The last expression, for example, could be simplified to

(PLUS 2 (TIMES 6 X))

by building a few obvious simplification rules into the program. This task is assigned to the second function, SIMPLIFY. SIMPLIFY takes as its input the Polish prefix expression produced by DERIV and outputs the expression in simplified form. In carrying out the simplifications, SIMPLIFY calls SPLUS and STIMES; these two functions in turn call the function COLLECT.

A glance at the program for SIMPLIFY shows that it uses a compound statement with two program variables A and B; these are introduced for convenience in writing the program. The algorithm is purely recursive and states that if E is an atom then the value of SIMPLIFY(E) is E itself, whereas if E is a list, i.e. an algebraic expression of the form

(operator argument_1 argument_2)

then, depending on whether the operator is PLUS or TIMES, the value of SIMPLIFY(E) is respectively the value returned by SPLUS when applied to the list

(PLUS SIMPLIFY(argument_1) SIMPLIFY(argument_2))

or the value of STIMES applied to

(TIMES SIMPLIFY(argument_1) SIMPLIFY(argument_2))

The logic here is that SPLUS and STIMES are called only for arguments that are already in their simplest form owing to the earlier application of SIMPLIFY on the sub-expressions. The implementation of the actual simplification rules is therefore assigned by SIMPLIFY to SPLUS and STIMES (and to COLLECT since the latter is called by SPLUS and STIMES).

The two functions SPLUS and STIMES are fairly similar in structure and we shall discuss only SPLUS. We first observe that the program calls upon six as yet undefined functions – NUMBERP, ZEROP, ONEP, AND, EQUAL and EVAL. These are in fact LISP system functions [1, 11]. NUMBERP is a predicate function which takes a single argument and tests if it is a number (i.e. a numeric atom). If so, it returns T, if not, it returns NIL. ZEROP and ONEP are single argument predicates which are true if their arguments are 0 and 1 respectively. AND is a predicate function which takes any number of arguments (in our case only two: NUMBERP CADR E and ZEROP CADR E in line 26) and returns T if all the arguments are true, otherwise NIL. EQUAL is a predicate function of two arguments X and Y; it returns T if X and Y are identical S-expressions and NIL if they are different. Thus EQUAL is the generalization to arbitrary S-expressions of the elementary predicate EQ.

The final function EVAL is one which plays a key role in the workings of the LISP interpreter [1, 11]. It is used here for convenience only. When applied to a list of the form

$$(\text{fn} \quad \text{arg}_1 \quad \text{arg}_2 \ldots \text{arg}_n)$$

where fn is some system or user-defined function of n arguments, EVAL returns as its value the function fn applied to the values of the n arguments. If, in particular, the n arguments are quoted then the value returned by EVAL is just

$$\text{fn}(\text{arg}_1, \text{arg}_2 \ldots \text{arg}_n)$$

Armed with these definitions we now turn to the actual operation of the function SPLUS. The input to SPLUS is always a list of the form

(PLUS a b)

where a and b are either atoms or lists (algebraic expressions in Polish form) that are already assumed to be in their simplest form. The function first checks to see if both a and b are numeric atoms. If so then (line 23) SPLUS returns as its value the result of applying EVAL to the list (PLUS a b). PLUS is in fact a LISP system function which takes two numeric atoms as its arguments and returns their sum. Thus EVAL – and hence SPLUS – will return the value a + b in this case. As an example, if SPLUS is given the argument (PLUS 2 1) it will return 3 as the value.

Assuming now that b is a number but a is not, the algorithm goes on (line 24) to check if b is zero; if so, the value returned is a. If b is non-zero, SPLUS forms the list (PLUS b a) and passes it to COLLECT (line 25) for further possible simplifications.

If b is not a number then line 26 checks to see if a is zero; if so, b is returned as the value of SPLUS. If this fails, the algorithm proceeds to check if a and b are identical S-expressions. If they are, SPLUS forms

the list (TIMES 2 a) and calls COLLECT with this argument (line 27). If all these steps fail the original list is passed on to COLLECT.

The input to COLLECT from SPLUS is therefore a list of the form (PLUS a b) in which the first argument a is always the numeric atom if there is one; the second argument is either a literal atom or a list. COLLECT attempts to carry out some further simplifications by means of some elementary pattern matching. If COLLECT receives a list (PLUS a b) where a is a list but where b is a (non-numeric) atom it calls itself recursively with the list (PLUS b a) as its argument (line 44). This ensures that the first argument is always an atom if there is one. It then carries out the following simplifications:

(1) If a and b are numbers then (lines 47-49)

(PLUS a (PLUS b c)) → (PLUS a + b c)

(TIMES a (TIMES b c)) → (TIMES a*b c)

(2) If a and c are numbers then (lines 52-54)

(PLUS (PLUS a b) (PLUS c d)) → (PLUS a+c (PLUS b d))

(TIMES (TIMES a b) (TIMES c d)) → (TIMES a*c (TIMES b d))

If none of these simplifications can be applied, COLLECT returns the list unchanged.

This concludes our discussion of the program. The actual listing follows. The final statement in the program illustrates the calling sequence when applying the program to its initial data.

```
LISP PROCEDURE DERIV (E,X);                                          1
    IF ATOM E THEN                                                   2
       IF EQ(E,X) THEN 1                                             3
       ELSE 0                                                        4
    ELSE IF EQ(CAR E,'PLUS) THEN                                     5
       LIST('PLUS,DERIV(CADR E,X),DERIV(CADDR E,X))                  6
    ELSE IF EQ(CAR E,'TIMES) THEN                                    7
       LIST('PLUS,                                                   8
            LIST('TIMES,CADDR E,DERIV(CADR E,X)),                    9
            LIST('TIMES,CADR E,DERIV(CADDR E,X)))                    10
    ELSE NIL;

LISP PROCEDURE SIMPLIFY E;                                           12
    BEGIN SCALAR A,B;                                                13
          IF ATOM E THEN RETURN E;                                   14
          A := CAR E;                                                15
          B := LIST(A,SIMPLIFY CADR E,SIMPLIFY CADDR E);             16
          RETURN IF EQ(A,'PLUS) THEN SPLUS B                         17
                 ELSE IF EQ(A,'TIMES) THEN STIMES B                  18
                 ELSE B                                              19
END;                                                                 20
```

```
LISP PROCEDURE SPLUS E;                                                    21
     IF NUMBERP CADDR E THEN                                               22
        IF NUMBERP CADR E THEN EVAL E                                      23
        ELSE IF ZEROP CADDR E THEN CADR E                                  24
        ELSE COLLECT LIST(CAR E,CADDR E,CADR E)                            25
     ELSE IF NUMBERP CADR E AND ZEROP CADR E THEN CADDR E                  26
     ELSE IF EQUAL(CADR E,CADDR E) THEN COLLECT LIST('TIMES,2,CADR E) 27
     ELSE COLLECT E;                                                       28

LISP PROCEDURE STIMES E;                                                   29
     IF NUMBERP CADDR E THEN                                               30
        IF NUMBERP CADR E THEN EVAL E                                      31
        ELSE IF ZEROP CADDR E THEN 0                                       32
        ELSE IF ONEP CADDR E THEN CADR E                                   33
        ELSE COLLECT LIST(CAR E,CADDR E,CADR E)                            34
     ELSE IF NUMBERP CADR E THEN                                           35
        IF ZEROP CADR E THEN 0                                             36
        ELSE IF ONEP CADR E THEN CADDR E                                   37
        ELSE COLLECT E                                                     38
     ELSE COLLECT E;                                                       39

LISP PROCEDURE COLLECT E;                                                  40
     IF ATOM E THEN E                                                      41
     ELSE IF ATOM CADDR E THEN                                             42
        IF ATOM CADR E THEN E                                              43
        ELSE COLLECT LIST(CAR E,CADDR E,CADR E)                            44
     ELSE IF EQ(CAR E,CAADDR E) AND NUMBERP CADR CADDR E THEN              45
        IF NUMBERP CADR E THEN                                             46
           LIST(CAR E,                                                     47
                EVAL LIST(CAR E,CADR E,CADR CADDR E),                      48
                CADDR CADDR E)                                             49
        ELSE IF ATOM CADR E THEN E                                         50
        ELSE IF EQ(CAR E,CAADR E) AND NUMBERP CADADR E THEN                51
           LIST(CAR E,                                                     52
                EVAL LIST(CAR E,CADADR E,CADR CADDR E),                    53
                LIST(CAR E,CADDR CADR E,CADDR CADDR E))                    54
        ELSE E                                                             55
     ELSE E;                                                               56
SIMPLIFY(DERIV('(TIMES X X),'X));                                          57
```

REFERENCES

[1] McCARTHY, J., ABRAHAMS, P.W., EDWARDS, D.J., HART, T.P., LEVIN, M.I., LISP 1.5 Programmer's Manual, M.I.T. Press (1965).
[2] CAMPBELL, J.A., HEARN, A.C., J. comput. Phys. 5 (1970) 280.
[3] CAMPBELL, J.A., Comput. Phys. Communs 1 (1970) 251.
[4] BRODY, T.A., Symbol Manipulation Techniques for Physics, Gordon and Breach, New York (1968).
[5] CALMET, J., PERROTTET, M., J. comput. Phys. 7 (1971) 191.
[6] HEARN, A.C., REDUCE 2 Symbolic Mode Primer, Stanford Artificial Intelligence Laboratory Operating Note 62 (Oct. 1970).
[7] HEARN, A.C., REDUCE 2 User's Manual, Stanford Artificial Intelligence Project Memo No. AIM-133 (Oct. 1970).
[8] HEARN, A.C , REDUCE 2: A System and Language for Algebraic Manipulation, Proc. Second Symp. Symbolic and Algebraic Manipulation, Los Angeles, Calif., March 1971.
[9] HEARN, A.C., "REDUCE, a user-oriented interactive system for algebraic simplification", Interactive Systems for Experimental Applied Mathematics (KLERER, M., REINFELDS, J., Eds), Academic Press, New York, London (1968) 79.
[10] BERKELEY, E.C., BOBROW, D.G., The Programming Language LISP: Its Operation and Applications, M.I.T. Press (1964).
[11] WEISSMAN, C., LISP 1.5 Primer, Dickenson Publ. Co. Inc. Belmont, Calif. (1967).
[12] HEARN, A.C., Standard LISP, Stanford Artificial Intelligence Project Memo No. AI-90 (May 1969).
[13] CHURCH, A., The Calculi of Lambda Conversion, Princeton University Press, Princeton, N.J. (1941).
[14] McCARTHY, J., Recursive functions of symbolic expressions and their computation by machine, Communs Ass. comput. Mach. 3 (1960) 184.

GENERATION OF FEYNMAN DIAGRAMS BY THE USE OF FORTRAN

M. PERROTTET
Centre de physique théorique,
C.N.R.S., Marseille,
France

Abstract

GENERATION OF FEYNMAN DIAGRAMS BY THE USE OF FORTRAN.

A FORTRAN program which generates unrenormalized Feynman diagrams is described. This program is based on combinatorial analysis only and acts in the following way: given either an interaction or a mixture of interactions, a number of vertices, a number of external lines and their types, it constructs all the corresponding graphs. To achieve this, one introduces a matrix N(i, j), $1 \leq i \leq IS$, $1 \leq j \leq IT$, where i is the index of rows, IS is the number of vertices, j is the index of columns, and IT/2 is the number of lines involved in the interaction. To each kind of field appearing in the interaction a line of definite type is associated, which is characterized by a positive integer. A line of type b connected to the vertex K will be labelled by the pair (K b). All the lines of the vertex K lie in the K-th row of N, which is simply the juxtaposition of all the pairs (K bj), $j = 2, 4, \ldots, IT$. An internal line is taken to be the connection of two pairs of the same type; the construction of diagrams consists of all possible connections of such pairs. The external lines are defined to be lines which cannot be connected: they are expressed in N by pairs (0 0). The matrix obtained from N by taking into account the external lines is the main input of the program. In the output a diagram is given by a three-column matrix denoted by IRES. A row of IRES is of the form (K b H) and represents an internal line of type b connecting the vertices K to H. Various elimination tests are performed in order to obtain only inequivalent diagrams.

The program which we are going to describe, is called FRENEY; it deals with the problem of generating all the Feynman diagrams [1] to a given order for a given physical process, i.e. for given external lines. There were two reasons for carrying out this work; in fact, at high orders in the perturbation expansion, when writing the diagrams by hand, we risk, first, forgetting a graph and, secondly, writing the same diagram twice, under different forms. For example, the graphs shown in Fig. 1 are quite different in QED, but identical in the $\lambda\Phi^3$-theory.

We now present some definitions. A diagram (or graph) [2] is built up from dots (or vertices) and lines, under conditions which we are now going to make precise. Let us start with the definitions: a line is said to be internal if both ends are attached to dots, and external if only one end is attached to a dot. We have the condition that a line is attached, at least, to one dot and a dot, at least, to one line. The lines can be different; in this case, we shall speak of different types of lines. For example, in Fig. 1 we have two types of lines, symbolized by a dashed line and a solid line. A dot is said to be internal if all lines joined to it are internal; it is said to be external in the opposite case. For a given interaction, the number and type of lines joined to a dot are fixed. In general, the interaction will be the same for each dot of a graph; if this is not the case, we shall speak of a mixture of interactions. Finally, the order of a graph is the number of its dots, both internal and external.

FIG. 1. Graphs different in QED, but identical in $\lambda \Phi^3$.

Let us give an example. In Fig. 1, the graphs are of order 7. We have more precisely 4 internal dots (labelled by 4, 5, 6 and 7), 3 external dots (labelled by 1, 2 and 3), 9 internal lines and 3 external lines. The interaction for these graphs can be written

$$\lambda \Phi_1 \Phi_2^2$$

the field (or line) Φ_1 corresponding to the dashed line (we could say that this line is of type 1), the field Φ_2 corresponding to the solid line (let us say, type 2). The power of the Φ_1 field indicates how often it is connected to a dot. λ is some coupling constant, completely irrelevant for our purpose.

Generally, the problem of constructing diagrams can be formulated in the following way:

Given two integers IS and m, with IS > 0, $m \geq 0$, and an interaction

$$\lambda \prod_{\ell=1}^{A} \Phi_{i_\ell} \tag{1}$$

We have now to find all the graphs with IS dots and m external lines of fixed type, i. e. of fixed index i_ℓ.

Of course, IS and m must be compatible with the chosen interaction. With regard to this point, we recall the well-known relation for the $\lambda \Phi^A$ theory (here $i_1 = i_2 = \ldots = i_A$)

$$A \cdot IS - 2L = m \tag{2}$$

where L is the number of internal lines of the diagram. We note that similar relations hold for more complicated interactions where we have different types of fields. So we have for the $\lambda \Phi_1 \Phi_2^2$ theory [3]:

$$\begin{aligned} 2\,IS - 2\,L_2 &= m_2 \\ IS - 2\,L_1 &= m_1 \end{aligned} \tag{3}$$

where L_1 is the number of internal lines of the Φ_1 field, L_2 that of the Φ_2 field, m_1 the number of external lines of the Φ_1 field, m_2 that of the Φ_2 field. A possible way of solving the problem stated above is to introduce a matrix

$$N(i,j)\ ,\ i = 1,2,\ldots,IS\ ,\ j = 1,2,\ldots,IT$$

In this matrix, i labels the lines and j the columns. By definition, IT = 2A (see expression (1)), i.e. IT is twice the number of lines joined to a dot. For example, in QED we have IT = 6 and in $\lambda\,\Phi^4$ theory IT = 8. Let us remark that the FRENEY program will be general enough to allow the construction of graphs with a mixture of interactions; in this case IT will be defined as twice the maximum number of lines joined to a dot. We should now like to emphasize the fact that the IS dots of a graph must always be labelled by the integers $1,2,\ldots,IS$. This remark is essential for the program to work properly. This allows one to associate each line of the matrix N(i,j) to a dot and, more precisely, the i-th line to the i-th dot. As any interaction is allowed, like in expression (1), it must be possible to have lines of different types: these types will be labelled by the integers $1,2,3,\ldots$ We can now build up a pair (s b) where s is the label of a dot and b the type of a line joined to this dot. Let us denote by $(s\ b_j)$ $j = 2,4,\ldots,$ IT the A pairs corresponding to the set of the lines joined to the dot s. The i-th line of the N matrix will be the juxtaposition of the pairs $(i\ b_j)$ $j = 2,4,\ldots,IT$, i.e.

$$(i\ b_2)\ (i\ b_4)\ (i\ b_6)\ \ldots\ (i\ b_{IT}) \tag{4}$$

Let us remark that, for technical reasons, we impose the order

$$b_j \leq b_{j+2} \tag{5}$$

The relation between expression (4) and the N matrix elements can immediately be written as

$$\begin{aligned} N(i,j) &= i \quad \text{for } j \text{ odd} \\ N(i,j) &= b_j \quad \text{for } j \text{ even} \end{aligned} \tag{6}$$

When there is just one kind of interaction, expression (6) holds for all i. A slightly more complicated rule must be used in the case of a mixture of interactions [4]. Let us illustrate the rules we have just stated with an example in $\lambda\,\Phi_1\,\Phi_2^2$ theory and try to build up the matrix giving the 4-th order graphs; there are two types of lines to consider; we must take $b_2 = 1$, $b_4 = b_6 = 2$ in order to satisfy rule (5) concerning the order between the types. One has IS = 4 and IT = 6 since we have 3 lines per dot. Following rule (6) and taking account of the order between the b_j we have

$$N(i,j) = \begin{bmatrix} 1 & 1 & 1 & 2 & 1 & 2 \\ 2 & 1 & 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 2 & 3 & 2 \\ 4 & 1 & 4 & 2 & 4 & 2 \end{bmatrix} \tag{7}$$

To build up the graphs, the problem is then to combine two by two the pairs of the N matrix, in all possible ways. According to our previous definitions, the external lines will be lines which cannot be combined; we shall replace in N(i, j) the elements of the pairs corresponding to these lines by zeros. So the test will be easy for the computer: if the second element of a pair is different from zero, one tries to combine this pair with another pair of the same type; if it is zero, we go to the following pair, etc. For our example, if one wants the self-energies (i. e. graphs with two external lines) of the Φ_1 field at 4-th order, the N matrix will become:

$$N(i,j) = \begin{bmatrix} 0 & 0 & 1 & 2 & 1 & 2 \\ 0 & 0 & 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 2 & 3 & 2 \\ 4 & 1 & 4 & 2 & 4 & 2 \end{bmatrix} \tag{8}$$

showing clearly that a type-1 external line (i. e. field Φ_1) is attached to the dot 1 and another type 1 external line is attached to the dot 2. The external lines must be absolutely tied to the first dots of the N-matrix. Thus in expression (8) it is not possible to take 1 and 4 or 3 and 4 as external dots. However, their position in a fixed line does not matter, as long as it corresponds to the type wanted.

Finally, if one has m_1 external lines attached to a dot, m_2 external lines to another, m_3 to a third, etc. with $m_1 > m_2 > m_3 > \ldots$ then one must always attach the m_1 external lines to dot 1, the m_2 external lines to dot 2, etc.

The main data of the program will be a matrix such as matrix (8), taking into account the external lines. Of course, matrix (7) may be data, but it will give 4-th-order graphs without external lines, in the $\lambda \Phi_1 \Phi_2^2$ theory. Let us now adduce some examples with illustrations. First, there are some graphs generated by matrix (7) (Fig. 2) and by matrix (8) (Fig. 3).

The self-energies of the Φ_2 field at 6-th order are obtained by taking the following N-matrix:

$$N(i,j) = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 2 \\ 2 & 1 & 0 & 0 & 2 & 2 \\ 3 & 1 & 3 & 2 & 3 & 2 \\ 4 & 1 & 4 & 2 & 4 & 2 \\ 5 & 1 & 5 & 2 & 5 & 2 \\ 6 & 1 & 6 & 2 & 6 & 2 \end{bmatrix} \tag{8a}$$

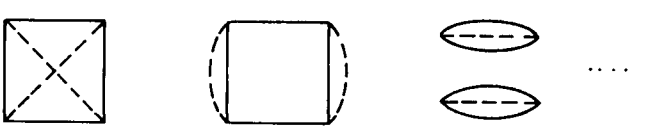


FIG. 2. Graphs generated by matrix (7).

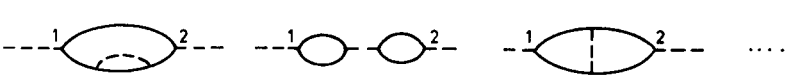


FIG. 3. Graphs generated by matrix (8).

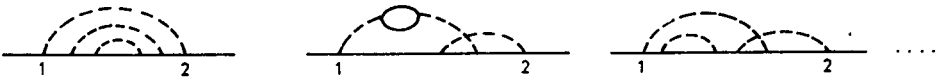


FIG. 4. Graphs generated by matrix (8a).

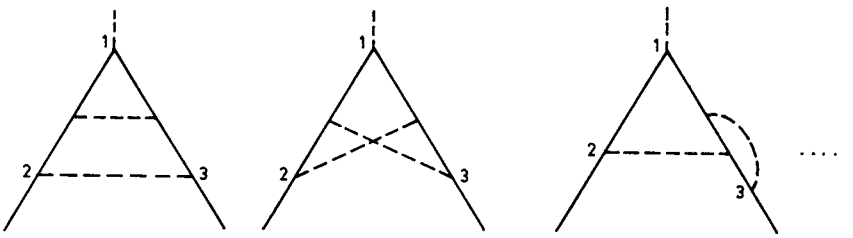


FIG. 5. Graphs generated by first of matrices (8b).

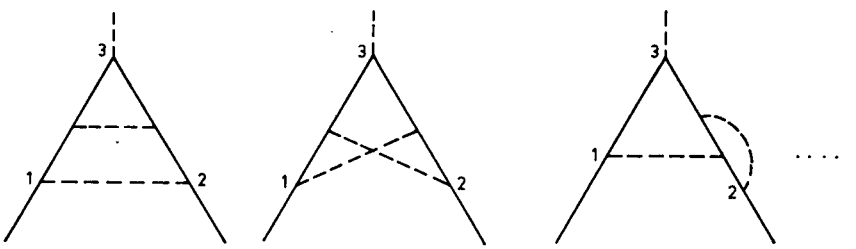


FIG. 6. Graphs generated by second of matrices (8b).

This matrix gives rise to graphs as shown in Fig. 4.

The vertices (i. e. graphs with 3 external lines) at 5-th order are obtained from

$$N(i,j) = \begin{bmatrix} 0 & 0 & 1 & 2 & 1 & 2 \\ 2 & 1 & 0 & 0 & 2 & 2 \\ 3 & 1 & 0 & 0 & 3 & 2 \\ 4 & 1 & 4 & 2 & 4 & 2 \\ 5 & 1 & 5 & 2 & 5 & 2 \end{bmatrix} \quad \text{or} \quad N(i,j) = \begin{bmatrix} 1 & 1 & 1 & 2 & 0 & 0 \\ 2 & 1 & 2 & 2 & 0 & 0 \\ 0 & 0 & 3 & 2 & 3 & 2 \\ 4 & 1 & 4 & 2 & 4 & 2 \\ 5 & 1 & 5 & 2 & 5 & 2 \end{bmatrix} \qquad (8b)$$

The first matrix (8b) will produce graphs as shown in Fig. 5, while the second matrix (8b) will give the graphs shown in Fig. 6.

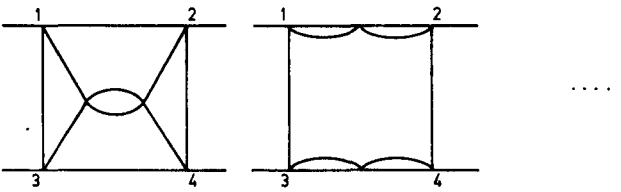


FIG.7. Graphs generated by matrix (8c).

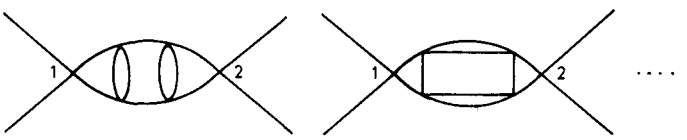


FIG.8. Graphs generated by matrix (8d).

Of course, these graphs are the same as the previous ones since they differ only by the dot numbering.

The scattering graphs (i.e. with 4 external lines) at 6-th order in the $\lambda \Phi^4$ theory are obtained from the matrix:

$$N(i,j) = \begin{bmatrix} 0&0&1&1&1&1&1&1 \\ 0&0&2&1&2&1&2&1 \\ 0&0&3&1&3&1&3&1 \\ 0&0&4&1&4&1&4&1 \\ 5&1&5&1&5&1&5&1 \\ 6&1&6&1&6&1&6&1 \end{bmatrix} \qquad (8c)$$

which will give rise to the graphs displayed in Fig. 7.

We can also consider the matrix

$$N(i,j) = \begin{bmatrix} 1&1&1&1&0&0&0&0 \\ 2&1&2&1&0&0&0&0 \\ 3&1&3&1&3&1&3&1 \\ 4&1&4&1&4&1&4&1 \\ 5&1&5&1&5&1&5&1 \\ 6&1&6&1&6&1&6&1 \end{bmatrix} \qquad (8d)$$

which will give rise to the graphs shown in Fig. 8.

Thus we know how to build up the N-matrix for every physical process. We now want to describe how the FRENEY program works to combine the pairs of the N-matrix to generate all the diagrams. The basic rule to be observed is that we can only combine pairs of the same type. We recall here that the type is defined with a positive integer. We shall take the following procedure to build up the diagrams, after having done an N-matrix copy in the NCϕ matrix (this in order to keep the content of the N-matrix somewhere).

Procedure A

One looks for the first non-zero pair; for this purpose, we begin by i = 1, j = 2 and sweep the line from left to right with only even j; we go to the following line if all types of the first line are zero, etc. Having found a non-zero pair, let us say for i = I, j = J, we call the dot number NSØ1 (it is NCØ(I, J-1) = I by expression (6)) and the type NTY (it is NCØ(I, J)). Then the pair is erased i. e. its type is set to be zero, NCØ(I, J) = 0. Then we look in the I + 1st line, always by sweeping from left to right for a pair with the same type of NTY. If we do not find it, we go to the following line, etc. It is possible that we do not find it in any of following lines. Then we are sure that we must seek in the NSØ1 line. When the second pair is picked up, for i = I, j = J, we can build up a triplet (NSØ1 NTY I) which constitutes an internal line of type NTY joining dots NSØ1 and I. We then erase the last found pair with the statement NCØ(I, J) = 0.

Let us now consider NCØ(i, j) = N(i, j) for all i and j. Let us denote by NINT the number of internal lines for the sought diagrams. Procedure A applied NINT times to the NCØ matrix, this matrix being modified at each application of A since one then erases two pairs, will give rise to NINT internal lines; these lines will be memorized in the IRES (K, NINT, 3) matrix with K = 1, K labelling the successively constructed IRES matrices. Let us remark that NINT, easily determined, constitutes data for FRENEY. So, for matrix (8), we have NINT = 5; let us apply 5 times the procedure A to the copy NCØ of this N matrix; we obtain successively the internal lines

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

This matrix is the first graph built up; it is memorized in the IRES (1, ℓ, m) matrix. It corresponds to the graph shown in Fig. 9. Now we have NCØ(i, j) = 0 for all i and even j. Suppose we have obtained the graph number K. To build up the next graph, we use the following procedure, after having done an N matrix copy in the NCØ matrix:

Procedure B

We look in the NINT-1-th line of the IRES (K, NINT, 3) matrix that we have just obtained to see whether the dot (called NSØ) of the third column can be modified, i. e. increased by one unit. It cannot be modified in the two following cases:

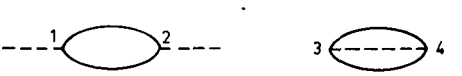


FIG. 9. Graph corresponding to internal lines 122, 122, 314, 324, 324.

1) the dot NSØ is the dot number IS;
2) the modified dot NSØ (called NSØP = NSØ + 1) with the corresponding type constitutes a pair which appears let us say NEPAL times in the N matrix. If this pair appears LIFE times in the previous lines of IRES it is not possible to make it appear again if LIFE = NEPAL.

Now if the dot NSØ cannot be modified we look again in the same line of IRES whether the dot NSØP can be modified. If the dot NSØP cannot be modified, we look whether the dot NSOP = NSØ + 2 can be modified, etc. During this search, two cases can occur: either one of the NSOP dots (or NSO) can be modified or at some point NSØP (or NSØ) = IS. In this last case we begin again procedure B in the NINT-2-th line of IRES. If in this line the dot of the third column cannot be modified (in the general sense given above), we look in the NINT-3-th line of IRES, etc. If the dot of the third column of the first line cannot be modified, we have built up all the diagrams and the procedure stops. However, suppose we have modified the dot of the third column of the L-th line. Then we erase from NCØ(i, j) all the pairs appearing in the IRES matrix up to the modified L-th line. Once this has been done, we copy IRES (K, L, 3) modified in IRES(K+1, L, 3) and we begin again procedure A with the last pairs of NCØ(i, j). The internal lines so obtained allow us to complete the IRES (K+1, NINT, 3) matrix, which we seek then to modify following procedure B, etc.

Of course, it has been necessary to improve these two basic procedures. We shall now briefly describe these improvements.

It is easy to see that it is possible to obtain, by application of procedures A and B, two IRES matrices which differ only by a permutation of lines and thus correspond to the same graph. So we have been led to the definition of the so-called standard form of an IRES matrix, which permits the comparison between the differently constructed graphs. This standard form is defined as follows, for any K and for all i = 1, 2, . . . , NINT-1:

Procedure C

Because of procedure A, we always have

$$\text{IRES}(K, i, 1) \leq \text{IRES}(K, i+1, 1) \tag{9}$$

If the equality holds in this relation, we must have with the same i:

$$\text{IRES}(K, i, 2) \leq \text{IRES}(K, i+1, 2) \tag{10}$$

If the equality holds in this last relation, we always must have with the same i:

$$\text{IRES}(K, i, 3) \leq \text{IRES}(K, i+1, 3) \tag{11}$$

Let us remark that if the equality in relation (9) does not hold, then relations (10) and (11) are possibly not satisfied. The procedure C constitutes the first elimination test of graphs.

We now go to the second elimination test of graphs. With regard to the internal dots, we remark that we are going to obtain, by applying procedures A and B, the two graphs (we take again the N matrix (8)) shown in

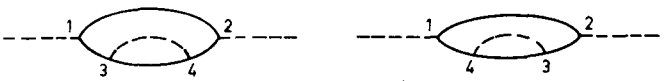


FIG. 10. Graphs produced by matrix (8).

Fig. 10. These two diagrams differ only by the dot numbering. The second elimination test will allow us to eliminate graphs that differ from a given graph only by another internal dot numbering. This test works as follows:

Procedure D

One makes a permutation of the labels of the internal dots on the IRESYM copy of the IRES matrix of the graph considered. The resulting matrix (still called IRESYM) is then put in the standard form and finally compared to previously constructed IRES matrices. Procedure D is repeated for all possible permutations of the internal dot labels, until one finds the equality with another graph, or until all permutations are exhausted.

We now come to the general scheme of the FRENEY program. Suppose we have constructed the graph number K (with $K \geq 2$); then

Procedure E

One looks whether it is possible to modify the graph number K which is not still in the standard form.

E1) the answer is yes: Then we denote by L the number of the IRES (K, NINT, 3) line which has been modified. Then we put the graph number K-1 in the standard form and we compare it to the previous ones. Two cases can occur for this first elimination test (C test):

α_1) The graph number K-1 is equal to the one of the previous graphs. We must then eliminate it, which is achieved by replacing it by the graph number K. One then applies the end of procedure B which constructs a new IRES matrix, having number K. Then we go to E.

α_2) The graph number K-1 is different from all the previous graphs. One then applies the second elimination test to it, if necessary: i. e. the number IMAX of internal dots is ≥ 2 (for IMAX = 0 or 1, we use the end of the procedure B, which constructs a new IRES matrix having number K+1 and we return to E). Two cases can occur for this second elimination test D-test):

β_1) For some permutation of the internal dot labels, we find the equality with the one of the previous graphs. The graph number K-1 is eliminated by replacing it by the graph number K. One applies then the end of procedure B which constructs a new IRES matrix having number K and we return to E).

β_2) After exhausting all possible permutations, we find that the graph number K differs from all the previous ones. One then applies the end of procedure B which constructs a new IRES matrix having number K+1 and we return to E.

TABLE I. SOME RESULTS

1) QED (interaction $\lambda \Phi_1 \Phi_2 \Phi_3$)	KMAX	NADJA	MAX	NINT	T
4th order, photon self-energy	5	7	2	5	7"
6th order, photon self-energy	34	271	4	8	12"
4th order, electron self-energy	5	7	2	5	6"
6th order, electron self-energy	34	271	4	8	13"
5th order, vertex	14	19	2	6	5"
7th order, vertex	151	1351	4	9	1'12"
4th order, photon-photon scattering	9	9	0	4	6"
6th order, photon-photon scattering	117	181	2	7	9"
2) Theory $\lambda \Phi_1 \Phi_2^2$					
4th order, Φ_1 field self-energy	5	11	2	5	6"
6th order, Φ_1 field self-energy	39	664	4	8	25"
4th order, Φ_2 field self-energy	8	11	2	5	6"
6th order, Φ_2 field self-energy	58	619	4	8	24"
5th order, vertex	25	58	2	6	7"
7th order, vertex	303	5431	4	9	9'7"
3) Theory $\lambda \Phi^4$					
3th order, self-energy	4	8	1	5	7"
4th order, self-energy	13	69	2	7	9"
5th order, self-energy	60	1334	3	9	25"

E2) The answer is no: Then the program stops, after examination of the two last constructed matrices; different cases occur according to IMAX $\geq$ 2 or IMAX $\leq$ 1.

We would now like to add some comments to this rough description of the FRENEY program:

1) The K index labels the different graphs; an index NADJA labels all the successively constructed graphs.

2) It is possible to save much machine time by modifying the stop test as follows: the program will stop as the second internal dot appears in IRES (K, 1, 3). Because of the symmetric character of the internal dots, new graphs will not be built up from this point.

Finally, we should like to make precise the input, the output and give some results. At the beginning of the program there are two PARAMETER cards which content the numerical values of

IS IT NINT KM

MAX MAXFA

IS, IT and NINT have already be defined. KM is a number greater than the maximum of K and must be chosen large enough because we do not a priori know the number of differently constructed matrices. MAX is the number of internal dots and MAXFA = MAX!. Then the program reads the N matrix, line by line. We successively find in the output:

1) the matrix N(i, j);

2) the matrix NPERMU, if necessary, which contains all the permutations of internal dot labels;

3) the numerical values of some quantities, which indicate the manner in which the program stopped. For example, if the last two constructed matrices have been eliminated, or only the last one, etc.

4) the KMAX different constructed matrices IRES.

The program has been checked by obtaining the well-known diagrams at low order in $\lambda \Phi_1 \Phi_2^2$, $\lambda \Phi^4$, QED theories. For QED we have used the interaction $\lambda \Phi_1 \Phi_2 \Phi_3$, field Φ_1 corresponding to the photon, Φ_2 and Φ_3 to the electron. This scheme allows us to do oriented electron lines and to avoid the presence of "tadpoles" (i. e. of electron loops with an odd dot number). We know, by Furry's theorem, that these graphs have a zero contribution. We should also like to emphasize the fact that, by its construction, the program gives exactly the diagrams which appear in the S-matrix. For example, the photon-photon scattering graph at 4-th order appears 6 times, while the corresponding graph in $\lambda \Phi^3$ theory appears 3 times, because in this case lines are not directed.

In Table I, , we indicate some results. T is the machine time needed on 1108 Univac Computer.

REFERENCES

[1] Any textbook on quantum field-theory.

[2] For general properties of graphs, see
BERGE, C., Théorie des graphes et ses applications, Dunod (1967), Paris.
HARARY, F., Graph Theory and Theoretical Physics, Academic Press, London and New York (1967).

[3] VISCONTI, A., Théorie quantique des champs, tome II, Gauthier-Villars et Cie., Paris (1965).

[4] More details on the FRENEY Program, and the listing, can be found in PERROTTET, M., Thèse de 3ème Cycle, Marseille (1970).

COMPUTER SOLUTION OF SYMBOLIC PROBLEMS IN THEORETICAL PHYSICS *

A. C. HEARN
Department of Physics, University of Utah,
Salt Lake City, Utah,
United States of America

Abstract

COMPUTER SOLUTION OF SYMBOLIC PROBLEMS IN THEORETICAL PHYSICS.

A survey of the computing techniques currently available for the solution of non-numerical problems in theoretical physics and related areas is presented. The subject is considered in three general sections. In the first section, a comparison is made between the more familiar numerical calculation by computer and the analogous algebraic calculation, and the limitations of the latter are discussed in detail. Several systems currently available for performing such algebraic calculations are reviewed. A survey is also made of some of the techniques which are presently under development but not yet generally available. In the second section, a detailed examination of a specific problem in algebraic simplification, namely polynomial manipulation, illustrates some of the programming techniques which are used in this field. Several specific programming examples are presented and some current system implementations are discussed in detail. In the final section, a more general symbolic problem is discussed, namely the computer analysis of physical theories represented in terms of diagrams. It is also discussed to what extent a completely automated computer calculation of an analytic expression for a physical cross-section or related parameter in quantum electrodynamics is possible from a diagrammatic specification of the problem.

INTRODUCTION

It often comes as something of a surprise to physicists used to thinking of a computer as a numerical or control device to learn that it can also be used to perform algebraic and more general symbolic calculations. In particular, much of the theoretical progress in our understanding of quantum electrodynamics during the last few years has been helped by the use of computers to perform the tedious gamma matrix algebra involved in these calculations. In this paper, I shall survey some of the applications of symbolic computing techniques which have been made to physical problems. Because the field is extremely broad, however, I cannot hope to cover every application and so I shall limit myself to a selection of topics which reflect my own research interests. These topics are considered under three general section headings. Section 1 is intended for the casual reader who wishes to obtain a rough idea of the extent to which the computer can be used to carry out algebraic calculations. Section 2 is for those who want to know in greater detail how the computer actually does an algebraic calculation. This is illustrated by considering in detail a particular aspect of algebraic simplification, namely polynomial manipulation. Finally, in Section 3, a more general symbolic problem is discussed, namely the computer analysis of physical theories represented in terms of diagrams.

* Work supported in part by the National Science Foundation under Grant No. GJ-32181. Computer time supported by the Advanced Research Projects Agency of the Office of the Department of Defense under Contract No. F30602-70-C-0 300 at the University of Utah.

1. COMPUTER SOLUTION OF ALGEBRAIC PROBLEMS IN THEORETICAL PHYSICS

1.1. Preliminary

Because the idea of doing algebra by computer is so unfamiliar to most physicists, I shall begin by comparing a simple FORTRAN calculation with a similar algebraic one. The example I shall take is the computation of Legendre polynomials by the use of the following standard recurrence relation found in any suitable textbook:

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$P_{n+1}(x) = ((2n+1) \times P_n(x) - n P_{n-1}(x))/(n+1)$$

A possible FORTRAN program for doing this with the argument x = 1 is as follows:

```
      DIMENSION P(5)
      P(1) = 1
    1 X = 1
      P(2) = X
      DO 10 N = 1, 3
      P(N+2) = ((2*N+1)*X*P(N+1)-N*P(N))/(N+1)
      N1 = N+1
      N2 = N+2
   10 PRINT 20, N1, P(N2)
   20 FORMAT (3H0P (, I1, 4H) =, F4.1)
      END
```

Apart from a possible change in the PRINT statement, this program will run correctly under most FORTRAN systems, and the result in this case will be as follows:

```
P(2) = 1.0
P(3) = 1.0
P(4) = 1.0
```

Let us now see what happens if we leave out the statement numbered 1 in our program. Most FORTRAN compilers will now immediately give us the error diagnostic that X has no value. However, suppose our FORTRAN system was sufficiently general that it could represent the variable X as a Hollerith constant and carry out polynomial operations in the unknown X as it calculates the higher-order polynomials. Then, with an appropriate change in the format statement, our results might come out like the following:

```
P(2) = 3/2*X**2 - 1/2
P(3) = 5/2*X**3 - 3/2*X
P(4) = 35/8*X**4 - 15/4*X**2 + 3/8
```

This example shows us one of the basic differences between a numerical calculation using FORTRAN, for example, and an algebraic calculation in which indeterminants such as X which have no pre-assigned value can appear in the calculation. However, there are many other differences between systems for doing algebra on the one hand and arithmetic on the other. We discuss these in the next subsection.

1.2. Differences between algebraic and numeric calculations

Some of the major differences between algebraic and numeric calculations by computer are summarized in Table I. Most of the entries in this table are self-explanatory, but it is probably worthwhile to discuss them in more detail. The first point is that most numeric systems are now very well documented and widely available. For example, standards exist on the form of FORTRAN, and if they are followed, the chances are that the program will run on a random computer. On the other hand, algebraic systems are experimental in nature, subject to constant change and, except in a very few cases, available only on specific computers. The second major contrast is in the method of operation of the system. A FORTRAN or ALGOL system, for example, is usually organized in three stages. In the first stage, the user program is compiled into assembly language. In the second stage, a system loader loads the program into computer memory and, finally, the program is executed in the GO step. This compile-load-go mode of operation is particularly well suited to batch operation of computers and makes the most efficient use of computer core, as the compiler is not required, for example, after the first stage. Algebraic systems on the other hand are quite often designed to be interactive so that the user

TABLE I. DIFFERENCES BETWEEN NUMERIC AND ALGEBRAIC SYSTEMS

NUMERIC	ALGEBRAIC
WELL DOCUMENTED	EXPERIMENTAL
WIDELY AVAILABLE	LIMITED DISTRIBUTION
SMALL SYSTEMS external libraries	LARGE SYSTEMS internal libraries
C-L-G (compile-load-go)	INTERACTIVE/INTERPRETIVE
FIXED SPACE REQUIREMENTS	UNKNOWN SPACE REQUIREMENTS
CALCULABLE TIME REQUIREMENTS	UNKNOWN TIME REQUIREMENTS
POWER GROWTH RATES	EXPONENTIAL GROWTH RATES
FIXED OUTPUT SIZE	UNKNOWN OUTPUT SIZE

communicates with the computer statement by statement rather than in one complete program. They also tend to be interpretative rather than compiled in the sense that the system will interpret each statement given by the user and only compile it if asked by the user. Because a large number of system subroutines are necessary for algebraic calculations, a major part of an algebraic system must be kept in core or easily accessible during the whole calculation. It may even be necessary to keep a compiler resident or available during execution if it proves necessary to compile some of the statements in the interactive dialogue. Some of these subroutines (and a compiler is one) need not be in core all the time, but, as I shall explain later, for reasons of operating efficiency it is usually necessary to keep a relatively large number of these routines in core during execution. Numerical systems also have a large system subroutine library at their disposal, containing elementary functions such as sine and cosine, or more sophisticated routines such as those for Bessel function evaluation or numerical integration. However, it is already known at the loading step which routines are needed in a compile-load-go system, whereas the system does not know which routines are needed in advance of the execution steps in an interactive calculation.

The third major difference is that, once the numeric program has been compiled, the amount of memory necessary for execution can be calculated, and the system can therefore decide whether there is sufficient space in the computing system to run the job before execution starts. In algebraic systems on the other hand, the algebraic expressions must often be stored in core during the calculation and one cannot predict in advance how large these will be. It is therefore possible for an algebraic calculation to terminate during execution because the amount of core necessary for the calculation exceeds that available in the system. The same thing can be said about the time requirements for a calculation. For the numeric program, sufficient research in the mathematical theory of computation has been performed to enable one to gauge relatively accurately how long a calculation will take. On the other hand, the techniques for symbolic calculation are so new that one cannot make such accurate time estimates, and in fact the same calculation may take orders of magnitude longer if an inefficient organization step was introduced during the calculation. Related very closely to this point is the fact that most numeric calculations follow simple power growth rate laws. For example, the calculation of the inverse of a matrix is a process whose time requirements grow roughly as n^3 with the order n of the matrix. On the other hand, the time requirements for algebraic calculations such as polynomial manipulation can grow at exponential rates with the size of the expression. Thus, relatively minor changes in the organization of such a calculation can have dramatic effects on the time which the calculation takes. For these reasons it is necessary to design the algorithms for symbolic computation very carefully if one is to complete calculations in an economic time.

A final point is that whereas one knows in a numerical calculation how much output one can expect in a normal calculation, it is impossible to predict in many cases how much output a symbolic calculation will produce. For example, the determinant of a symbolic matrix of order n may contain $n!$ terms if no cancellations occur. Thus, the calculation of the determinant of a modest-sized matrix can astonish the user by producing output containing thousands and thousands of terms covering many pages of paper.

TABLE II. SOME ALGEBRAIC SYSTEMS IN CURRENT USE

SYSTEM	DATE	COMPUTER	APPLICATIONS
ALPAK[a]	~1960	IBM 7090	✓
7090 FORMAC[a]	1963	IBM 7090	✓
7090 SCHOONSCHIP[a]	1964	IBM 7090	✓
MATHLAB	1965	7090/PDP-6/10	X
REDUCE 1[a]	1965	PDP-6/10, IBM 360	✓
6600 SCHOONSCHIP	1966	CDC 6600	✓
PL/I FORMAC	1966	IBM 360	✓
SYMBAL	1967	CDC 6600	?
SAC	1967	most	?
REDUCE 2	1969	many	✓
CAMAL	1969	ATLAS II (TITAN)	✓
IAM	1970	PDP-10	?
MACSYMA[b]	?	MIT PDP-10	X
SCRATCHPAD[b]	?	IBM 360	X
ALTRAN[b]	?	most	X

[a] obsolescent

[b] under development

I have spent some time going through these differences simply to convince the intended user that symbolic computing techniques may not be the panacea for all algebraic problems. One has to exercise a certain amount of restraint in choosing problems to solve by these means. On the other hand, we do see an increasing number of successful applications of such techniques to realistic problems, and I would predict that in the next few years a whole new range of calculations may open up because of these methods.

I have mentioned that there are systems for performing algebraic calculations by computer, and in Table II some (but by no means all) examples of algebra systems are listed which have been developed over the last ten years and which offer the user facilities with varying degrees of sophistication. Several of these programs are now obsolescent but do have historical importance for developments in the field. Anyone interested in doing research in this area should certainly investigate the capabilities of these various systems, and one good source of information is Ref. [1].

As far as Table II is concerned, I think that one of the most important columns from a user's point of view is the last, which lists whether the system has actually been used for any published applications. In point of fact, only CAMAL [2], FORMAC [3], REDUCE [4] and SCHOONSCHIP [5] have been used in any significant number of published calculations so far. Even more interesting is the fact that three of these systems have been developed by physicists! (FORMAC was not). Those systems in Table II which are not referenced specifically are described in Ref. [1].

1.3. General characteristics of algebraic systems

So far, we have been concerned with the differences between systems designed for non-numerical algebraic calculations and those designed for numerical calculations. However, suppose that after all my warnings you decide that you want to use one of the programs listed in Table II. What general characteristics can you expect to find in these programs? A discussion of this subject falls naturally into four subsections, namely:

(1) The form of the language by which the user communicates with the computer.

(2) The general capabilities such as polynomial manipulation, pattern matching or substitution, for example, offered to the user of the program.

(3) The specific packages provided by the system or available from other users.

(4) The man-machine interface, in other words, the ease or lack of it in actually communicating with the computer.

Since the subject of language design is covered so well by the paper of Campbell [6], I shall merely outline the main problems here. Users are usually offered a complete system for algebraic manipulation rather than a set of subroutines in a program library. Such systems usually adopt a programming style which is similar in form to one of the commonly used numerical programming languages such as FORTRAN, ALGOL or PL/1. Some of these, such as FORMAC, are imbedded directly in the numerical system (in this case PL/1). Others, such as REDUCE 2, have a top level ALGOL structure but are imbedded in another language (LISP in this case). Finally, there are systems, such as SCHOONSCHIP, which are written mainly in assembly language, but which offer the user an ALGOL- or FORTRAN-like style (in this case FORTRAN).

All systems offer users commands which are unique to algebraic manipulation and thus they provide extensions of the numerical language style adopted. The most powerful systems offer the user the ability to extend the language further, or modify the source language directly. For example, in the case of REDUCE, the source language in which the program is written is the same as that used by the user when he writes programs. The point is that any one system is not all things to all people, and so a user who is offered a system incapable of easy modification is bound to be frustrated eventually.

1.4. General capabilities available in algebra programs

No matter how flexible or beautiful the language offered by a system to the user, its ultimate success in solving problems really depends on the

general capabilities which the designer built into his program and the efficiency they possess in execution. The systems presently available offer such a variety of capabilities that I would like to discuss those of most importance to us in physics. Let me begin by making a few comments about my distinction between what I call general capabilities and what I call specific packages. And again an analogy with the way that, say, FORTRAN operates is probably the easiest to use. When a FORTRAN program is compiled, the FORTRAN compiler, which is loaded into core, knows enough about the syntax of common source expressions to generate the required assembly code. In other words, the compiler contains at a very basic level a syntax analyser and possible optimizer designed to generate efficient code for an expression such as A + B - C. On the other hand, if a user requests a call to the function COS, then most compilers reference this function by leaving instructions for the loader to load this function from secondary storage. If the routine required is not available in this manner then the user must supply it himself. Thus the Bessel function may be known to the FORTRAN compiler but not the Gegenbauer polynomial. I refer to these latter examples as specific packages because they are not always required during the average calculation and therefore need not always be in core during the execution process. However, the syntax analysis which I discussed earlier is an example of what I call a general capability in the sense that it is always needed during compilation and it therefore becomes inefficient to segment this part of the program and swap it in and out of secondary storage. The same thing is true to a larger extent for algebraic calculations, but in this case, the decision as to what is a general capability and what is a specific package is not so clear cut. However, there are certain procedures which appear to be so fundamental that it becomes impossibly inefficient if these are not kept in core or easily accessible during a calculation. These are the general capabilities that I now wish to discuss; a list of them is given in Table III.

The most basic of all these general capabilities as far as an algebra program is concerned is polynomial manipulation. By this I mean the ability of a program to expand polynomials, collect like terms, order the terms, and so on. Because these techniques are so important, Section 2 is devoted to an account of the most common methods used today. In addition, by my discussing this particular problem in depth, you will be able to appreciate the type of programming techniques which are necessary throughout this field.

A surprisingly wide range of algebraic problems can be solved by a system which contains routines for polynomial manipulation plus some capability for elementary function manipulation, substitution and differentiation. In fact, a system such as FORMAC is of this type. However, in my mind the next logical general capability to be considered as part of a complete system for algebra is the ability to manipulate rational functions. At first sight, it might seem that all this requires is the ability to represent the ratio of two polynomials. However, a new important technique is required in this case, namely, the ability to compute the greatest common divisor of two polynomials. For example, the expression (A**2+2*A*B+B**2)/(A**2-B**2) is obviously less aesthetically pleasing than (A+B)/(A-B). What we have really done in going from the first to the second expression is to recognize that the polynomials have a greatest common divisor of A+B and divide it out. This is a less complicated process

TABLE III. GENERAL CAPABILITIES OF AN ALGEBRA SYSTEM

POLYNOMIAL MANIPULATION
RATIONAL FUNCTION MANIPULATION
POLYNOMIAL EXTENSIONS
SUBSTITUTION
STRUCTURAL EXTENSIONS

than factoring the polynomials. To see that this is true, consider the numerical analogue. The greatest common divisor (gcd) of two numbers can be found quite readily by Euclid's algorithm, which is well known to any mathematics student. On the other hand, the factorization of a number into its prime irreducible components requires a more complicated sieve process, and from a computational view is more time-consuming. The same is true in the algebraic case.

It turns out in fact that one can also extend Euclid's algorithm to polynomials. However, whereas the numerical Euclidean algorithm will find the greatest common divisor of two numbers relatively quickly, the analogous algebraic algorithm can be excruciatingly slow in some cases [7]. For example, it might take a day of computation time and an extremely large core on a CDC 6600 to find the greatest common divisor of two polynomials of degree 100, say, by Euclid's algorithm. Although polynomials of such a large degree are not normally encountered in the day-to-day work of a physicist, we do unfortunately tend to use a large number of variables, and it turns out that the degree of complexity of the calculation is roughly equivalent to the number of variables multiplied by the highest power of all variables. Consequently, the calculation of the greatest common divisor of two polynomials involving ten variables, in which the highest degree is 5, turns out to be at least as complicated as the computation of the greatest common divisor of two polynomials of degree 50 in one variable.

Because of the extreme inefficiency of Euclid's algorithm for such calculations, much effort has been devoted over the last five years to finding a more efficient algorithm. Recently, it has been found that one can exploit some nineteenth-century mathematics to achieve a much faster calculation. What one does is to use finite field or modular arithmetic. In other words, one calculates the greatest common divisor of two single variable polynomials in which the integer coefficients are considered in a finite field. Thus, if one was working, say, modulo 7, then the only integers one would consider would be 0 through 6, and 6+2, for example, would be replaced by 1. The computation of a greatest common divisor using modular arithmetic takes significantly less time than the analogous calculation over the integers. The modular calculation becomes relevant because of a very simple theorem which says that if two polynomials are relatively prime (i.e. have a greatest common divisor of 1) over a prime modular field then they are relatively prime over the integers. Since two

polynomials more often or not are relatively prime, one can determine this fact very quickly by using finite field arithmetic. If the two polynomials are not relatively prime, then it is possible by a straightforward construction procedure to determine the gcd from various prime field results. (If anyone is interested in reviewing the history of the interesting search for a more efficient gcd algorithm, he is referred to Ref. [7]. An account of these techniques can also be found in an excellent book by Knuth [8].) The results can be dramatic. For example, Loos [9] has programmed the latest gcd algorithm in REDUCE and we find that the gcd of two polynomials of degree 100 can be found in a matter of seconds now. In terms of the computing time necessary, the new algorithm represents a dramatic improvement over the Euclidean algorithm; the latter has an exponential growth rate, whereas the computing time for the new algorithm grows at a rate less than n^3 with the degree n of a univariate polynomial. This again illustrates that, in symbolic manipulation, it is very important that the algorithms be designed as efficiently as possible if realistic calculations are to be possible. No matter how large the computer is, the exponential growth rate of the Euclidean algorithm limits drastically the class of problems which one can solve by using it.

Polynomial extensions. There are very few calculations in physics which do not require the introduction of at least the elementary functions such as sine and cosine in the calculation. Therefore, most systems provide the user with certain of the elementary functions together with their common properties. The better systems also allow the user to add their own functions and define rules for manipulation. One way of introducing these functions is to regard those residual forms which result after simplification rules have been applied as polynomial variables and manipulate them as such in calculations. For example, an expression such as cos(x+y) might be best treated by expanding it by the rule cos(x+y) = cos x cos y - sin x sin y. In other calculations it might best be left alone. In the former case, residual expressions cos x, cos y, sin x and sin y result from such expansions, whereas in the latter case cos (x+y) remains as a residual form. My point now is that these residual forms can be considered as variables as far as polynomial and rational function manipulation is concerned. For this reason, I regard the introduction of such functions as polynomial extensions. REDUCE, for example, considers them in this manner. This is not the only way to treat elementary functions, however, and some systems might treat them in an entirely different manner. This is one place where one can see the wide differences in the philosophy concerning representations of expressions which exist among the manufacturers of the various algebra systems mentioned. There is a tremendously wide range of views on this subject; those interested in learning more about this are referred to the review article by Moses [10].

Substitution. There are very few symbolic problems which do not require some substitution capabilities for their solution. Such substitutions might be a simple as

"replace all occurrences of m by 0"

or involve complicated conditions such as

"for all values of x and y, replace cos(x)*cos(y) by
(cos(x+y) + cos(x-y))/2"

or

"replace $\cos(x)^2 + \sin(x)^2$ by 1 for all values of x"

or again,

"for all p, μ and ν, replace $p_\mu \delta_{\mu\nu}$ by p_ν"

or

"if n = 0 then replace fac(n) by 1, otherwise
replace fac(n) by the value of n*fac(n-1)".

It is obvious that if a sufficiently rich computer language existed for expressing such replacements, plus a system which could implement them, then most algebraic problems for which a known algorithm exists could be solved by such a system. Most of the algebraic systems now available offer the user facilities for introducing some rules of this type. In addition, a whole class of rules, such as those which define Dirac gamma matrix algebra, for example, may be offered as an integral part by the system builder. In the latter case, the rules are usually compiled into efficient machine code procedures which can exploit knowledge of the explicit internal representation of expressions and so gain efficiency in execution. On the other hand, most systems which allow user-introduced replacements offer interpretative evaluation of these rules, in that the parts of the expression under consideration are matched, identifier by identifier, against the rules introduced by the user. This technique is obviously less efficient than the use of compiled code, but it offers greater flexibility to those users who wish to change rules during a calculation. However, experience has shown that at least the basic rules for polynomial and rational function manipulation must be compiled if calculations are to be finished in an economic time. This point was best proved by an experimental system [11] which did in fact perform all algebraic manipulations by interpretative matching against rules introduced by the user. This system was so general that even a rule for the collection of equal terms as simple as x + x = 2x must be specified by the user. Such generality led to impossibly inefficient calculations, and therefore any system designed for practical use must offer a nucleus of compiled procedures in addition to efficient techniques for interpretative pattern matching and substitution. The ideal system would of course allow for the replacement of any given expression f(x, y, . . , z) by another expression g(x, y, . . . , z) where the parameters x, y and z may stand for constant expressions or have various conditions imposed on them. As can easily be seen, all the examples which we gave earlier of possible general substitutions are of this type.

This general matching problem, which I have discussed in several publications [12, 13], has still not been solved efficiently enough for use in large-scale calculations, and as a result most systems compromise

at some point in the type of substitutions allowed. Probably MACSYMA comes closest to providing completely general substitutions [14] although REDUCE allows a wide range of possible rules. Apart from these two systems, however, the possible substitutions allowed by most other systems are much more limited, and this does tend to limit their effectiveness in some calculations.

Structural extensions. The capabilities discussed so far, together with, say, a differentiation package which I shall consider in the next section, make up quite a powerful algebraic manipulation system. However, unless the user has relatively straightforward problems, he will quickly discover that he wants to add new facilities to the program or even change the way that something as basic as a polynomial is treated by the system. Thus the final general capability which I think the good algebra system must provide is the ability to perform structural extensions, in other words, the user should be able to change the algorithms rather straightforwardly or to add new ones to suit his own purposes. This problem is more one of language design than of capabilities per se because if the program is written in a modular way and the user language is related in a straightforward way to the language used for the implementation of the system, then the informed user can make such changes. However, as this question must be considered when one designs the algorithms for performing the general capabilities discussed, I have included it in this section for completeness.

1.5. Specific packages

Once a system has been built which includes the general capabilities discussed in the previous section, it becomes a relatively straightforward matter for the system builder and often for the user to add the routines necessary to perform the specific calculations of interest to him. A list of some such possible specific capabilities or packages which might be necessary is given in Table IV.

I am sure that there are some system builders who would consider the first such facility in the list, namely differentiation, as belonging to the previous section. The point is that the rules for differentiation are so easy to define that the computer implementation is a very straightforward matter. In fact, to my knowledge, the very first computer programs written for performing algebraic manipulation were differentiation programs [15,16]. These were, of course, very rudimentary by our present-day standards but represented a tremendous advance in conceptual thinking about computers twenty years ago. Differentiation is now considered so easy to perform by computer that it is usually given as an exercise in classes on non-numerical programming.

It is interesting to observe that the ease with which functions can be differentiated symbolically by computer contrasts markedly with the numerical analogue. Numerical differentiation is a relatively difficult process because it tends to accentuate irregularities in the function being differentiated, and therefore, if there is any significant statistical error in the input data, the resulting numerical derivative can be quite inaccurate. The opposite is true of integration. Because it is a smoothing process, numerical integration has been one of the great successes of computerized numerical analysis, whereas symbolic integration is an extremely difficult

TABLE IV. SPECIFIC CAPABILITIES OF AN ALGEBRA SYSTEM

DIFFERENTIATION
INTEGRATION
GAMMA MATRIX ALGEBRA
TENSOR ALGEBRA
GENERAL NON-COMMUTATIVE ALGEBRA
POLYNOMIAL FACTORIZATION
MATRIX MANIPULATION
ASYMPTOTIC OPERATIONS
TRANSFORM TECHNIQUES

and complicated process to perform by computer. However, it represents one of the most interesting research problems in algebraic manipulation at the moment.

Apart from two pioneering experimental programs for analytical integration [17, 18], most symbolic integration by computer has been done by table look-up or pattern matching. This latter method has of course been very useful in several important quantum electrodynamics (QED) calculations, but it is important to emphasize that in this case we are simply using the computer as a book-keeping device in order to make the substitutions for the various integrals which arise in the calculations. More important, one needs to know the integral of every basic function encountered in order to carry out the integration. If we meet a function whose integral has not been previously computed, then either we must sit down and do it by hand or leave it in an undetermined form. Current work in this area promises techniques for the symbolic integration of indefinite integrals by essentially algorithmic means. This means that the program, when confronted with a new integrand never seen before, will actually find an analytical form for the integral if it exists in the class of functions that we are considering. This work is based on some important theoretical results by Risch [19] who showed that a complete algorithmic decision procedure exists for the evaluation of integrals for most elementary functions including logarithmic or exponential extensions, provided that the integral exists as a member of this class. The algorithm also decides if the integral does not belong to this class. Thus, to give two simple examples, this algorithm can determine the integral of cos x, since both integrand and integral are composed of exponentials. On the other hand, the algorithm will decide that the error function $\int \exp(-x^2)\, dx$ is not an elementary function.

Risch has recently extended his ideas to cover nearly all functions which a physicist is likely to meet in day-to-day analysis. It should now be possible, for example, to produce a program capable of integrating all

single variable functions which one meets in most physical calculations, and several groups are attempting to develop such programs. Whether these ideas can be successfully extended to multidimensional integrals remains to be seen, but one can of course be optimistic. However, it is clear that we now know enough about analytical integration to be able to perform a complete symbolic computation of any fourth-order process in QED, and several groups, including my own, are studying this problem in detail. I shall say more about this in Section 3. For those interested, a recent survey paper outlines some of the problems which arise in implementing Risch's work [20].

It may come somewhat as a surprise to readers to find that I have listed gamma matrix algebra and tensor manipulation as specific packages. From a historical point of view, the need to calculate very complicated processes in QED motivated the development of many programs for solving these specific problems several years ago. Two such programs, namely Veltman's SCHOONSCHIP and my own REDUCE, which were originally developed for this purpose, remain in wide use today. Although these two programs began as specific systems for solving problems in high-energy physics, it was quickly recognized by Veltman and myself that the techniques employed were quite general, and thus the programs evolved into general-purpose algebraic manipulation systems. Several other programs for gamma matrix algebra which appeared at about the same time have not survived because of their lack of generality or poor language design. What we realize now, as I have said earlier, is that the key capability in such programs is polynomial manipulation, and once one has an understanding of how to represent polynomials and polynomial extensions in the computer, the addition of routines for solving the specific problems associated with gamma matrix algebra and tensor manipulation is straightforward. One can build quite respectable routines for this purpose by simply programming the algorithms one finds in standard textbooks on quantum electrodynamics such as that by Bjorken and Drell [21]. However, there are algorithms more suited for computer evaluation of traces of gamma matrices, and, in particular, one by Kahane [22] represents a distinct advantage over those found in textbooks. Recently, Chisholm and I developed a new algorithm for trace calculation [23] which extends that developed by Kahane and which, although rather complicated to describe, should be relatively straightforward to program for the computer. We believe that this new algorithm will lead to further efficiencies in trace calculation and also reduce the amount of computer code necessary. We may thus reach the point where the amount of code necessary to provide for these facilities is a very small part of the whole system. This is rather ironic when you consider that REDUCE was originally regarded as a program for taking traces of Dirac gamma matrices!

Many other useful facilities such as polynomial factorization, general matrix manipulation, asymptotic operations and the manipulation of expressions by transform techniques become possible, given the basic facilities described in the previous section. If the language is powerful enough, then the user can add such facilities himself in terms of the primitives provided to perform the basic operations. However, it should be emphasized that there is no system yet in existence which can provide all these techniques, although they will surely become available in the next years.

1.6. Man-machine communication

The fourth and last general feature of algebraic simplification systems which I wish to discuss is the question of man-machine communication. By this, I mean the way in which the user communicates with the program both in terms of input of problems and output of results. Although the program language itself may provide specific operations for input and output, this feature really depends much more upon the computer available than the algebra program itself, although the latter must be organized in a way to take account of interactive facilities if these are available.

A discussion of the complete input-output problem could again occupy a whole paper and so I shall limit myself to a few general observations. First, our experience has shown that painless algebraic calculations from the user's point of view can only be accomplished in a computer with a high speed memory of 40 000 words or more. Of course, useful calculations have been performed in smaller memories, and a clever programmer can make optimum use of whatever facilities he has at his disposal. However, the average user who does not want to involve himself too heavily in questions of system design will find himself in a constant fight with the operating system on a small computer in order to get the space he needs for his calculations. I would therefore strongly advise anyone against attempting such calculations unless a large enough computer is available. Secondly, the most successful calculations have been made using time-sharing interactive computing systems, such as the system available on the Digital Equipment Corporation PDP-10 with which I am most familiar. A complete algebraic calculation depends so much on the knowledge gained at each step in the calculation that constant man-machine interaction is necessary. This can only be done properly if the need for resubmitting a job at each step is eliminated, as is only possible in an interactive system. Within such an environment a cathode-ray tube display and keyboard backed by some method for making hard copy is the best terminal available today. In fact, by using displays with line-drawing capabilities it is possible to display symbolic output in a two-dimensional textbook form, which is of course a much more readable and informative presentation of results than can be achieved by line printers or teletypes. Although such terminals are quite expensive at present, new models are now appearing which should be well within the budget of most computer centres. In addition, cheap hard copy of the display images by xerographic means is becoming available.

Finally, we have the problem of input. Our natural two-dimensional representation of mathematical equations must be converted to a linear form for computer input using presently available hardware. However, some promising research in progress on the use of input tablets or light or sonic pens points to a future time when we can input our problems direct form our notebooks in standard textbook notation. Programs for doing this already exist, but they are too slow and cumbersome for general use at present

2. POLYNOMIAL MANIPULATION

2.1. Preliminary

Because of its extreme importance as the key capability in algebraic simplification, I would like to spend some time considering techniques

which have been developed for manipulating polynomials by computer.

I shall consider the problem under three headings, namely, polynomials with no variables, polynomials of one variable, and polynomials of many variables. In the context we are discussing here, a polynomial with no variables is of course a number, and one might think at first sight that this is a trivial case which need not be discussed. However, it is important to realize that in an algebraic calculation, where one is concerned with exact results, it is more suitable to consider numbers as integers, or ratios of integers, rather than as floating point numbers with their inherent inaccuracies. For example, an algebraic calculation whose result is zero is significant because one then has an exact proof that the relevant expression is identically zero. On the other hand, if the result has a floating point value of 10^{-38}, can one then say that the result is still really zero? In order to ensure uncompromising accuracy in calculations, then, so that such questions can be resolved, most systems provide for users to introduce not only integers but rational numbers as the ratio of two integers. Algebra programs usually also allow floating point numbers by way of compromise because engineers typically like to introduce numbers in this form! If one is working with irrational numbers it is better to introduce them as symbols. In other words, an algebraic result $3\pi^2-10$ is far more meaningful than an equivalent floating point expression.

Once one has decided in a given calculation to do all arithmetic exactly, a problem arises if the integers one is working with grow larger than the word length of the machine being used. One can of course resort to double or triple precision arithmetic, but this again imposes an ultimate limit on the size of the numbers which can be handled. It is obvious that one needs unlimited precision arithmetic for complete accuracy. In other words, you must use a representation for numbers which does not limit you to any fixed size; if you do impose a limit, then the next calculation is always bound to exceed it. (This must be some form of Parkinson's Law!) In Ref. [8], Chapter 4, one finds a very elegant discussion of the problem of doing multiple precision arithmetic by computer, and I would certainly commend a study of this particular section to anyone interested. However, Knuth depends on the use of arrays for doing his calculations because he works in terms of fixed-sized, although large, multiple precision numbers rather than variable-size numbers. For arbitrary precision integers, a list-structured representation is usually employed [24].

In order to understand how one can do arbitrary precision arithmetic by computer, let us recall that integers are written as a sequence of digits with an implicit radix. Thus the decimal integer 10372 stands for the sum $2 + 7 \times 10 + 3 \times 10^2 + 1 \times 10^4$. The so-called "new math" teaches high-school students these days that the radix need not necessarily be ten, and one can have binary, octal, and so on, representations of numbers. Provided one knows the radix then, one could represent an integer with an arbitrary number of digits as a list of those digits, as in (1 0 3 7 2) for the above example. Such a list, of course, would provide a computer representation for a number of any size, and Lurié [25] shows how such lists can be represented in the computer. However, it is clearly inefficient to make each number in the list a single decimal digit. A more efficient representation results if we make the radix the size of a computer word so that the "digits" in our list become single precision integers. However, even though the individual elements of such a list $(u_1\ u_2 \ldots u_n)$ can now be quite

large integers, it is usual to refer to this as an n-place integer, and to each u_i as a digit with respect to the radix b we are using. However, if the radix is taken to be, say, 2^{36}, then we require a translation phase to show that the number (12345 7890) has the decimal representation 848341940313810, but this translation is quite straightforward. It also turns out to be easier to work with a list in which the last significant digit appears first on the list. Thus we represent the number $u = u_1u_2 \ldots u_n$ by the list $(u_n u_{n-1} \ldots u_1)$. The number 10372 with radix 10 would therefore be written as (2 7 3 0 1).

We still have to represent the sign of such an integer, and there are several ways of doing this. One easy way is to associate a sign indicator (e.g. POSNUM or NEGNUM) with the atom representing the arbitrary precision integer and make the above list structure the corresponding property. Once a convention such as this is adopted, one need only perform manipulations on the magnitude of the integers concerned, and I shall therefore consider only this from now on. There are also more efficient ways for representing such numbers internally in the computer, but I shall not discuss these here.

The routines for handling such arbitrary integers with precision are described in some detail in Ref. [24] and can be obtained from Knuth's multiple precision routines by simple transposition. In order to illustrate a typical program which one could write, the list-processing equivalent of the array program which Knuth uses for multiplying two such numbers is given here. I shall write this in a form which mirrors Knuth's notation, so that anyone interested can compare this list-processing program with the array-processing program of Knuth. In a practical implementation of this algorithm, one would probably write as much as possible of it in assembly language in order to gain efficiency, but the form of the algorithm would be much as given here.

The algorithm we are considering takes two non-negative arbitrary precision integers $u = u_1u_2 \ldots u_n$ and $v = v_1v_2 \ldots v_m$ with radix b, written in our list notation as $(u_n \ldots u_2u_1)$ and $(v_m \ldots v_2v_1)$, and forms their product $w = w_1w_2 \ldots w_{m+n}$, written as $(w_{m+n} \ldots w_2w_1)$. The usual pencil and paper way of forming such a product is to calculate the partial products $u_1u_2 \ldots u_n \times v_j$ first for $1 \leq j \leq m$ and then add these together with an appropriate scale factor based on the radix. However, Knuth points out that in a computer it is best to do the addition concurrently with the multiplication as is done in the following procedure LMULT:

```
SYMBOLIC PROCEDURE LMULT (U,V);
   BEGIN SCALAR W,U1,W1,W2; INTEGER K,T;
    M1: W := NZERO(LENGTH U + LENGTH V);
        %we only really need to set first N elements to 0;
        W1 := W;
    M2: IF NULL V THEN RETURN W;
        W2 := W1;
    M3: U1 := U;
        K := 0; %this variable stores carry in M4 loop;
    M4: T := CAR U1*CAR V + CAR W2+K;
        K := T/B;
        RPLACA(W2,T-K*B);
```

```
    M5: U1 := CDR U1;
        W2 := CDR W2;
        IF U1 THEN GO TO M4;
        RPLACA(W2,K);
    M6: V := CDR V;
        W1 := CDR W1;
        GO TO M2
    END;

SYMBOLIC PROCEDURE NZERO N;
    %this function creates a list of N zeros;
    IF N=0 THEN NIL ELSE 0. NZERO (N-1);
```

Additional list-processing algorithms for addition and division of such numbers may be found in Ref. [24]. It is also a useful exercise to carry through the multiplication operations by hand using the above algorithm.

2.2. Univariate polynomial manipulation

The discussion of the computer manipulation of polynomials in one variable becomes trivial once we have considered routines for arbitrary precision integer arithmetic. It is easily seen that a polynomial such as $x^4 + 3x^2 + 7x + 2$ is no more than a generalized arbitrary precision integer in which the radix is now the variable x rather than the integer b. Thus, we could represent the above polynomial by a list representation (2 7 3 0 1) with the lowest power first and the variable name X implicit (it could be stored in a global variable VAR, for example). There is, however, an important difference between the integer case and the polynomial case, namely, that the carry in our multiplication algorithm is no longer present so that the algorithm is much simpler. Thus, a program for multiplying polynomials using the above list representation could be written as follows:

```
SYMBOLIC PROCEDURE PMULT (U,V);
    BEGIN SCALAR W,U1,W1,W2;
    M1: W := NZERO(LENGTH U + LENGTH V);
        W1 := W;
    M2: IF NULL V THEN RETURN W;
        W2 := W1;
    M3: U1 := U;
    M4: RPLACA(W2,CAR U1*CAR V + CAR W2);
    M5: U1 := CDR U1;
        W2 := CDR W2;
        IF U1 THEN GO TO M4;
    M6: V := CDR V;
        W1 := CDR W1;
        GO TO M2
    END;
```

We thus see that we can consider single variable polynomial operations as simply numerical operations on the coefficients. The transformation of the list of coefficients to, say, an ALGOL-like output form is then a simple matter of translation. However, there are some disadvantages in the above

approach. Quite often the polynomials that we encounter even in one variable are sparse in the sense that not all powers of each variable actually appear in the expression encountered. The above representation requires an explicit zero to be placed in the list for any power which does not actually appear in the polynomial under consideration. The polynomial $x^{100}+1$ would therefore have a representation consisting of a list of two ones enclosing ninety-nine zeros, which is clearly wasteful in space and ultimately in the time taken for manipulation. In order to see how we can get around this problem, we pass on now to multivariate polynomial representations, of which, of course, single variable polynomials are a sub-class.

2.3. Multivariate polynomial manipulation

In discussing representations of multivariate polynomials, I shall limit myself to those which are canonical. By this I mean that if we have two polynomials which are representations of the same function then they will have the same canonical form. If one surveys current systems, one finds that there are only two classes of such representations in general use. The first of these represents the polynomial $p(x_1, x_2, \ldots, x_n)$ in the distributive form

$$p(x_1, x_2, \ldots, x_n) = \sum C_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \tag{2.1}$$

where the C's are numerical coefficients. In the second representation, the polynomial p is written as a power series in one variable whose coefficients are functions of the remaining n-1 variables. Thus,

$$p(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{m_i} p_i(x_2, \ldots, x_n) x_1^i \tag{2.2}$$

The polynomial coefficients $p_i(x_2, \ldots, x_n)$ are themselves expanded as a power series in x_2 and this representation continues until only numbers remain. For example, the polynomial

$$P = (x+y)^2 + 3x - 2y \tag{2.3}$$

when written in the first representation has the form

$$1x^2y^0 + 2xy + 3xy^0 + 1x^0y^2 - 2x^0y \tag{2.4}$$

whereas in the second representation it has the form

$$(1y^0)x^2 + (2y + 3y^0)\,x + (1y^2 - 2y)x^0 \tag{2.5}$$

However, although there are only two such basic representations in general use, one can find as many ways of implementing them as there are systems! For example, the variables can occur explicitly in the representation, or be stored separately and occur implicitly. Similarly, variables raised to a zero power can occur explicitly or be omitted. Numbers can occur as integers, ratios of integers or real numbers. The choice of the leading variable in the second representation can be made in many different ways;

it might be introduced by the user, or it might be decided by some internal ordering in the system, and so on. However, it is important to recognize that neither representation requires every power of each variable between highest and lowest to occur as did our single variable polynomial representation discussed in the previous section.

Experimental analyses by various people have shown that in most algebraic calculations the second representation is more efficient in principle than the first in terms of time and space required to perform a given calculation. However, the differences in many cases can be very small, and so both representations have their champions.

In order to illustrate practical implementations of these representations, I shall consider one example of each type, although, as I said earlier, nearly every system in Table II could provide examples of one or the other representation.

As an example of a system which uses the distributive representation, ALPAK (and its newer version ALTRAN) is probably the best documented [26]. In this system, although list structures are used, they are not quite the LISP types which Lurié [25] has discussed. In the ALPAK case, a polynomial is represented by a basic unit of memory called a block which contains three pointers. The first points to a vector which contains a list of the names of the variables in the polynomial, the second points to a vector of coefficients, and the third points to an array of exponents. In the latter array, the columns correspond to the variables and the rows to the coefficients. The rows are ordered numerically. Thus, the polynomial in Eq. (2.3) has the representation shown in Fig. 1.

In this representation, operations on polynomials, such as addition and multiplication, become numerical operations on the coefficient and exponent arrays. Addition, for example, is a simple merging of the exponent arrays with appropriate adjustments to the coefficients when two exponent rows are equal. Full details on the implementation of these techniques may be found in the cited references.

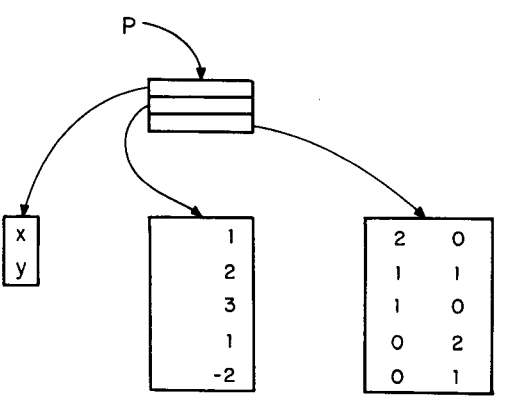


FIG. 1. ALPAK representation of the polynomial $P = (x + y)^2 + 3x - 2y$.

As an example of a system which uses the recursive representation, I would like to discuss the standard form adopted in the REDUCE system. Using the LISP dotted pair notation, the REDUCE standard form is defined by the following Backus-Naur syntax:

$$\langle\text{standard form}\rangle ::= () \mid \langle\text{integer}\rangle \mid (\langle\text{standard term}\rangle . \langle\text{standard form}\rangle) \tag{2.6}$$

$$\langle\text{standard term}\rangle ::= (\langle\text{standard power}\rangle . \langle\text{standard form}\rangle) \tag{2.7}$$

$$\langle\text{standard power}\rangle ::= (\langle\text{variable}\rangle . \langle\text{non-zero positive integer}\rangle) \tag{2.8}$$

Definition (2.6) means that a standard form is either an empty list (representing zero), an integer, or an expression consisting of a dotted pair of a standard term and a standard form. Similarly, a standard term is a dotted pair of a standard power and a standard form, and a standard power is a dotted pair of a variable and a non-zero positive integer. Thus a standard term represents one term in the power series Eq.(2.2), and a standard power represents a variable raised to a positive integer power. Comparison of Eq.(2.2) with the above equations also shows that the dotted pair represents an implicit addition in Eq.(2.6), multiplication in Eq.(2.7) and exponentiation in Eq.(2.8).

The BNF definition of the standard form is not complete until details of the ordering convention are specified. This is based on the machine location in core of the constituent variables (which are stored uniquely). Similarly, the order of standard terms in a standard form is based on the exponent of the leading standard power, highest exponent first. With these conventions, equal polynomials have the same standard form.

Using this particular representation, the polynomial in Eq.(2.3) would be written in the form (assuming X is ordered ahead of Y):

```
(((X . 2) . 1) ((X . 1) ((Y . 1) . 2) . 3) ((Y . 2) . 1) ((Y . 1) . -2))
```

Polynomial combination operations on such forms are again relatively easy to program because of the recursive nature of the representation. As an example of the type of programming which one finds in this case, a function ADDF which adds two polynomials represented in this form is given below. This particular function uses the auxiliary function ADDN for adding a number to a polynomial and a system function ORDP to define an ordering among expressions. The definition is quite straightforward, provided one is used to recursive programming, and it is another useful exercise for the reader to take two polynomials written in standard form and carry through the addition by means of this procedure:

```
SYMBOLIC PROCEDURE ADDF (U,V);
    IF NULL U THEN V
     ELSE IF NULL V THEN U
     ELSE IF ATOM U THEN ADDN(U, V)
     ELSE IF ATOM V THEN ADDN(V, U)
     ELSE IF CAAR U = CAAR V
```

```
    THEN BEGIN SCALAR X,Y;
        X := ADDF(CDAR U,CDAR V);
        Y := ADDF(CDR U,CDR V);
        RETURN IF NULL X THEN Y ELSE (CAAR U . X) . Y
    END
   ELSE IF  ORDP(CAAR U,CAAR V) THEN CAR U . ADDF(CDR U,V)
   ELSE CAR V . ADDF(U,CDR V);

SYMBOLIC PROCEDURE ADDN(N,V);;
  IF NULL V THEN N
   ELSE IF ATOM V THEN (LAMBDA M; IF M=0 THEN NIL ELSE M)(N+V)
   ELSE CAR V . ADDN(N,CDR V);
```

2.4. Infix-prefix translation

Once the system designer has decided upon a particular representation (or representations) for polynomials, such as those outlined here, there still remains the problem of converting a polynomial written in a FORTRAN or ALGOL-like notation into this internal representation and the translation of the results back into such an infix form. This particular problem is more one of syntax translation than algebraic simplification per se, but it is still part of the general subject of symbol manipulation with which this paper is concerned. However, a complete discussion of this subject would be a paper in itself, so I shall have to limit myself to a few general comments. Most systems begin by translating the input into an internal prefix form in ways similar to those described by Kuo-Petravić [27]. This prefix form is converted into whatever polynomial representation is used, and then translated back to prefix form and hence printed in infix form by similar techniques. I would, however, like to close by mentioning some interesting ideas which Petrick [28] discusses in a recent paper. He suggests the possibility of using a translator writing system (that is, a program which automatically writes translators for you) to develop a translator which goes direct from the infix input right to the standard form employed, thus bypassing the intermediate prefix step. His paper is eminently readable and uses the Weissman differentiation program [29], which Lurié [25] discusses, as an example of a problem where this method could be applied. I commend a study of Petrick's paper to anyone at all interested in working in this field.

3. SYMBOLIC ANALYSIS FOR PHYSICAL THEORIES REPRESENTED BY DIAGRAMS

3.1. Preliminary

There are many processes in physics and related fields which are represented in terms of diagrams. In quantum electrodynamics, for example, we classify parts of the perturbative expansions from quantum field theory in terms of diagrams, known as Feynman diagrams, which describe the evolution of the process with time. In electrical circuit theory, one represents the circuit by a diagram in which special symbols are used to represent the various components in the circuit, and the lines

represent the wires along which the currents flow. In operations analysis, one is concerned with depicting some process in terms of a flow chart which could describe, for example, at what times materials and workmen should arrive at a building site so that everything is there when needed, but not before (so that cost is minimized). Again, in statistical mechanics, one uses diagrams to describe terms in a perturbative approximation to a physical process. Similar examples may be found in many-body theory and the theory of turbulence, to name a few.

The usual way in which such diagrams are utilized is to perform a mapping on the diagrammatic structure so that a symbolic expression results whose value is a particular parameter associated with the diagram. For example, in QED, the matrix element (in the quantum mechanical sense) which corresponds to any diagram can be derived unambiguously from it by the application of a well-defined set of rules, known as Feynman rules. The matrix element so derived is now an algebraic expression which can be related by conventional techniques to a physically measurable quantity such as the cross-section of a process or the magnetic moment of a particle. In circuit analysis, one applies the standard network formulas to a given diagram in order to calculate, for example, the current of the outputs from the circuit. Such representations of processes are ideally suited to symbolic analysis by computer. However, the techniques which we use are now more general than those which we have discussed in the previous sections because our data are no longer simply algebraic expressions. In particular, the analysis of such processes in these terms requires the use of graph theoretical ideas in order to represent the diagrams and analyse their topological properties.

In point of fact, a considerable amount of graphical analysis has already been performed by computer. Most of this has been in operations research [30], although work on more general graph theoretical problems has recently been published. In particular, there is one system called GRASPE written in LISP [31] which analyses very extensively the topological properties of a graph. This particular system uses a very general representation for graphs and can answer quite complicated questions about graphs relatively quickly. On the other hand, most of the graphical work in operations research has used programs written in assembly language for analysing diagrams. At least one ALGOL system for graph analysis is also in existence [32]. However, what I would like to discuss here is how one can use general LISP processing techniques for the analysis of such problems, and I shall concentrate on describing methods used in one particular program developed by Campbell and myself [33] for analysing Feynman graphs in QED.

Feynman's method for calculating a cross-section from a diagram, as given in any standard textbook on relativistic quantum mechanics such as Ref. [21], consists of the following eight steps:

(1) Trace through the diagram all fermion lines which start from an external line and write down the contributions to the matrix elements as given by the Feynman rules as each line and vertex is encountered.

(2) Trace the closed internal fermion loops in the diagram and write down the Feynman contributions to the matrix element for each line and vertex.

(3) Add the contributions to the matrix element from each boson and any remaining (fermion-free) vertices in the diagram.

(4) Put in the overall factors for the diagram such as a factor of -1 for each closed fermion loop.

(5) Simplify the matrix element to the extent of collecting all commuting terms together and possibly simplifying some of the terms by the use of the Dirac equation.

(6) If internal loops occur in the diagram, it is now necessary to integrate over the unknown loop momenta.

(7) If there are any such loops, then one should carry out any renormalization or regularization required, in order to extract the physically meaningful part of the diagram.

(8) One now has a matrix element which can be algebraically manipulated further to give the required result in terms of the masses and scalar products of the momenta of the particles occurring in the diagram.

During the last few years, several people, including myself, have been concerned with the possibility of carrying through non-trivial calculations of this type completely by computer. Although we are not yet at the point where we do have a completely automated calculation, we have come a long way along the road to success in this area. It can be seen that algebraic calculations of the type discussed in the first section are just one aspect of this whole calculation. Even though the algebraic techniques can be employed at various steps in this process, it is really only at the last step, when one takes the matrix element and calculates a physical quantity, that the problem is clearly algebraic. In the other steps, one is dealing with more general symbolic data and, in particular, with graphs and their representations. Let me therefore begin by describing our method for representing Feynman graphs in the computer. This is not the only method which can be employed and the GRASPE representation [31] is far more general. However, the representation discussed below suits Feynman diagrams particularly well because one conventionally associates physical properties with the lines in the diagrams rather than the vertices and hence can consider the lines as the basic entities in the diagram.

3.2. Computer representation of Feynman diagrams

In the representation we have chosen, we represent the vectors or momenta of a diagram by variables (or atoms in LISP). Although the components of the vectors have physical relevance, it is unimportant, as far as the graphical properties are concerned, to show these explicitly. We give all vectors in the diagram an explicit orientation. Following standard practice, fermions are oriented in the direction in which the particle travels. We choose the orientation of bosons to be in the direction of increasing time. To say that a particle enters or leaves a vertex is now explicit, and we can associate with each vertex a list of incoming vectors and a list of outgoing vectors. We then define a vertex as a dotted pair of these two lists. Thus, in Fig. 2, which shows some of the Feynman diagrams which contribute to the well-known process of electron Compton scattering, vertex A in the first diagram has incoming vectors P1 and K1, an outgoing vector P3, and a vertex representation ((P1 K1) . (P3)).

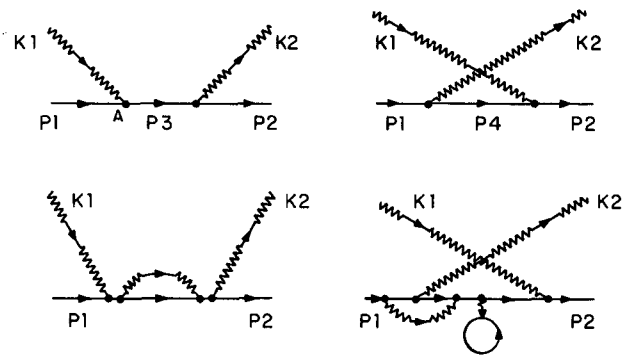


FIG. 2. Some of the Feynman diagrams which contribute to electron Compton scattering. Electrons are represented by straight lines and photons by wavy lines.

We can now define a diagram as a list of vertices and a process as a list of diagrams. The full representation for the first diagram in Fig. 2 is therefore (((P1 K1) .(P3)) ((P3) . (P2 K2))). This particular representation of a diagram requires no definite ordering of the vertices in the diagram list. Furthermore, all graphical properties of the diagram may be found by simple scanning of the representation. For example, those vectors internal to a diagram must occur in both the first half of one vertex and the second half of another. Therefore, if we scan the whole diagram and make a list of all vectors which enter each vertex, those vectors which appear twice in the list represent internal vectors. We can therefore find them by using the following three functions:

```
ALLVECTORS DIA := IF NULL DIA THEN NIL
                  ELSE APPEND(CAAR DIA, APPEND(CDAR DIA,
                                ALLVECTORS CDR DIA));

REPEATS X := IF NULL X THEN NIL
             ELSE IF CAR X MEMBER CDR X
              THEN CAR X . REPEATS CDR X
             ELSE REPEATS CDR X;

INTVECTORS DIA := REPEATS ALLVECTORS DIA;
```

Other properties of the diagram may be found in a similar fashion.

The various physical properties of the particles in a diagram may be included by using the property lists of the relevant atoms as described by Lurié [25]. For example, the statement

```
PUT ('P1, 'MASS, 'EM);
```

assigns the mass EM to the variable P1. However, in order to make it easier for the user of our program, a mass assignment can be made by the simpler declaration

```
MASS P1 = EM;
```

The command MASS has the effect of making a property list assignment of the above type. Other properties such as the spin of the particles, or whether the object is a particle or antiparticle and so on, may be declared in a similar manner.

Once we are able to determine the general properties of a diagram such as a specification of the inputs, internal vectors and so on, we are in a position to consider the various steps in the Feynman method for calculation listed above. I shall therefore consider these one by one and discuss to what extent it is possible to carry them through by computer.

Rather than build up the matrix element as we go, we have found it easier to trace out all the fermions and bosons in the diagram in a first pass and then apply the Feynman rules to the resulting list of paths in a second pass. The determination of these paths is quite straightforward. If we adopt the convention that all fermions have a flag FMN on their property lists, then it is simply a matter of scanning, say, the outputs of the diagram, finding each fermion and following it back through each vertex. We can then subtract those fermions encountered in this process from all fermions in the diagram. The remaining fermions must be part of closed loops, unless an error has been made in specifying the diagram, and we can therefore start with one of these and trace a closed path from it. This procedure can then be repeated until all remaining fermions are accounted for. Fermion conservation can of course be automatically checked as we scan the diagram. The remaining bosons and fermion-free vertices are finally found by a subtraction process.

3.3. Computation of the matrix elements

Now that we have the physical structure of the diagram mapped out, it is again relatively easy to determine the Feynman matrix element. The application of the Feynman rules to calculate the physical matrix elements from a diagram is in fact a good example of a unique mapping of one data space onto another. The rules which determine this mapping are easy to specify, and the result of the mapping depends only on the properties of the particles and vertices in a diagram. From a system design point of view, there are two problems which must be considered; first, the specification and storage of the rules themselves, and secondly, the application of the rules to a given diagram.

To specify a Feynman rule for an arbitrary line requires information on the particle type and also whether it is internal or external to the diagram. Except for the ambiguity between the photon and neutrino, it is sufficient to identify a particle by its status (external or internal) and mass to specify a rule uniquely. To include the photon and neutrino in this scheme, they are given the symbolic masses WLA and WNU respectively. Adjustments are made in the matrix routines so that these symbols are replaced by zero when they occur in expressions. Other particles are given mass attributes which reflect their conventional labellings.

A general rule for a line also contains information on the momentum, polarization vectors and indices of the particle, which are not known at the time the rule is specified. To allow for this, the program accepts the atom MOM as a (dummy) representation for the momentum of the particle, the atoms IN1, IN2, etc., for indices, and POL1, POL2, etc., for polarization vectors. In specifying rules for lines involving gamma matrices and

spinors we also use a dummy line identification symbol LN as the first argument of the relevant operator. This symbol is required in general in order to distinguish between different fermion lines in a diagram. Having adopted these conventions, a rule for a line in a diagram is specified in the form

$$L(\langle \text{mass symbol} \rangle, \langle \text{status} \rangle) = \langle \text{expression} \rangle,$$

where

$$\langle \text{status} \rangle ::= \text{EXT} \mid \text{INT}.$$

Such rules are added to the system by the command FRULE as in the following examples:

```
FRULE L(EM,INT) = I*(G(LN,MOM) + EM)*PROPR(MOM,EM),
      L(EM,EXT) = U(LN,MOM);
```

In these examples, the only unfamiliar symbols are G which is used to represent a Dirac gamma matrix operator (in the sense that $G(L,P) \equiv \gamma.P$ for a fermion line labelled by the line identification symbol L), and PROPR, which represents a propagator denominator (i.e., $PROPR(P,M) \equiv 1/(P.P-M^2)$).

The specification of a rule for a vertex is made in a similar manner. Full details on this are given in Ref. [33] for anyone interested.

Once the appropriate rules for lines and vertices have been specified and stored internally on tables, the calculation of the relevant matrix elements from the physical structure we have mapped out from the diagram becomes a simple problem in table look-up and pattern matching. In fact, the techniques required to do this are similar to those which are used in implementing the substitution mechanisms discussed in Section 3.2. Again, a full account of these may be found in Ref. [33], and so I shall not go into detail here. Some simplification of the matrix elements can also be performed at this stage by the use of an appropriate algebraic simplification program.

3.4. Integration over internal loops

If it is possible to trace out at least one continuous closed path in a Feynman diagram, the matrix element derived from that diagram will contain at least one non-trivial four-dimensional integral over an internal four-momentum. A second type of integral that arises in calculating measurable quantities is defined over phase space or momentum space for particles produced in the process but not detected. This latter type of integral, however, is encountered during the simplification and reduction of the absolute square of the matrix element but not at this logical point in the calculation of the Feynman diagram. However, we mention it here because the problems associated with both types of integrals are essentially the same and so we can limit ourselves to a consideration of the first type of integral.

Basically, the method used to obtain such integrals is to parametrize the integrands in terms of a convenient set of variables [34] and then write

the contribution of such a term as a multidimensional integral which the computer can then manipulate either as an undetermined form or as a closed analytic expression. Until recently, the analytic calculation of such multidimensional integrals by computer had been a formidable task and only amenable to automatic computation in a few simple cases. However, it was recently recognized that most integrals occurring in calculable orders of QED may be classified in terms of a special class of functions introduced by Nielsen [35]. Several people are now looking in detail at programs to perform such integration, although the only one published to date is the Peterman SINAC program [36]. We must stress, however, that such techniques are not completely algorithmic (as were the Risch methods discussed in Section 1.5). It is necessary to know in advance all possible basic integrals which can occur in the input expression in order to perform an integration successfully. However, once one has classified these integrals, it is possible to take a general expression which arises in such calculations and reduce it automatically to a sum of terms, each of which is a known integral.

Two possible types of divergences can be met during this integration procedure. The first arises because individual terms in the answer may be themselves divergent even though the sum is not. Care must therefore be taken in combining the terms so that these divergences disappear. This process is normally referred to as regularization and creates no real problems in computer implementation. The second type of divergence requires the explicit removal of infinities in the result by a process known as renormalization.

As far as developing a completely automated system for quantum electrodynamics is concerned, renormalization presents one of the most difficult challenges. Of all the processes discussed so far, this is the one in which heuristics play the greatest role in deciding how the expressions should be manipulated. However, recent work suggests that it is possible to perform renormalization completely automatically, and programs for doing this are under development. The conventional textbook approach is biassed towards a diagrammatic analysis [34, 37], but there are other renormalization schemes [38, 39] in existence which are more algebraic. Current research in computer implementation tends towards the graphical method of renormalization, however, because there are too many inspired guesses necessary in the algebraic approach to make it easily adaptable to computer implementation. An interesting account of some work in this area may be found in Ref. [37].

3.5. Final determination of physical parameters

Now that we have our integrals expressed either in terms of exact analytic forms or undetermined expressions and all divergences removed, we are in a position to calculate a physically interesting quantity, such as a cross-section or a magnetic moment. In either case, the extraction of the relevant quantity is again a straightforward problem and presents little difficulty after one has mastered the techniques discussed in the earlier sections. This then provides us with an algebraic expression which an algebra program such as REDUCE or SCHOONSCHIP is able to evaluate successfully.

In a typical non-trivial problem, the final step requires a numerical calculation, especially if we have left integrals in an undetermined form. Because algebraic programs themselves are usually inefficient for extensive numerical computation, the answer in such cases is usually provided in the form of a FORTRAN or ALGOL expression which can be inserted in the appropriate numerical program.

By combining all the programs described in this section, we see that we do indeed have a system capable of carrying through an appropriate calculation completely automatically. We are even able to generate the diagrams themselves from a specification of the order of the process considered, if need be [33]. However, it is important to recognize that we are not really at the point yet where we can perform a non-trivial calculation in fourth-order QED by these means without human intervention. The trouble is that many complicated heuristic decisions must be made during the calculation in order to keep the size of the intermediate expressions formed under control. This is a lesson which a theoretical physicist must quickly learn if he hopes for success in hand calculations in this field. It is equally important in a computer program that such control be exercised. This point must be stressed even more in symbolic than numeric calculations because, as discussed earlier, the computation time and storage requirements tend to grow at exponential rates rather than the power rates encountered with numerical analysis.

Future work in this field must be directed towards understanding ways in which one can keep the size of these expressions under control. Our own experience suggests that this is not an insoluble problem although it may be hard to teach the computer everything that is necessary. We have found, for example, that we can make considerable reduction in the size of expressions by exploiting the many symmetries which occur in the physical problem under consideration. Basic geometrical and topological symmetries of the diagrams under study, for example, profoundly limit the possible form of the expressions which can occur in these calculations. So far, we have only been able to exploit such symmetries by hand intervention at various steps of the calculation. This is hardly automated physics, but I think that one should look on the bright side and declare this a great success for interactive computing, which of course it is. However, I still look forward to the day when non-trivial calculations in QED of the type we are discussing can be done completely automatically, but to do this we shall have to understand much better how to recognize the deep structure of expressions and exploit that structure automatically.

CONCLUSION

In the previous three sections, I have given examples of physical problems which are being solved by symbolic means on a computer. The utilization of computers for such calculations is now a matter of record. In addition to the examples discussed, one can find applications of these methods to problems in celestial mechanics, general relativity, plasma theory, theoretical chemistry, chemical engineering and structural engineering, to name a few.

I have also tried to indicate those calculational techniques which are sufficiently well developed to be of practical use right now and those which

should be available in the near future. In two or three years, for example, symbolic integration by computer will become commonplace, results will be displayed in textbook notation on cathode-ray tube displays or other output devices, and maybe we shall even input our problems by simply writing the relevant expressions on a computer tablet. Some problems which one now considers tedious and time-consuming, such as those which occur in quantum electrodynamics, will be completely automated. Moreover, because we shall have better calculational tools at our disposal, we shall also gain a much better idea of the range of problems which are amenable to exact solution. It is evident that this field, as far as physical application is concerned, has a very promising future, and I recommend it to anyone interested in innovative uses of the computer for solving physical problems.

REFERENCES

[1] PETRICK, S.R. (Ed.), Proc. Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, New York (1971).

[2] BARTON, D., BOURNE, S.R., HORTON, J.R., Comput. J. 13 (1970) 243.
BOURNE, S.R., HORTON, J.R., The CAMAL System Manual, Computer Laboratory, Cambridge, England (1971).

[3] TOBEY, R.G. et al., PL/I-FORMAC Symbolic Mathematics Interpreter, IBM Corporation Contributed Program Library 360 D 03.3004, Hawthorne, New York (1967).

[4] HEARN, A.C., "REDUCE, a user-oriented interactive system for algebraic simplification", Interactive Systems for Experimental Applied Mathematics (KLERER, M., REINFELDS, J., Eds), Academic Press, New York, London (1968) 79.
HEARN, A.C., REDUCE 2 User's Manual, Stanford Artificial Intelligence Project Memo No. AIM-133 (1970).

[5] VELTMAN, M., SCHOONSCHIP, A CDC 6600 Programme for Symbolic Evaluation of Feynman Diagrams, CERN, Geneva (1965).

[6] CAMPBELL, J.A., "Comparative survey of programming languages", paper IAEA-SMR-9/21, these Proceedings.

[7] BROWN, W.S., J. Ass. comput. Mach. 18 (1971) 478.

[8] KNUTH, D.E., The Art of Computer Programming, Vol.2, Seminumerical Algorithms, Addison-Wesley, Reading, Mass. (1969).

[9] LOOS, R., A Set of REDUCE 2 Functions for the Computation of the Greatest Common Divisor using Modular Arithmetic, University of Utah Computational Physics Group Rep. No. UCP-4 (1971).

[10] MOSES, J., Communs Ass. comput. Mach. 14 (1971) 527.

[11] FENICHEL, R., An On-line System for Algebraic Manipulation, Ph. D. thesis, Harvard, Cambridge, Mass. (1966).

[12] HEARN, A.C., Proc. 1968 Summer Institute on Symbolic Mathematical Computation, IBM Programming Laboratory Rep. FSC 69-0312 (1969) 3.

[13] HEARN, A.C., The Computer Solution of Algebraic Problems by Pattern Matching, Proc. Second Colloquium on Advanced Computing Methods in Theoretical Physics, CNRS, Marseilles (1971).

[14] FATEMAN, R., Proc. Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, New York (1971) 311.

[15] KAHRIMANIAN, H.G., Symp. Automatic Programming for Digital Computers, Office of Naval Research, Washington, D.C. (1954) 6.

[16] NOLAN, J., Analytical Differentiation on a Digital Computer, M.A. thesis, Massachusetts Inst. of Tech., Cambridge, Mass. (1953).

[17] SLAGLE, J.R., J. Ass. comput. Mach. 10 (1963) 507.

[18] MOSES, J., Symbolic Integration, Project MAC Rep. No. MAC-TR-33, Ph.D. thesis, Massachusetts Inst. of Tech., Cambridge, Mass. (1967).

[19] RISCH, R., Trans. Am. math. Soc. 139 (1969) 167.

[20] MOSES, J., Communs Ass. comput. Mach. 14 (1971) 548.

[21] BJORKEN, J.D., DRELL, S.D., Relativistic Quantum Mechanics, McGraw Hill, New York (1964).
[22] KAHANE, J., J. math. Phys. 9 (1968) 1732.
[23] CHISHOLM, J.S.R., HEARN, A.C., to be published.
[24] COLLINS, G.E., Communs Ass. comput. Mach. 9 (1966) 578.
[25] LURIĆ, D., "Introduction to LISP", paper IAEA-SMR-9/9, these Proceedings.
[26] BROWN, W.S., HYDE, J.P., TAGUE, B.A., The ALPAK System for Nonnumerical Algebra on a Digital Computer, Bell Syst. tech. J. 42 (1963) 2081; 43 (1964) 785; 43 (1964) 1547.
HALL, A.D., Jr., Communs Ass. comput. Mach. 14 (1971) 517.
[27] KUO-PETRAVIĆ, G., PETRAVIĆ, M., ROBERTS, K.V., "Translation of Symbolic ALGOL I to Symbolic ALGOL II", paper IAEA-SMR-9/22, these Proceedings.
[28] PETRICK, S.R., Proc. Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, New York (1971) 224.
[29] WEISSMAN, C., LISP 1.5 Primer, Dickenson, Belmont, Calif. (1967).
[30] KAHN, A.B., Communs Ass. comput. Mach. 6 (1963) 473.
[31] PRATT, T.W., FRIEDMAN, D.P., Communs Ass. comput. Mach. 14 (1971) 460.
[32] CRESPI-REGHIZZI, S., MORPURGO, R., Communs Ass. comput. Mach. 13 (1970) 319.
[33] CAMPBELL, J.A., HEARN, A.C., J. comput. Phys. 5 (1970) 280.
[34] BJORKEN, J.D., DRELL, S.D., Relativistic Quantum Fields, McGraw Hill, New York (1965).
[35] NIELSEN, N., Nova Acta Leopoldina, Halle 90 (1909) 125.
KÖLBIG, K.S., MIGNACO, J.A., REMIDDI, E., BIT 10 (1970) 38.
[36] PETERMAN, A., Subtracted Generalized Polylogarithms and the SINAC Program, CERN Rep. No. TH. 1451 (1972).
[37] CALMET, J., PERROTTET, M., J. comput. Phys. 7 (1971) 191.
[38] KUO, P.K., YENNIE, D.R., Ann. Phys. 51 (1969) 496.
[39] APPELQUIST, T., Ann. Phys. 54 (1969) 27.

FACULTY AND PARTICIPANTS

ORGANIZING COMMITTEE

Abdus Salam (Chairman)	International Centre for Theoretical Physics, Trieste, Italy
L. Kowarski	D.D. Division, CERN, Switzerland
K.V. Roberts	Culham Laboratory, UKAEA, Abingdon, Berks, UK
S.J. Lindenbaum	Department of Physics, Brookhaven National Laboratory, Upton, L.I., N.Y., USA and Department of Physics, City College of the City University, New York, N.Y., USA
L. Bertocchi (Local co-ordinator)	Institute of Theoretical Physics, University of Trieste, Italy

LECTURERS

J.P. Boris	Plasma Physics Division, Naval Research Laboratory, Washington, D.C. 20 390, USA	USA
J. Browne *	Department of Computer Sciences, The University of Texas, Austin, Tex. 78712, USA	USA
R. Bullough	Theoretical Physics Division, Building 8.9, Atomic Energy Research Establishment, Harwell, Didcot, Berks., UK	UK
J.A. Campbell	Department of Physics, The University of Texas, Austin, Tex. 78712, USA	Australia
V. Casarosa *	C.N.U.C.E. Via S. Maria 36, 56100 Pisa, Italy	Italy
S. Cohen	Physics Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, Ill. 60439, USA	USA

* The contribution of this lecturer is not published in these Proceedings.

A.C. Hearn	Department of Physics, The University of Utah, Salt Lake City, Utah 84112, USA	USA
R.W. Hockney	Computer Unit, University of Reading Whiteknights, Reading, Berks., UK	UK
F. James *	D.D. Division, CERN, 1211 Geneva 23, Switzerland	USA
J. Killeen	Lawrence Radiation Laboratory, University of California, Livermore, Calif. 94550, USA	USA
L. Kowarski	D.D. Division, CERN, 1211 Geneva 23, Switzerland	France
C. Kretschmar *	Digital Equipment Corporation, Maynard, Mass. 01754, USA	USA
G. Kuo-Petravić	Department of Engineering Science, Oxford University, 43 Banbury Road, Oxford, UK	Yugoslavia
R. Le Bail *	S.I. Division, CERN, 1211 Geneva 23, Switzerland	France
W.A. Lester	IBM Research Division, Monterey & Cottle Roads, San José, Calif. 95114, USA	USA
S.J. Lindenbaum	Department of Physics, Brookhaven National Laboratory, Upton, L.I., N.Y. 11973, USA	USA
E. Lohrmann	DESY, Notkestieg 1, 2 Hamburg 52, Federal Republic of Germany	Federal Republic of Germany
D. Lurié	D.D. Division, CERN, 1211 Geneva 23, Switzerland permanent: Department of Physics, Technion-Israel Institute of Technology, Haifa, Israel	Belgium
L. Montanet *	T.C. Division, CERN, 1211 Geneva 23, Switzerland	France
R.K. Nesbet	IBM Research Laboratory K07/028, Monterey and Cottle Roads, San José, Calif. 95114, USA	USA
E. Pagiola *	T.C. Division, CERN, 1211 Geneva 23, Switzerland	Italy
M. Perrottet	Centre de Physique Théorique, 31, Chemin J. Aiguier, 13 Marseille 9e, France	France

M. Petravić	Department of Engineering Science, Oxford University, 43 Banbury Road, Oxford, UK	Yugoslavia
D.E. Potter	Plasma Physics Department, Imperial College of Science and Technology, Prince Consort Road, London SW 7, UK	UK
S.P. Ratti	Istituto di Fisica, *Gruppo Alte Energie*, Via Celoria 16, 20133 Milano, Italy	Italy
J. Raynal	CEN de Saclay, Service de Physique Théorique, B.P. No.2, 91 Gif-sur-Yvette, France	France
K.V. Roberts	Culham Laboratory, UKAEA, Abingdon, Berks.,UK	UK
M.N. Rosenbluth	School of Natural Sciences, The Institute for Advanced Study, Princeton, N.J. 08540, USA	USA
Y. Sarrazin*	Digital Equipment Corporation, 81 Route de l'Aire, 1211 Geneva 23, Switzerland	France

EDITOR

A.M. Hamende	International Centre for Theoretical Physics, Trieste, Italy	Italy

LIST OF LECTURERS FOR INTRODUCTORY WEEK ON FORTRAN LANGUAGE

G. Alberi	Istituto di Fisica Teorica, c/o ICTP, Miramare, Italy	Italy
B. Carminati	Centro di Calcolo, Via A. Diaz 19, Trieste, Italy	Italy
M. Hmeljak	Centro di Calcolo, Via A. Diaz 19, Trieste, Italy	Italy

PARTICIPANTS

S.Z. Abas	Department of Mathematics, University College of North Wales, Bangor, North Wales, UK	Pakistan
S.S. Abdel-Zaher (Mrs.)	National Research Centre, Sh. El-Tahrir, Dokki-Cairo, Arab Republic of Egypt	Arab Republic of Egypt
M.C. Abramo (Miss)	Istituto di Fisica, Università di Messina, Via dei Verdi, Messina	Italy
M.H. Abul-Magd (Mrs.)	Physics Department, Cairo University, Giza, Arab Republic of Egypt	Arab Republic of Egypt
J.-C. Adam	CEN/FAR, B.P. No. 6, 92 Fontenay-aux-Roses, France	France
K. Afewu	Leipzigerstrasse 38, 6 Frankfurt/M 90, Federal Republic of Germany	Ghana
A. Aharony	Department of Physics and Astronomy, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel	Israel
S.S. Ahmad	Istituto di Fisica, Via Marzolo 8, 35100 Padova, Italy	Pakistan
A.M. Airoldi (Mrs.)	Istituto di Fisica, Gruppo Plasmi – CNR, Via Celoria 16, 20133 Milano, Italy	Italy
D.E. Ajakaiye (Miss)	Department of Physics, Ahmadu Bello University, Zaria, Nigeria	Nigeria
M.A. Alam	Physics Department, University of Tabriz, Tabriz, Iran	Pakistan
S. Alcántara Montes	Physique Mathématique, Université Libre de Bruxelles, 50, Ave. F.D. Roosevelt, B.1050 Brussels, Belgium	Mexico
G. Aleix	Institut de Calcul Numérique, 118, route de Narbonne, 31-Toulouse 04, France	Andorra

*Means that the participants also gave a seminar.

S. Ali	Institute of Physics, University of Islamabad, 77-E Satellite Town, Rawalpindi, Pakistan	Pakistan
F.K.A. Allotey	University of Science and Technology, University Post Office, Kumasi, Ghana	Ghana
D. Andrews	Danish Space Research Institute, Lundtoftevej 7, 2800 Lyngby, Denmark	UK
C. Aragone	Instituto de Física, J. Herrera y Reissig 565, Montevideo, Uruguay	Uruguay
F. Arbab	Departamento de Física, Universidad del Valle, Cali, Colombia	USA
L.F. Arnould	Institut de Physique, Université de Liège, Sart Tilman (Liège 1), B-4000, Belgium	Belgium
C. Arzola-Sánchez	Departamento Académico de Física, Universidad Nacional Mayor de San Marcos, La Victoria, Lima, Peru	Peru
D. Banks	Physics Department, University of Stirling, Stirling, Scotland, UK	UK
M.B. Bari	Department of Physics, Faculty of Science, University of Tabriz, Tabriz, Iran	Pakistan
M. Barmawi	Institut Teknologi Bandung, Ganeca 10, Bandung, Indonesia	Indonesia
S. Barmo	Institut für Theoretische Kernphysik, Kaiserstrasse 12, 75 Karlsruhe, Federal Republic of Germany	Syria
S.C. Bazaj	Department of Computer and Communications Sciences, University of Michigan, Ann Arbor, Mich. 48104, USA	India
R. Bednarz	Institute of Nuclear Research, Department XXI, Świerk, Poland	Poland
F. Benassi	Laboratorio Fisica Cosmica, Gruppo Spazio, Via Celoria 16, 20133 Milano, Italy	Italy

E. Bittoni	CNEN - Centro di Calcolo, Via Mazzini 2, 40126 Bologna, Italy	Italy
L. Bodini (Mrs.)	INFN, Via Celoria 16, 20133 Milano, Italy	Italy
E. Boeker	Natuurkundig Laboratorium der Vrije Universiteit, De Boelelaan 1081, Amsterdam, Buitenveldert, Netherlands	Netherlands
G.C. Bonsignori	Istituto di Fisica "A. Righi", Via Irnerio 46, 40126 Bologna, Italy	Italy
F.G. Bottiglioni	Association EUR-CEA, B.P. No.6, 92 Fontenay-aux-Roses, France	Italy
M. Bouten	CEN/SCK, Boeretang 200, B-2400 Mol, Belgium	Belgium
M.-C. Bouten-Denys (Mrs.)	CEN/SCK, Boeretang 200, B-2400 Mol, Belgium	Belgium
C. Bovet	S.I. Division, CERN, 1211 Geneva 23, Switzerland	Switzerland
F.M.O. Brander	Institute of Theoretical Physics - FACK, S-402 20 Göteborg 5, Sweden	Sweden
D.J. Buchanan	Department of Theoretical Physics, University of St. Andrews, St. Andrews, Fife, Scotland, UK	UK
I. Čadež	Institute of Physics, Studentski trg 16/IV, P.O. Box 57, 11001 Belgrade, Yugoslavia	Yugoslavia
G.M. Carabelli (Mrs.)	IMAF, Laprida 854, Córdoba, Argentina	Italy
C.A. Caroni (Mrs.)	C.I.T.E.F.A., Zufrategui y Varela, Prov. de Buenos Aires, Argentina	Argentina
E. Castelli	Istituto di Fisica, Via A. Valerio 2, 34127 Trieste, Italy	Italy

F. Castelli (Mrs.)	Laboratorio Astronomico, Via Tiepolo 11, 34100 Trieste, Italy	Italy
F. Cavallini	Istituto di Fisica, Via A. Valerio 2, 34127 Trieste, Italy	Italy
M. Ceschia	Istituto di Fisica, Via A. Valerio 2, 34100 Trieste, Italy	Italy
S. Chamswasdi (Mrs.)	Applied Scientific Research Corp. of Thailand, Bangkhen, Bangkok, Thailand	Thailand
L.-C. Chen	Instituut voor Theoretische Fysica, Katholieke Universiteit, Nijmegen, Netherlands	China
C-x. Chuân	Département de Physique Atomique, Faculté des Sciences, Université de Saigon, Sô 227, duòng Công-Hòa, Saigon, Viet-Nam	Viet Nam
E. Coffou	Institute "R. Bosković", P.O. Box 1016, Zagreb, Yugoslavia	Yugoslavia
A.P. Colleraine	Department of Physics, Florida State University, Tallahassee, Fla. 32306, USA	UK
O.D. Corbella	IMAF, Laprida 854, Córdoba, Argentina	Argentina
S. Costa	Istituto di Elettrotecnica e di Elettronica, Centro Gas Ionizzati - CNR, Via Gradenigo 6/a, 35100 Padova, Italy	Italy
G.D. Craig	Institut für Theoretische Kernphysik, Schlossgartenstrasse 9, Technische Hochschule Darmstadt, 61 Darmstadt, Federal Republic of Germany	USA
J.G. Dardis	Office of Naval Research, Arlington, Va. 22217, USA	USA
E.F. Da Silveira	CEN de Saclay, D. Ph. N. B. E., B.P. No.2, 91 Gif-sur-Yvette, France	Brazil
L. De Anda (Mrs.)	Computer Centre, Essex University, Colchester, UK	Argentina

O. De Barbieri	Service IGN, Centre d'Etudes Nucléaires, Cedex 85, 38 Grenoble Gare, France	Italy
N. Deciu	Institute of Atomic Physics, P.O. Box 35, Bucharest, Romania	Romania
M.J. De Jong	Environmental Laboratory, Box 15, Code 5334, Point Mugu, Calif. 93041, USA	USA
H.R. Depuydt (Miss)	CEN/SCK Boeretang 200, B-2400 Mol, Belgium	Belgium
S. De Schrijver*	CDC, 46, Avenue des Arts, 1040 Brussels, Belgium	Belgium
E. Detyna	Department of Physics, Royal Holloway College, Englefield Green, Surrey, UK	Poland
E.A. Doğukan (Miss)	Physics Department, Middle East Technical University, Ankara, Turkey	Turkey
A. Drukier	Danish Space Research Institute, Lundtoftevej 7, 2800 Lyngby, Denmark	Poland
C.J. Elliott	Department of Physics, University of Alberta, Edmonton 7 (Alberta), Canada	Canada
M.K. El-Mously	Physics Department, Faculty of Science, Ain Shams University, Abbassia, Cairo, Arab Republic of Egypt	Arab Republic of Egypt
F. Fabbri (Miss)	CNEN, Via Mazzini 2, 40138 Bologna, Italy	Italy
D. Falk	Institut für Theoretische Physik, Freie Universität, Arnimallee 3, 1 Berlin 33, Federal Republic of Germany	Federal Republic of Germany
P. Fasoli-Stella (Mrs.)	ISPRA, CCR-EURATOM, 21021 Varese, Italy	Italy
M. Foshina (Miss)	Instituto de Energia Atômica, Caixa Postal 11049, São Paulo, Brazil	Brazil
B.M. Furmańska (Miss)	Institute of Nuclear Research, Lab. of High Energy Physics, Al. Mickiewicza 30, Krakow, Poland	Poland

J.C. Gallardo	c/o ICTP, Miramare 586, Trieste, Italy	Argentina
M. Gbordzoe	Weimarstrasse 39, 7 Stuttgart 1, Federal Republic of Germany	Ghana
C. Georgalas	N.R.C. "Democritos", Aghia Paraskevi, Attikis, Athens, Greece	Greece
L. Georgalas (Mrs.)	N.R.C. "Democritos", Aghia Paraskevi, Attikis, Athens, Greece	Greece
A.L. Gibson	Department of Natural Philosophy, University of Glasgow, Glasgow W2, Scotland, UK	UK
P. Giupponi	CNEN, Laboratorio Gas Ionizzati, Via E. Fermi, 00044 Frascati (Roma), Italy	Italy
V.E. Godwin	Fourah Bay College, University of Sierra Leone, Freetown, Sierra Leone	Sierra Leone
S.P. Goel	Physics Department, Kurukshetra University, Kurukshetra (Haryana), India	India
L.A. Gomez de Ibarra	Istituto di Fisica, Piazza Torricelli 2, 56100 Pisa, Italy	Mexico
G.P. Gopal	Rutherford Laboratory, Chilton, Didcot, Abingdon, Berks., UK	Kenya
J. Gratton	Departamento de Física, Facultad de Ciencias Exactas y Naturales, Perú 222, Buenos Aires, Argentina	Italy
R.C. Grimm	Culham Laboratory, UKAEA, Abingdon, Berks., UK	Australia
P.S. Grover	Department of Physics and Astrophysics, University of Delhi, Delhi 7, India	India
M. Guillemot	S.P.P., B.P. No.6, 92 Fontenay-aux-Roses, France	France
T.D. Hadnagy	Department of Physics, University of Utah, Salt Lake City, Utah 84112, USA	USA
L. Halpern	Physique Mathématique, Université Libre de Bruxelles, 50, Ave. F.D. Roosevelt, B-1050 Brussels, Belgium	Austria

M.Y.M. Hassan	Physics Department, Cairo University, Giza, Arabe Republic of Egypt	Arab Republic of Egypt
C. Hegedűs	Central Research Institute for Physics, P.O. Box 49, Budapest 114, Hungary	Hungary
P. Heinrich	Kärntner Ring 11, IAEA, A-1010 Vienna, Austria	Austria
J. Hendeković	Institute "R. Bosković", P.O. Box 1016, Zagreb, Yugoslavia	Yugoslavia
A. Houari	Département de Physique, Université Mohammed V, Avenue Moulay Chérif, Rabat, Morocco	Morocco
M.H. Hughes	Culham Laboratory, UKAEA, Abingdon, Berks., UK	UK
F. Insinger	IBM Nederlands N.V., Scientific Relations, Postbus 9999, Amsterdam, Netherlands	Netherlands
C. Jacoboni	Istituto di Fisica, Laboratorio di Elettronica, Via Vivaldi 70, 41100 Modena, Italy	Italy
Z. Jaeger	Israel Atomic Energy Commission, Soreq Nuclear Research Centre, Yavne, Israel	Israel
L.E. Janicke	Institut für Plasmaphysik, Postfach 365, 517 Jülich, Federal Republic of Germany	Federal Republic of Germany
M. Jireš	Institute of Physics, Na Slovance 2, Prague 8, Czechoslovakia	Czechoslovakia
J.-L. Joly	Institut de Calcul Numérique, 118, route de Narbonne, 31-Toulouse 04, France	France
T. Kamimura	Institute of Plasma Physics, Nagoya University, Nagoya, Japan	Japan
B.R. Karlsson	Institute or Theoretical Physics - FACK, S-402 20 Göteborg 5, Sweden	Sweden
A.B. Khalil	Department of Mathematics, Faculty of Science, Cairo University, Giza, Arab Republic of Egypt	Arab Republic of Egypt

T. Khan (Mrs.)		c/o ICTP, Miramare 586, Trieste, Italy	Pakistan
J.D. Kilkenny		Plasma Physics Group, Imperial College of Science and Technology, Prince Consort Road, London SW 7, UK	UK
H. Kobeissé		Department of Physics, Lebanese University, P.O. Box 7142, Beirut, Lebanon	Lebanon
D. Kodric	home:	Via Ghirlandaio 22/2, 34100 Trieste, Italy	Italy
A.A. Kobine		D.D. Division, CERN, 1211 Geneva 23, Switzerland	UK
K.S. Kölbig		D.D. Division, CERN, 1211 Geneva 23, Switzerland	Federal Republic of Germany
S.R. Komy		Faculty of Science, Assyout University, Assyout, Arab Republic of Egypt	Arab Republic of Egypt
A. Lagonegro (Mrs.)		Centro di Calcolo, Via A. Diaz 19, 34100 Trieste, Italy	Italy
M. Lagonegro	home:	Via B. Angelico 2/1, 34100 Trieste, Italy	Italy
A. Lamont		Department of Natural Philosophy, University of Glasgow, Glasgow W 2, Scotland, UK	UK
D.W. Lamotte		S.I. Division, CERN, 1211 Geneva 23, Switzerland	Belgium
A. Laratta		Istituto di Elaborazione della Informazione, Via S. Maria 46, 56100 Pisa, Italy	Italy
A.C. Lejeune		Physique Nucléaire Théorique, Université de Liège, Sart Tilman (Liège 1), Belgium	Belgium
A.B. Lidiard *		Theoretical Physics Division, Building 8.9, Atomic Energy Research Establishment, Harwell, Didcot, Berks., UK	UK
I.A. Lilja		Department of Physics, University of Jyväskylä, Nisulankatu 78, Jyväskylä, Finland	Finland

T.J.L. Lindén	ESRO-ESOC, Robert-Bosch-Strasse 5, 6100 Darmstadt, Federal Republic of Germany	Sweden
P. Lipnik	Centre de Physique Nucléaire, Avenue Cardinal Mercier, Parc d'Arenberg, 3030 Héverlé-Louvain, Belgium	Belgium
M.D. Lloyd	Unilever Computer Services, Ltd., Station House, Harrow Rd., Wembley, Middlesex, UK	UK
S.C. Loh	Computing Centre, The Chinese University of Hong Kong, 545 Nathan Road, Kowloon, Hong-Kong	UK
M.D. de Lonngi (Mrs.)	Comision Nacional de Energia Nuclear, Av. Insurgentes Sur 1079, Apdo. Postal 27 190, Mexico 18, D.F., Mexico	Mexico
P.A. Lonngi	Comisión Nacional de Energía Nuclear, Av. Insurgentes sur 1079, Apdo. Postal 27-190, Mexico 18, D.F.	Mexico
V. Lopac (Mrs.)	Institute "R. Bosković", P.O. Box 1016, Zagreb, Yugoslavia	Yugoslavia
P. Lundborg	Institute of Physics, University of Stockholm, Vanadisvägen 9, S-113 46 Stockholm, Sweden	Sweden
C.H. Lutterodt	Department of Mathematical Physics, University of Birmingham, Edgbaston, Birmingham 15, UK	Ghana
P.A. Machin (Miss)	Atlas Computer Laboratory, Chilton, Didcot, Berks., UK	UK
D. Mack	Institut für Theoretische Physik, Universität Tübingen, Köstlinstrasse 6, D-7400 Tübingen, Federal Republic of Germay	Federal Republic of Germany
F.H. Makary	National Research Centre, Sh. El-Tahrir, Dokki-Cairo, Arab Republic of Egypt	Arab Republic of Egypt
L. Mango (Mrs.)	CSN Casaccia - CNEN, Via Anguillarese km 1.6, S. Maria di Galeria, 00060 Roma, Italy	Italy

I. Manno	Central Research Institute for Physics, P.O. Box 49, Budapest 114, Hungary	Hungary
M. Mazzanti	INFN, Via Celoria 16, 20133 Milano, Italy	Italy
C.R. McKee	New Mexico Institute of Mining and Technology, Socorro, N.M. 87801, USA	USA
G.A. Medrano	Instituto de Estudios Nucleares, Junta de Energía Nuclear, Madrid 3, Spain	Spain
E. Menapace	CNEN, Via Mazzini 2, 40138 Bologna, Italy	Italy
P. Mengoli	CNEN, Via Mazzini 2, 40138 Bologna, Italy	Italy
N. Metzler	Department of Physics, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel	Israel
F. Michel	Theoretical Physics Department, Centre Universitaire de l'Etat, 7000 Mons, Belgium	Belgium
M. Milgrom	Physics Department, The Weizmann Institute of Science, Rehovoth, Israel	Israel
P. Mondino	Istituto di Elettrotecnica e di Elettronica, Centro Gas Ionizzati - CNR, Via Gradenigo 6/a, 35100 Padova, Italy	Italy
J.R. Morales	Universidad de Chile, Facultad de Ciencias, Casilla 653, Santiago, Chile	Chile
K.J.M. Moriarty	Theoretical Physics Department, Imperial College of Science and Technology, Prince Consort Road, London SW 7, UK	Canada
F. Moser	Institut für Plasmaforschung, Universität Stuttgart, Pfaffenwaldring 31, 7 Stuttgart 80, Federal Republic of Germany	Federal Republic of Germany
H.A. Munera	Instituto de Asuntos Nucleares, Apartado 8595, Bogotá, D.E., Colombia	Colombia

A. Musgrove	A.A.E.C., Lucas Heights, 2232 New South Wales, Australia	Australia
P. Mussio	Laboratorio Fisica Cosmica - CNR, Via Celoria 16, 20133 Milano, Italy	Italy
J. Nadrchal	Institute of Solid State Physics, Cukrovarnická 10, Prague 6, Czechoslovakia	Czechoslovakia
M.Y. Navet	Ecole des Mines, Parc de Saurupt, 54 Nancy, France	France
I. Nedelkov	Institute of Physics, 72 Boul. Lenin, Sofia 13, Bulgaria	Bulgaria
O.A. Novaro	Instituto de Física UNAM, Z.P. 20, Mexico City, Mexico	Mexico
C. Nussbaum	Institut de Physique, Université de Neuchâtel, 2000 Neuchâtel	Switzerland
C.E. Okeke	Physics Department, University of Nigeria, Nsukka, East Central State, Nigeria	Nigeria
S.E. Okoye	Department of Physics, University of Nigeria, Nsukka, East Central State, Nigeria	Nigeria
P.P. Ong	Physics Department, University of Singapore, Bukit Timah Road, Singapore	Singapore
A.H. Osman	Physics Department, University of Khartoum, Khartoum, Sudan	Sudan
R. Oteng	Department of Physics, Faculty of Science, University College, Cape Coast, Ghana	Ghana
P. Palazzi *	D.D. Division, CERN, 1211 Geneva 23, Switzerland	Switzerland
L. Panattoni	IBM, Lungarno Mediceo 40, 56100 Pisa, Italy	Italy
R.A. Pasmanter	Department of Electronics, The Weizmann Institute of Science, Rehovoth, Israel	Argentina

A. Patkós	Department of Atomic Physics, Eötvös University, Puskin u. 5-7, Budapest VIII, Hungary	Hungary
E. Peleg	Department of Nuclear Physics, The Weizmann Institute of Science, Rehovoth, Israel	Israel
N. Peleg (Miss)	State of Israel, Ministry of Defense, P.O. Box 7063, Tel-Aviv, Israel	Israel
J.O. Peralta	Facultad de Ciencias, Universidad de Los Andes, Merida, Venezuela	Argentina
M.-T. Peralta (Mrs.)	Facultad de Ciencias, Universidad de Los Andes, Merida, Venezuela	Argentina
F. Petrossi	Facoltà di Scienze, Università degli Studi, 34100 Trieste, Italy	Italy
J.S. Phillips	Department of Physics, University of Toronto, Toronto 5 (Ontario), Canada	UK
M. Popescu	Institutul de Fizica, Bd. Pacii 222, Bucharest 16, Romania	Romania
R. Pozzoli	Gruppo Plasmi, Via Celoria 16, 20133 Milano, Italy	Italy
M. Profant	Zentralinstitut für angewandte Mathematik, Postfach 365, 517 Jülich 1, Federal Republic of Germany	Czechoslovakia
L. Questa (Miss)	Istituto di Fisica, Via Celoria 16, 20133 Milano, Italy	Italy
V. Radojević	Institute "B. Kidrić", Zgrada 53, P.O.B. 522, Belgrade, Yugoslavia	Yugoslavia
J. Rant	Institute "J. Stefan", Jamova 39, Ljubljana, Yugoslavia	Yugoslavia
K. Rashid	Department of Physics, University of the Punjab, New Campus, Lahore, Pakistan	Pakistan

C.R. Rathbone	Mathematics Department, University of Malaya, Kuala Lumpur, Malaysia	UK
L. Renni	Istituto di Chimica, Università di Trieste, 34100 Trieste, Italy	Italy
J.A. Reynolds	Culham Laboratory, UKAEA, Abingdon, Berks, UK	UK
M. Ribarić	Institute "J. Stefan", Jamova 39, Ljubljana, Yugoslavia	Yugoslavia
A. Rigal	Institut de Calcul Numérique, 118, route de Narbonne, 31-Toulouse 04, France	France
A.M. Romaya	Department of Nuclear Physics, University of Oxford, Keble Road, Oxford OX1 3RH, UK	Iraq
L.P. Rosa	COPPE-UFRJ, P.O. Box 1191, ZC-00 Rio de Janeiro, Brazil	Brazil
Saifuddin	Institute of Physics, University of Islamabad, 77-E Satellite Town, P.O. Box 226, Rawalpindi, Pakistan	Pakistan
I.A. Sakr	National Research Centre, Sh. El-Tahrir, Dokki-Cairo, Arab Republic of Egypt	Arab Republic of Egypt
S. Sala	INFN, Via Celoria 16, 20133 Milano, Italy	Italy
M.A. Sallam	Ministry of Education, Sanàa, Yemen	Yemen A.R.
V.K. Samaranayake *	Department of Mathematics, University of Sri Lanka, Colombo 3, Sri Lanka	Sri Lanka
A. Sandoval	Max-Planck Institut für Kernphysik, Postfach 1248, 69 Heidelberg 1, Federal Republic of Germany	Mexico
M.G.R. Scherer	Institut d'Aéronomie Spatiale, Avenue Circulaire 3, B-1180 Brussels, Belgium	Belgium
J.N. Schmit	Institut de Physique, Université de Liège, Sart Tilman (Liège 1), B-4000, Belgium	Luxembourg

P. Schmuck	Zentrum für Datenverarbeitung, Universität Tübingen, Köllestrasse 1, 7400 Tübingen, Federal Republic of Germany	Austria
M. Sessa	Istituto di Fisica, Via A. Valerio 2, 34127 Trieste	Italy
V.P. Seth	Physics Department, Allahabad University, Allahabad, India	India
W.J. Sharp	Department of Natural Philosophy, University of Glasgow, Glasgow W 2, Scotland, UK	UK
M.C.A. Silva (Mrs.)	Faculdade de Ciencias, R. Escola Politecnica, Lisbon 1, Portugal	Portugal
A. Skorupski	Institute of Nuclear Research, ul. Hoża 69, Warsaw, Poland	Poland
M.A. Smiljanić	Institute of Physics, 11000 Belgrade, Yugoslavia	Yugoslavia
J.Z. Šoln	Department of Physics, University of Illinois, P.O. Box 4348, Chicago, Ill. 60680, USA	Yugoslavia
M. Soop	Casella Postale 70, ESRIN, 00044 Frascati (Roma), Italy	Sweden
A. Sørenssen*	M.P.S. Division, CERN, 1211 Geneva 23, Switzerland	Norway
Y. Soshkovich	Department of Physics and Astronomy, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel	Israel
T.S. Stanev	Institute of Physics, Cosmic Rays Laboratory, 72 Boul. Lenin, Sofia 13, Bulgaria	Bulgaria
A.M. Sternlieb	Department of Physics and Astronomy, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel	Israel
H.J. Strubbe	Institute for Theoretical Physics, University of Utrecht, Maliesingel 23, Utrecht, Netherlands	Belgium
P. Sverzellati	INFN, Via Celoria 16, 20133 Milano, Italy	Italy

S. Svetina	Institute "J. Stefan", Jamova 39, Ljubljana, Yugoslavia	Yugoslavia
B.D.C.B. Syme	Kelvin Laboratory, N.E.L., East Kilbridge, Glasgow, Scotland, UK	UK
A. Taroni	CNEN - Centro di Calcolo, Via Mazzini 2, 40138 Bologna, Italy	Italy
C. Tebaldi	Istituto di Fisica, Via Irnerio 46, 40126 Bologna, Italy	Italy
Z.D. Thomé Jr.	COPPE-UFRJ, Caixa Postal 1191, ZC-00 Rio de Janeiro, Brazil	Brazil
M. Tomás	División de Física, Junta de Energía Nuclear, Madrid, Spain	Spain
M. Tomšič	Institute "J. Stefan", Jamova 39, Ljubljana, Yugoslavia	Yugoslavia
R.W. Townsend	Simulation Laboratory, Code 5340, Point Mugu, Calif. 93041, USA	USA
M. Troppmann	M.P.I. für Plasmaphysik, D-8046 Garching, Federal Republic of Germany	Federal Republic of Germany
J.K. Trulsen	Department of Physics, University of Tromso, P.O. Box 387, Tromso 9001, Norway	Norway
M.E. Trunk	Institut für Plasmaforschung, Universität Stuttgart, Pfaffenwaldring 31, 7 Stuttgart 80, Federal Republic of Germany	Federal Republic of Germany
G.H. Tuttle	Culham Laboratory, UKAEA, Abingdon, Berks., UK	UK
C. Uberoi (Mrs.)	Department of Applied Mathematics, Indian Institute of Science, Bangalore 12, India	India
E. Valente	INFN, Piazzale delle Scienze 5, 00185 Roma, Italy	Italy
R.A. Valls	Centro de Investigación Digital (CID), Facultad de Tecnología, Universidad de la Habana, Havana, Cuba	Cuba

A. Van der Meulen	F.O.M. Institute of Atomic and Molecular Physics, Kruislaan 407, Amsterdam-0, Netherlands	Netherlands
A. Van Deursen	Department of Mathematics, Delft University of Technology, Julianalaan 132, Delft, Netherlands	Netherlands
R. Van Maercke	Physics Department, University of Leuven, Celestijnenlaan 200D, Héverlé, Belgium	Belgium
G. Verri	Laboratori Nazionali di Frascati, Casella Postale 70, 00044 Frascati (Roma), Italy	Italy
T. Vertse	Institute of Nuclear Research, Bem-tér 18c, Debrecen 1, Pf. 51, Hungary	Hungary
G. Viola	Olivetti, Ivrea, Direzione Marketing, Servizio Software, Via Jervis 77, Ivrea 10015, Italy	Italy
F. Vitalis	F.O.M. Institute of Atomic and Molecular Physics, Kruislaan 407, Amsterdam-0, Netherlands	Netherlands
K.U. Von Hagenow	Institut für Plasmaphysik, D-8046 Garching, Federal Republic of Germany	Federal Republic of Germany
S. Vuković	Institute of Physics, P.O. Box 57, 11001 Belgrade, Yugoslavia	Yugoslavia
H.G.K. Walther	Kernforschungsanlage, 517 Jülich, Federal Republic of Germany	Federal Republic of Germany
M.L. Watkins	Plasma Physics Group, Imperial College of Science and Technology, Prince Consort Road, London SW 7, UK	UK
H. Weicksel	Institut für Physik, Jacob-Welder-Weg 11, Postfach 3980, 6500 Mainz, Federal Republic of Germany	Federal Republic of Germany
M. Weissmann (Miss)	Facultad de Ciencias, Universidad de Chile, Casilla 653, Santiago, Chile	Argentina
P.H. Whiting III	Department of Physics, Syracuse University, Syracuse, N.Y. 13210, USA	USA

V.A. Williams	Department of Physics and Electronics, University of Ife, Ile-Ife, Nigeria	Nigeria
T.-C. Wong	Physics Department, University of Western Ontario, London (Ontario), Canada	UK
P.E.S. Wormer	Department of Theoretical Chemistry, University of Nijmegen, Toernooiveld Nijmegen, Netherlands	Netherlands
D. Yeboah-Amankwah	Department of Physics, University of Ghana, Legon, Accra, Ghana	Ghana
P.E. Zadunaisky	Departamento de Física, Universidad de la Plata, Calle 115 y 49, La Plata, Argentina	Argentina
J.D.L. Zeiler	Institut für Theoretische Physik, Freie Universität, Arnimallee 3, 1 Berlin 33, Federal Republic of Germany	Federal Republic of Germany
A.O. Zohni	c/o ICTP, Miramare 586, Trieste, Italy	Arab Republic of Egypt

and physicists present at the Centre.

HOW TO ORDER IAEA PUBLICATIONS

Exclusive sales agents for IAEA publications, to whom all orders and inquiries should be addressed, have been appointed in the following countries:

UNITED KINGDOM	Her Majesty's Stationery Office, P.O. Box 569, London S.E.1
UNITED STATES OF AMERICA	UNIPUB, Inc., P.O. Box 433, New York, N.Y. 10016

In the following countries IAEA publications may be purchased from the sales agents or booksellers listed or through your major local booksellers. Payment can be made in local currency or with UNESCO coupons.

ARGENTINA	Comisión Nacional de Energía Atómica, Avenida del Libertador 8250, Buenos Aires
AUSTRALIA	Hunter Publications, 58 A Gipps Street, Collingwood, Victoria 3066
BELGIUM	Office International de Librairie, 30, avenue Marnix, Brussels 5
CANADA	Information Canada, 171 Slater Street, Ottawa, Ont. K1A OS 9
C.S.S.R.	S.N.T.L., Spálená 51, Prague 1 Alfa, Publishers, Hurbanovo námestie 6, Bratislava
FRANCE	Office International de Documentation et Librairie, 48, rue Gay-Lussac, F-75 Paris 5e
HUNGARY	Kultura, Hungarian Trading Company for Books and Newspapers, P.O.Box 149, Budapest 62
INDIA	Oxford Book and Stationery Comp., 17, Park Street, Calcutta 16
ISRAEL	Heiliger and Co., 3, Nathan Strauss Str., Jerusalem
ITALY	Agenzia Editoriale Commissionaria, A.E.I.O.U., Via Meravigli 16, I-20123 Milan
JAPAN	Maruzen Company, Ltd., P.O.Box 5050, 100-31 Tokyo International
NETHERLANDS	Martinus Nijhoff N.V., Lange Voorhout 9—11, P.O.Box 269, The Hague
PAKISTAN	Mirza Book Agency, 65, The Mall, P.O.Box 729, Lahore-3
POLAND	Ars Polona, Centrala Handlu Zagranicznego, Krakowskie Przedmiescie 7, Warsaw
ROMANIA	Cartimex, 3-5 13 Decembrie Street, P.O.Box 134-135, Bucarest
SOUTH AFRICA	Van Schaik's Bookstore, P.O.Box 724, Pretoria Universitas Books (Pty) Ltd., P.O.Box 1557, Pretoria
SWEDEN	C.E.Fritzes Kungl. Hovbokhandel, Fredsgatan 2, Stockholm 16
U.S.S.R.	Mezhdunarodnaya Kniga, Smolenskaya-Sennaya 32-34, Moscow G-200
YUGOSLAVIA	Jugoslovenska Knjiga, Terazije 27, Belgrade

Orders from countries where sales agents have not yet been appointed and requests for information should be addressed directly to:

Publishing Section,
International Atomic Energy Agency,
Kärntner Ring 11, P.O.Box 590, A-1011 Vienna, Austria

INTERNATIONAL
ATOMIC ENERGY AGENCY
VIENNA, 1972

PRICE: US $21.00
Austrian Schillings 488,-
(£9.00; F.Fr.105,-; DM 67,-)

SUBJECT GROUP: III
Physics/Theoretical Physics