A review of algebraic manipulative programs and their application

D. Barton and J. P. Fitch

University of Cambridge Computer Laboratory, Corn Exchange Street, Cambridge CB2 3QG

This paper describes the applications area of computer programs that carry out formal algebraic manipulation. The first part of the paper is tutorial and several typical problems are introduced which can be solved using algebraic manipulative systems. Sample programs for the solution of these problems using several algebra systems are then presented. Next, two more difficult examples are used to introduce the reader to the true capabilities of an algebra program and these are proposed as a means of comparison between rival algebra systems. A brief review of the technical problems of algebraic manipulation is given in the final section.

(Received May 1972)

1. Introduction

The computer's ability to perform large quantities of elementary arithmetic has made such machines indispensable in many areas of science. However, it is only during the past decade that their ability to perform large amounts of elementary algebra has been exploited, and it is reasonable to suppose that as their capability in this area is increased, their use for solving practical problems in science and engineering will also grow. There is little doubt that the facilities which exist at present for the mechanisation of routine algebra on computers are extremely elementary and for the most part rather crude, but nevertheless they have been found useful in a number of different branches of science and in some areas they have made possible significant advances which, in their absence, would have been much harder. The users' contact with a present day algebra system is generally via a programming language which is similar in its syntax structure to languages intended for numerical calculation. However, algebraic programming languages differ from conventional languages in the essential respect that the 'values' of the variables (or vectors or matrix elements) declared in the program are algebraic expressions represented formally within the computer memory. Thus, when the program contains an instruction to form the sum of two variables, the symbolic expressions that are the values of those variables are added together according to the usual rules of algebra. The result of the sum is itself an expression that, if printed, will appear on the page as text containing letters, digits, parentheses, and so forth.

In order to make use of an algebra system it is first necessary to break down the problem that is to be solved into a sequence of simple steps involving only such elementary operations as addition, multiplication or differentiation, in exactly the same way as a numerical problem must be broken down. Thus, if the problem is to solve a given differential equation and obtain the formal solution subject to certain boundary conditions, an algorithm must be derived which reduces that problem to a sequence of elementary steps. This reduction may not be easy, but it is a labour that properly belongs to the users of algebra systems and not to the system designer. It can be expected that, as algebra systems become more widely available, library routines will be constructed for the solution of various important and central problems such as the integration of differential equations; recent work demonstrates that this has already begun to occur. The existence of a large library of algebraic routines will of course greatly simplify the task of obtaining algebraic solutions to problems in exactly the same way as numerical library subroutines simplify the writing of numerical programs.

For a number of reasons it has taken a long time for the development of computer programs for the manipulation of algebraic expressions to reach the present state where they can be of some use to scientists. Perhaps the most obvious difficulty that arises in the construction of an algebra system is that the hardware of a computer is not designed for the manipulation of algebraic expressions, and consequently the simple facilities of addition and multiplication between expressions must be provided by program. No algebraic facilities can exist on the computer at all until the program has been written, and the writing of the program, while it is in principle straightforward, has in practice required the use of advanced and in many cases recently developed programming techniques. It has frequently been the case that these programs could only be constructed by small groups of dedicated programmers with access to large computing facilities providing interactive terminal systems. It appears that, at present, general purpose algebraic manipulative systems can only be made available on large computers. The construction of programs for the elementary combination of algebraic expressions under, say, addition and multiplication is, however, only the first step towards the provision of a useful system for their manipulation. Once such programs exist a new set of problems appear, and these must be overcome before the programs can be of any real value. Perhaps the most important of these new problems is the one that is known as 'blow-up'. It will be clear that when algebraic expressions are written on paper they are not all of the same length. So, when they are represented in the memory of a computer they do not all occupy the same amount of space. It has proved to be the case in practice that even simple problems involving the manipulation of functions, problems starting from and resulting in quite short expressions, frequently lead to intermediate expressions of inordinate length in the course of computation. If these intermediate expressions become too long they cannot be retained in the computer's memory, and the calculation cannot be completed. The problem of blow-up is the problem of controlling this intermediate expression swell. It has received a great deal of attention in the past few years (Collins, 1965; Barton *et al.*, 1970b; Bourne and Horton, 1971b) and to the extent that solutions to it have been found the second step towards a useful symbol manipulation system may be said to have been taken.

Suppose now that we have carried out some algebraic calculations on a computer—we have evaluated an integral or solved a differential equation. Obviously we need to be able to read the answer, and this simple remark leads the system engineer to the interesting and difficult problem: how should the results of algebraic calculations be presented to the user given the peripheral devices currently available on computers? When an algebraic expression is written on the page by hand, or printed in a book, it is commonly presented in a two-dimensional format using a very large character set extending over

several alphabets, founts and types. Furthermore, the meaning of the expression may well be context dependent, as is indicated by the simple example x^i, which means a component of the vector $\mathbf{x}$ in some areas of mathematics, and the quantity x raised to the power i in others. Significant progress has been made in the area of producing machine output in a human readable form using a number of different techniques (Martin, 1971) but the problem has not yet been satisfactorily overcome. The character sets available on present day line printers and typewriter terminals are so restricted that a really satisfactory solution is unlikely to be obtained using that equipment. If displays are available it is possible to construct complex mathematical symbols as these are needed from the primitive symbols of lines and dots. However, when this is done, the major limitation is the size of the display screen that allows only a comparatively small expression to be seen at any one time. It is possible that some progress could be made using high resolution displays and photographic techniques or, possibly, by automated control of a type setting machine, but that would inevitably be expensive and leads one to question whether long expressions should be printed or displayed at all. It may well be more convenient in some cases to convert those long expressions directly into the form of a FORTRAN program for subsequent numerical calculation and graphical display, but at present such facilities are hardly available in algebra systems. Notable exceptions here are FORMAC (Xenakis, 1971), Hearn's REDUCE (Hearn, 1968) and Mesztenyi's FORMAL (Mesztenyi, 1971). The latter two systems provide FORTRAN input, but this FORTRAN program must then be run independently of the algebraic program. There has been a tendency for researchers in the field to consider that the production of algebraic expressions is the aim of their work and little effort has been expended on the provision of facilities that enable these results to be used easily in subsequent numerical processing. It seems to the present authors that work in this area would be profitable.

Simplification, the transformation of a given algebraic expression into a previously defined 'simple' form under the normal rules of cancellation and using the various relationships between functions, is a central problem in the construction of algebra systems. Both the user and the system engineer have an interest in the form of an algebraic expression; the user will expect that any output resulting from a computation will appear at his terminal in a simplified form while the system engineer will be concerned with the simple and economic representation of algebraic expressions in the machine. Unfortunately it is not clear what constitutes a simplified form to an individual user and it may well have a subjective definition such as 'that representation most easily comprehensible in the context of the problem'. The system engineer's requirement will at least be more precisely stated; for example, it may be 'that representation which requires the least quantity of computer memory for its storage'. Let us consider the case of a programming system for the manipulation of polynomials over the rational field, and ask whether it is correct for the computer, for its internal use only, to represent an expression in a fully expanded form. It is obvious that if the expression $a^2 - b^2 - (a + b)(a - b)$ occurs the system should remove the brackets and obtain zero. Furthermore, it is essential that the machine do this because, should it fail to do so and the user's next instruction involve the above expression in some complicated fashion, the result will probably be blow-up. However, it is obviously not always correct to expand brackets as the example of $(1 + x)^{1000}$ demonstrates. Expansion here leads instantly to blow-up. Polynomials are, however, the simple case because the program can in principle expand the expressions it obtains, observe if the result requires less space and discard the less economic representation. This procedure is possible because we have a canonical form for polynomials and we can guarantee to recognise zero. The situation is similar for any class of expression for which a canonical form exists, and for which an algorithm for the reduction of an expression to that form is available. In general such is not the case, and it may be shown (Richardson, 1966, 1968) that for a sufficiently rich class of functions the simplification problem is undecidable in the sense that it is not possible to write a program that will reduce all expressions in the class to a canonical form. Indeed it is not even possible to determine if an expression is zero or not. Things are not in fact as bad as this rather pessimistic result would imply, but the problem should not be underestimated. The simplification of expressions occurring within the computer is of critical importance to avoid blow-up, while the discovery of even approximately simplified forms is an extremely difficult and time-consuming task in many circumstances.

An aspect of the simplification problem that has achieved an independent life of its own is the problem of determining the greatest common divisor (gcd) between two polynomials in several variables. In order to demonstrate the importance of this problem let us consider a very simple example concerned only with rational numbers. To add the two rationals P_1/Q_1 and P_2/Q_2 for integral P_1, P_2, Q_1 and Q_2, the following simple algorithm is sufficient,

$$\frac{P_1}{Q_1} + \frac{P_2}{Q_2} = \frac{P_1Q_2 + P_2Q_1}{Q_1Q_2} .$$

However, when applied to $\frac{1}{2}$ and $\frac{1}{2}$ this yields 4/4 and in order to obtain the expected result we must remove the factor 4. Should we fail to do so then continued calculation with rationals will lead to the representation of numbers as the ratio of very large integers which will ultimately overflow the capacity of the machine's registers. As might be imagined the corresponding computation for rational functions of polynomials leads rapidly to blow-up. The problem, however, is soluble using the Euclidean Algorithm, but this technique turns out to be inordinately expensive in terms of machine time and much work has been devoted to discovering more economical methods (Brown, 1971; Horowitz, 1971b). The work on gcd algorithms for polynomials is also of the greatest importance in another area of symbol manipulation—that of the formal integration of rational functions of polynomials where algorithms depend on the ability to write the integrand in its partially factorised form.

This brings us to the problem of the integration of an algebraic expression in closed form, and here is the great challenge for the system engineer. Integration is the main operation on algebraic expressions that human beings find a real intellectual challenge. However, the problem is clearly defined in a way that simplification is not, and the solution is of great practical value, as has been demonstrated by applications in quantum electrodynamics (Hearn, 1971) and celestial mechanics (Jefferys, 1971b). There is an undecidability result for the problem of the integration of a sufficiently rich class of functions (Richardson, 1966, 1968) but nevertheless, using heuristic techniques, systems have been designed that could integrate better than most college students (Slagle, 1961; Moses, 1967) and recent work indicates that the problem may be capable of total solution within the limits set by the undecidability theorem (Risch, 1968a, b; 1969a, b; 1971). Although the algorithm has yet to be programmed in full, it may be shown that it is possible to discover if a given elementary function has an integral in finite terms and, if it has, to find that integral in a finite series of steps. The joker in this algorithm is closely related to the simplification problem and it is unlikely to cause any trouble with expressions that could be integrated by ordinary techniques by a human being. The work that has been devoted to the integration problem has been rewarded with great success and it is one of the outstanding achievements of this area of computer science.

Let us now examine in a few words the nature of the tool that

is an algebra system and see in what areas of science it is likely to be useful. With this tool we can add, multiply and differentiate algebraic expressions with ease. In many practical cases we can express our results in a simplified form, although this will sometimes prove expensive. We can integrate any expression that a high school student would find easy and, at a little cost, most of those that he would find difficult. Again, at some cost, we can factorise some of our expressions and we can print them out in a legible format provided that they are not too long. Moreover, we can do all of these operations without error under the control of a program that we ourselves write. We need no longer concern ourselves with the length of the expressions that arise, unless they are huge, and the only constraint on the number of operations we perform is our ability to meet the computing bill. In what areas is this tool likely to be useful? The simplest answer to this question lies in the published literature that acknowledges the use of an algebra program but this is not an appropriate place to reference such material in detail. Here it must suffice to direct the reader to bibliographies of the work. FORMAC (Xenakis, 1971) is probably the most widely used system and the bibliography of Sammet (1966a) contains about 300 references to algebra systems and their applications that covers most of the early applications of that pioneering system. ALTRAN (Hall, 1971) has also been widely used and with reference to FORMAC and ALTRAN we must draw the reader's attention to the descriptions, reviews and bibliographies of Hyde (1964), Collins and Griesmer (1966), Tobey (1966a, b); Sammet (1966b, 1967, 1968, 1971), Raphael *et al.* (1968), Marks (1968), Bond and Cundall (1968) and Brown (1969a) which describe the application areas of these two systems. Reviews by Hearn (1971), Jefferys (1971a) and Barton and Fitch (1971) cover applications in particular problem areas and finally a review by the present authors (Barton and Fitch, 1972) gives over 200 references to published work by physicists that acknowledges the use of algebra programs, often for very large calculations indeed. In this latter field of theoretical physics, REDUCE (Hearn, 1970) is outstanding for the number of applications published in quantum electrodynamics while the field of celestial mechanics is outstanding for the number of applications using specially developed Poisson series manipulators for the computations.

This paper is divided into three main sections. In the next section we describe the solution of a number of simple problems using several different algebra systems. Our intention here is two-fold: to introduce the reader to the various algebra systems, and to help him to understand how an algebraic problem must be formulated prior to any computer work. In Section 3 we briefly indicate the manner in which algebra programs have been developed and in which applications have arisen. We then come to the problem of comparing different systems and propose two substantial calculations that could be used for this purpose. Complete programs are presented for the solution of these problems using REDUCE (Hearn, 1970) and SCRATCHPAD (Griesmer and Jenks, 1971). The problems chosen in this section are 'real' problems in the sense that they and their solutions have been previously published in the technical journals of physics. They involve an order of magnitude more algebraic computation than those examined in Section 2, but are substantially less formidable than are the largest known algebraic computations necessary in physics. While these problems are no longer simply demonstration examples there is no doubt that an algebra system should be able to complete them easily if it is to be really useful. In our presentation we have described only the mathematical formulation of the problems. We have tried to avoid technical terminology and to focus attention on the application of the computer. Section 4 presents a brief analysis of the central problems of manipulative algebra from the point of view of a system designer. Here we have confined ourselves to problems that are relevant to applications, but we are aware of the sketchy treatment that even this restricted field has received at our hands.

```
p<0>=1
p<1>=X
n in (2,3,...,5)
p<n>=((2*n-1)*x*p<n-1>-(n-1)*p<n-2>)/n
n in (0,1,...,5)
p<n>

P : 1
 0

P : X
 1

        2
     3X  - 1
P  :  ---------
 2        2

        3
     5X  - 3X
P : ----------
3        2

         4     2
     35X  - 35X  + 3
P  :  ----------------
4            8

          5      3
     63X  - 70X  + 15X
P : -------------------
5           8
```

Fig. 1. The Legendre polynomials obtained by the recurrence relation using SCRATCHPAD. The lines typed by the user appear in lower case and the computer replies in upper case.

We should like to acknowledge the permission granted by the Institute of Physics to reproduce in this review a substantial part of the material of our review (Barton and Fitch, 1972) that appeared in Reports in Progress in Physics.

We should like to express our sincere and grateful thanks to Miss Esther Sadie and Mrs. Vanessa Pearn for their help in the preparation of this manuscript.

2. Elementary examples

2.1. *Introduction*

This section of the paper is essentially tutorial and our purpose is on the one hand to introduce the reader to several different algebra systems and the programming languages with which they are used, while on the other to indicate the type of calculation that it is reasonable to expect of an algebra system. We do this by reference to some particularly simple examples. The section contains several problems that are treated using different algebra systems and appropriately annotated programs are presented for each. These programs are to be read as an integral part of the paper since they demonstrate in detail the communication interface between the user and the various algebra systems. A great deal of effort is currently being devoted to the improvement of this interface and the reader can best familiarise himself with the state of the work by studying the programs themselves. The algebra systems that we discuss are by no means exhaustive of the class of such systems. They have been chosen because they are widely available, or because they are particularly general systems, or because their linguistic features make them attractive. Other systems, frequently those

with which important research has been performed, are referenced in Barton and Fitch (1972). The examples chosen in this section are intended primarily to illustrate how an algebra system is programmed. The early ones are therefore so simple that the semantic features of the programs are obvious and all that concerns us is how to address the individual system in a way that solves the problems. Later in the section other examples of a more complex nature are used to illustrate how problems of manipulative algebra should be broken down to the point where a suitable program can be written. These latter problems are also very simple, of their kind, and it should be mentioned that the reduction of a problem in manipulative algebra to a state where programming can usefully be begun is frequently the most difficult task that one faces in obtaining a solution.

2.2. *Generation of classical functions*

2.2.1. *The Legendre polynomials*

The first problem discussed here is the generation and output of the Legendre polynomials and we do this using two separate algorithms. Perhaps the most obvious means of generating these functions is from the recurrence relation

$$nP_n = (2n - 1)xP_{n-1} - (n - 1)P_{n-2}\,,\ n \geqslant 2$$

with $P_0 = 1$ and $P_1 = x$. **Fig. 1** shows a 'program' written for the SCRATCHPAD system (Griesmer and Jenks, 1971) using this relation, together with the computer's reply. SCRATCHPAD is an interactive system that operates from an IBM 2741 communications terminal and runs on the IBM 360/65 computer at the Thomas J. Watson Research Center at Yorktown Heights. The system is still in the development stage, and as will be seen from the mathematical nature of the input typed by the user, the lines appearing in lower case, is of an advanced design. The main objective of SCRATCHPAD is to formalise a language close to conventional mathematical notation in order that the user may communicate with the algebra system in a convenient fashion. **Fig. 2** presents the protocol for the same calculation performed by the IAM system (Christensen and Karr, 1971). IAM is also an interactive system and it operates from a teletype terminal connected to a PDP 10 computer. This system is currently under development and it is clear from the example that its terminal language is very similar to JOSS (Shaw, 1964). However, in order to use IAM to compute the Legendre polynomials it was necessary to write a small program in which the part of the program that is repeated several times is numbered. From both the SCRATCHPAD and the IAM output we see that our results appear in a two-dimensional format and this is typical of interactive systems.

We now generate the Legendre polynomials using Rodrigues' formula

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2 - 1)^n$$

and for this example we obtain our results using the REDUCE system (Hearn, 1968). The program is presented in **Fig. 3.** REDUCE is also an interactive system operating from a teletype terminal and available on the PDP 10 computer, and it was on that machine that the examples of the system presented in this paper were obtained. However, REDUCE, being based on LISP (McCarthy *et al.*, 1965), is implemented on most other large computer systems. The language of the program (Fig. 3) is similar to ALGOL 60 in its syntactic structure and we see that the notation adopted for differentiation by REDUCE is $DF(F, X)$ for df/dx and $DF(F, X, N)$ for d^nf/dx^n. A linear notation for differentiation similar in character to that of

```
*COMMENT "LEGENDRE POLYNOMIALS BY RECURRENCE RELATION"
*COMMENT "STEP 1.1 DEFINES THE BASIC RECURRENCE RELATION"
*1.1: P(N)←((2*N-1)*X*P(N-1)-(N-1)*P(N-2))/N
*COMMENT "WE NOW DEFINE THE STARTING VALUES OF THE RELATION"
*P(0)←1
*P(1)←X
*COMMENT "NOW ARRANGE TO PRINT THE VALUES OF THE ARRAY P&
&AS THEY ARE GENERATED"
*MONITOR P
*FOR N←2 TO 10, DO PART 1

          2
P(2): 3*X /2 - 1/2

          3
P(3): 5*X /2 - 3*X/2

          4          2
P(4): 35*X /8 - 15*X /4 + 3/8

          5          3
P(5): 63*X /8 - 35*X /4 + 15*X/8
```

Fig. 2. The Legendre polynomials obtained by the recurrence relation using the IAM system. The lines typed by the user appear preceded by an asterisk and the ampersand character is used as a continuation marker. The computer replies by typing the remaining characters and the asterisks. Only part of the computer's output is included in the figures.

```
*COMMENT RODRIGUES' FORMULA FOR THE LEGENDRE POLYNOMIALS;
*ARRAY P(10);
*FOR I←1:10 DO BEGIN
*   P(I)←(X↑2-1)↑I;
*   FOR J←1:I DO P(I)←DF(P(I),X)/(2*J);
*   WRITE "P(",I,")  ",P(I)
*END;
P(1)  X

          2
P(2)  (3*X  - 1)/2

          3
P(3)  (X*(5*X  - 3))/2

          4          2
P(4)  (35*X  - 30*X  + 3)/8

          4          2
P(5)  (X*(63*X  - 70*X  + 15))/8

          6          4          2
P(6)  (231*X  - 315*X  + 105*X  - 5)/16

          6          4          2
P(7)  (X*(429*X  - 693*X  + 315*X  - 35))/16

          8          6          4          2
P(8)  (6435*X  - 12012*X  + 6930*X  - 1260*X  + 35)/128

          8          6          4          2
P(9)  (X*(12155*X  - 25740*X  + 18018*X  - 4620*X  + 315))/128

          10         8          6          4          2
P(10)  (46189*X   - 109395*X  + 90090*X  - 30030*X  + 3465*X  - 63)
      /256
```

Fig. 3. The Legendre polynomials obtained from Rodrigues' formula using REDUCE. The lines typed by the user are preceded by an asterisk. The computer replies by typing the asterisk and the appropriate polynomials.

```
        PROCEDURE MAIN
        ALGEBRAIC (X;10) ARRAY (10) P
        INTEGER N,M
        ALTRAN DIFF
        ALGEBRAIC DIFF
        DO N=1,10
        P(N)=(X*X-1)**N
        DO M=1,N
              P(N)=DIFF(P(N))/(2*M)
        DOEND
        WRITE P(N)
        DOEND
        END


# P(1)
        X

# P(2)
        3*X**2/2 - 1/2

# P(3)
        5*X**3/2 - 3*X/2

# P(4)
        35*X**4/8 - 15*X**2/4 + 3/8

# P(5)
        63*X**5/8 - 35*X**3/4 + 15*X/8
```

Fig. 4. The Legendre polynomials obtained from Rodrigues' formula using ALTRAN. The complete ALTRAN program is submitted to the computer. In this program an array P with 10 elements is declared that depends on the variable x up to the 10th power and the ALTRAN routine for differentiation is requested. The results, only part of which are presented, are obtained in the normal manner for jobs run in batch mode.

```
FOR N=1:1:10
        A[N]=(hh-1)^N
        FOR M=1:1:N
                A[N] (1/(2M))dA[N]/dh
        REPEAT
        PRINT('A[',N,']=',A[N],CRLF)
REPEAT
STOP
START




A[1]=   h
A[2]=   =<1/2-3/2h^2>
A[3]=   =<3/2h-5/2h^3>
A[4]=   <3/8-15/4h^2+35/8h^4>
A[5]=   <15/8h-35/4h^3+63/8h^5>
A[6]=   =<5/16-105/16h^2+315/16h^4-231/16h^6>
A[7]=   =<35/16h-315/16h^3+693/16h^5-429/16h^7>
A[8]=   <35/128-315/32h^2+3465/64h^4-3003/32h^6+6435/128h^8>
A[9]=   <315/128h-1155/32h^3+9009/64h^5-6435/32h^7+12155/128h^9>
A[10]=  =<63/256-3465/256h^2+15015/128h^4-45045/128h^6+109395/256h^8
        =46189/256h^10>
```

Fig. 5. The Legendre polynomials obtained from Rodrigues' formula using CAMAL. In this program multiplication is implicitly understood, and differentiation wrt h is written d/dh. In the PRINT statement the mnemonic CRLF outputs a carriage return and line feed symbol.

```
f(0)=1
g(0)=0
n in (1,2,3,...)
mudot=-3*mu*sigma
sigmadot=eps-2*sigma**2
epsdot=-(mu+2*eps)*sigma
f(n)=mudot*d(mu)f(n-1)+sigmadot*d(sigma)f(n-1)+epsdot*d(eps)f(n-1)-mu*g(n-1)
g(n)=mudot*d(mu)g(n-1)+sigmadot*d(sigma)g(n-1)+epsdot*d(eps)g(n-1)+f(n-1)
```

Fig. 6. The f and g series obtained by recurrence using SCRATCHPAD. The user has typed all the text shown here and the computer's reply is omitted from the figure.

```
*COMMENT "FIRST GIVE THE STARTING VALUES"
*1.1: F(0)←1
*1.2: G(0)←0
*COMMENT "U V W HOLD THE DERIVATIVES OF MU SIGMA EPSILON WRT TIME"
*COMMENT "MU IS REPRESENTED BY THE VARIABLE A"
*COMMENT "SIGMA BY THE VARIABLE B, AND EPSILON BY THE VARIABLE C"
*1.3: U←3*A*B
*1.4: V←C-2*B*B
*1.5: W←-B*(A+2*C)
*1.6: FOR N←0 TO 18,DO PART 2
*2.1: F(N+1)←U*PDERIV(F(N),A)+V*PDERIV(F(N),B)+W*PDERIV(F(N),C)-A*G(N)
*2.2: G(N+1)←U*PDERIV(G(N),A)+V*PDERIV(G(N),B)+W*PDERIV(G(N),C)+F(N)
*COMMENT "ARRANGE TO PRINT THE VALUES AS THEY ARE CALCULATED"
*MONITOR F,G
*DO PART 1
```

Fig. 7. The f and g series obtained by recurrence using the IAM system. The user has typed the lines preceded by an asterisk. Part of the results for this program appear in Fig. 14.

REDUCE is also used by a number of other systems including IAM and SCRATCHPAD. The ALTRAN system (Hall, 1971) developed at the Bell Telephone Laboratories by McIlroy and Brown is a development from ALPAK (Brown, 1963). ALTRAN is written in FORTRAN IV and runs on a wide variety of machines. It is not an interactive system and expects to be presented with a complete program which it then obeys. The results are subsequently obtained in the conventional manner for batch jobs. An example of the input and output for the Legendre polynomials using ALTRAN is presented in **Fig. 4.** The program for another early system is shown in **Fig. 5.** Here the same calculation is performed using the batch processing system CAMAL (Bourne and Horton, 1971b) developed in Cambridge for the TITAN computer. The language used for this program is syntactically similar to TITAN Autocode (Barron *et al.*, 1967). Both CAMAL and ALTRAN have presented their output in a linear form and consequently it is a great deal less legible than that of the previous systems.

2.2.2. *The* f *and* g *series*

A problem that has been used as a test and comparison example for a great number of algebra systems is that of the derivation of the f and g series (Sconzo *et al.*, 1965), that are used in certain expansions of elliptic motion in celestial mechanics. The series themselves are defined by recurrence relations in terms of three variables μ, σ and ε that are related by

```
      PROGRAM FANDG
      COMMON /CR1/ AVAIL,STAK,RECORD(72)
      COMMON /CR2/ SYMLST
      DIMENSION SPACE(15000)
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER PREAD,PVLIST,PDERIV,PPROD,PSUM,PDIF
      INTEGER SPACE,IN,OUT
      INTEGER F,G,MUPOL,MUDOT,SIGDOT,EPSDOT,FDOT,GDOT
      INTEGER VBLS,SIGMA,MU,EPS
      INTEGER FSIG,FMU,FEPS,GSIG,GMU,GEPS
      INTEGER P1,P2,P3,Q1,Q2,Q3,T,U,V
      IN=60.
      OUT=61
      CALL BEGIN(SPACE,15000)
      SYMLST=0
      F=PREAD(IN)
      G=PREAD(IN)
      MUPOL=PREAD(IN)
      MUDOT=PREAD(IN)
      SIGDOT=PREAD(IN)
      EPSDOT=PREAD(IN)
      T=IREAD(IN)
      CALL DECAP(N,T)
      PRINT 14,N
14    FORMAT(/5H N=,I2)
      VBLS=PVLIST(2)
      CALL DECAP(EPS,VBLS)
      CALL DECAP(MU,VBLS)
      CALL DECAP(SIGMA,VBLS)
      I=0.
      GO TO 20
20    FSIG=PDERIV(F,SIGMA)
      FMU=PDERIV(F,MU)
      FEPS=PDERIV(F,EPS)
      P1=PPROD(FSIG,SIGDOT)
      P2=PPROD(FMU,MUDOT)
      P3=PPROD(FEPS,EPSDOT)
      T=PSUM(P1,P2)
      FDOT=PSUM(T,P3)
      CALL PERASE(FSIG)
      CALL PERASE(FMU)
      CALL PERASE(FEPS)
      CALL PERASE(P1)
      CALL PERASE(P2)
      CALL PERASE(P3)
      CALL PERASE(T)
      GSIG=PDERIV(G,SIGMA)
      GMU=PDERIV(G,MU)
      GEPS=PDERIV(G,EPS)
      Q1=PPROD(GSIG,SIGDOT)
      Q2=PPROD(GMU,MUDOT)
      Q3=PPROD(GEPS,EPSDOT)
      T=PSUM(Q1,Q2)
      GDOT=PSUM(T,Q3)
      CALL PERASE(GSIG)
      CALL PERASE(GMU)
      CALL PERASE(GEPS)
      CALL PERASE (Q1)
      CALL PERASE(Q2)
      CALL PERASE(Q3)
      CALL PERASE(T)
      T=PPROD(G,MUPOL)
      U=PDIF(FDOT,T)
      V=PSUM(T,GDOT)
      CALL PERASE(F)
      CALL PERASE(G)
      CALL PERASE(T)
      CALL PERASE(FDOT)
      CALL PERASE(GDOT)
      PWT
      I=I+1
      IF (I-N) 20,20,21
21    STOP
      END
```

```
                                FANDG INPUT

(((+1F**0)M**0)S**0)
*2
(((+1F**0)M**1)S**0)
(((-3F**0)M**1)S**1)
(((-2F**0)M**0)S**2+((+1F**1)M**0)S**0)
(((-1F**0)M**1)*(-2E**1)M**0)S**1)
*12
```

```
                                FANDG OUTPUT

N=12

F SUB 0
(((+1F**0)M**0)S**0)

G SUB 0
*2

F SUB 1
*2

G SUB 1
(((+1F**0)M**0)S**0)

F SUB 2
(((-1F**0)M**1)S**0)

G SUB 2
*0
```

Fig. 8. The f and g series obtained by recurrence using the SAC-1 system. This is a FORTRAN program that calls the several routines of SAC-1. SAC-1 is written in FORTRAN and must be loaded with the above program. The text labelled FANDG INPUT must also be presented to the complete FORTRAN program as data. This data contains the expressions 1, 0, μ, $-3\mu\sigma$, $2\sigma^2+\epsilon$, $(-\mu-2\epsilon)\sigma$ and 12. The significance of these expressions to the computation is self evident. The results presented also appear in the same form as the input.

$$\dot{\mu} = -3\mu\sigma \quad , \quad \dot{\sigma} = \varepsilon - 2\sigma^2 \quad , \quad \dot{\varepsilon} = -\sigma(\mu + 2\varepsilon) \, ,$$

where dot denotes differentiation with respect to time. The recurrence relationships defining the series are

$$f_n = f_{n-1} - \mu g_{n-1} \text{ and } g_n = f_{n-1} + g_{n-1} \, ,$$

together with

$$f_1 = 1 \text{ and } g_1 = 0 \, .$$

These series were first generated as far as f_{27} and g_{27} by Sconzo *et al.* (1965) using FORMAC (Xenakis, 1971) and since then they have been frequently calculated on most other systems. In **Figs. 6 to 12** we present complete programs and specimen

```
*COMMENT THE F AND G SERIES;

*ARRAY FF(19) , GG(19);

*COMMENT HAVING DECLARED THE ARRAYS, AND SPECIFIED THEIR LENGTH,
*WE GIVE THE INITIAL VALUES;

*FF(0)←1; GG(0)←0;

FF(0) ← 1

GG(0) ← 0

*COMMENT THEN THE DERIVATIVES OF MU, SIGMA AND EPSILON WITH RESPECT
*TO TIME;

*MUDOT←-3*MU*SIGMA;

MUDOT ←  - 3*MU*SIGMA

*SIGMADOT←EPS-2*SIGMA↑2;

                        2
SIGMADOT ←  - (2*SIGMA  - EPS)

*EPSDOT←-SIGMA*(MU+2*EPS);

EPSDOT ←  - SIGMA*(MU + 2*EPS)

*COMMENT FINALLY WE GIVE THE RECURRENCE RELATIONS. IN THE FOLLOWING
*LOOP WE ALSO SET FF(N-1) AND GG(N-1) TO.ZERO TO SAVE STORE SPACE;

*FOR N←1:19 DO BEGIN
*   WRITE FF(N)←MUDOT*DF(FF(N-1),MU)+SIGMADOT*DF(FF(N-1)),SIGMA)+
*                 EPSDOT*DF(FF(N-1),EPS)-MU*GG(N-1);
*   WRITE GG(N)←MUDOT*DF(GG(N-1),MU)+SIGMADOT*DF(GG(N-1),SIGMA)+
*                 EPSDOT*DF(GG(N-1),EPS)+FF(N-1);
*   FF(N-1)←0; GG(N-1)←0
*END;

FF(1) ← 0

GG(1) ← 1

FF(2) ←  - MU

GG(2) ← 0

FF(3) ← 3*MU*SIGMA

GG(3) ←  - MU

                           2
FF(4) ← MU*(MU - 15*SIGMA  + 3*EPS)

GG(4) ← 6*MU*SIGMA

                                        2
FF(5) ←  - 15*MU*SIGMA*(MU - 7*SIGMA  + 3*EPS)

                           2
GG(5) ← MU*(MU - 45*SIGMA  + 9*EPS)
```

Fig. 9. The f and g series obtained by recurrence using REDUCE. The user has typed the lines preceded by an asterisk and REDUCE replies by typing the remaining text. Only part of the computer output is presented.

```
F AND G SERIES
DEFLIST (((A 1) (B 2) (C 3)) VAR)
CSET (U (- (* (3 1) A B)))
CSET (V (+ C (- (* (2 1) (** B (2 1)))))))
CSET (W (+ (- (* A B)) (- (* (2 1) B C))))
(LAMBDA ( ) (PROG (X FO GO FN GN)
(SETQ FN (QUOTE (1 1)))
(SETQ GN 0) (SETQ X 0)  A (ARBPRT (QUOTE METRIC) (SETQ FO FN))
(ARBPRT (QUOTE METRIC) (SETQ GO GN))
(COND ((EQUAL (SETQ X (ADD1 X)) 15) (RETURN (NEWPAGE T))))
(ARBTERPRI (QUOTE METRIC)) (ARBTERPRI (QUOTE METRIC))
(SETQ FN (SIMP (LIST PLUSS
(LIST STAR (COPY U) (DIFF (COPY FO) 1))
(LIST STAR (COPY V) (DIFF (COPY FO) 2))
(LIST STAR (COPY W) (DIFF (COPY FO) 3))
(LIST DASH (LIST STAR (QUOTE A) (COPY GO))) )))
(SETQ GN (SIMP (LIST PLUSS
(LIST STAR (COPY U) (DIFF (COPY GO) 1))
(LIST STAR (COPY V) (DIFF (COPY GO) 2))
(LIST STAR (COPY W) (DIFF (COPY GO) 3))
(COPY FO) )))
(GO A) )) ( )
STOP))))))

    3       2         5       2       2 3        3
638A +882BCA +10395AB +1575ABC -3150A B -9450ACB

       2     2 2       2       4       2  3
3150ACB +630A B -225AC -4725AB -54CA -A
```

Fig. 10. The f and g series obtained by recurrence using the CLAM system. The program is written in LISP and calls the LISP routines of which CLAM is composed. The results for f_6 and g_6 are presented.

results for this problem using SCRATCHPAD, IAM, SAC-1 (Collins, 1971a), REDUCE, CLAM (D'Inverno and Russell-Clark, 1971), ALTRAN and CAMAL respectively. The reader should have no trouble in understanding these programs and careful study of them will show the ease with which this type of problem can be programmed for the various systems. However, the problem does raise a technical difficulty and illustrates some of the points made in the introduction.

First let us discuss blow-up. Blow-up of two kinds occurs in this problem. The expressions f_n and g_n become very long and the numerical coefficients of the several terms become very large integers. Neither of these difficulties can be conveniently avoided but it is clear that in order to calculate f_n and g_n we require only the previous series f_{n-1} and g_{n-1}. We can therefore arrange to save some computer memory during the calculation by setting f_{n-1} and g_{n-1} to zero as soon as we have used them for the generation of f_n and g_n. This procedure has been adopted in the REDUCE program (Fig. 9) and could of course have been incorporated into the other programs. However, it

```
PROCEDURE MAIN

ALGEBRAIC (A:100,B:100,C:100) F,G,U,V,W
ARRAY (0:29) F,G

INTEGER N
ALTRAN DIFF

ALGEBRAIC DIFF

F(0) = 1; G(0) = 0; U = -3*A*B; V = C - 2*B**2
W = -B*(A + 2*C)

DO N = 1,19

F(N) = U*DIFF(F(N-1),A) +
        V*DIFF(F(N-1),B) +
       W*DIFF(F(N-1),C) -
       A*G(N-1)

G(N) = U*DIFF(G(N-1),A) +
      V*DIFF(G(N-1),B) +
   W*DIFF(G(N-1),C) +
      F(N-1)

IF(N == 1) WRITE F(1),G(1)
IF(N == 2) WRITE F(2),G(2)
IF(N == 19) WRITE F(19),G(19)
DOEND
END
```

Fig. 11. The f and g series obtained by recurrence using ALTRAN. In this program the variable A represents μ, B represents σ and C represents ϵ. The quantities F and G are declared to be arrays of 20 elements and they, together with U, V and W, are declared to depend upon A, B and C. A, B and C may occur to any power up to 100. The ALTRAN routine for differentiation is requested and the f and g series are computed in the obvious manner. The conditional statements at the end of the program arrange to write out f_n and g_n for = 1, 2 or 19. No results are presented for this program.

```
PROGRAM 0    F[19]  G[19]

1:
| We use the conventions
| a => mu , b => sigma , c => epsilon
        F[0] = 1; G[0] = 0; U = -3ab; V = c-2bb; W = -b(a+2c)
        FOR N = 1:1:19
                F[N] = UdF[N-1]/da + VdF[N-1]/db + WdF[N-1]/dc - aG[N-1]
                G[N] = UdG[N-1]/da + VdG[N-1]/db + WdG[N-1]/dc + F[N-1]
        REPEAT
        PRINT('F[19]= ',F[19],CRLF,'G[19]= ',G[19],CRLF)

STOP
START 1
```

Fig. 12. The f and g series obtained by recurrence using CAMAL. In the first line of the program the arrays F and G, each of 20 elements, are declared. Part of the results of this program appear in Fig. 13.

```
F[19]= <22561587455281874abc'8+349188823373670000a'2bc'7
+18352755073157400a'3bc'6+41307872222988000a'4bc'5+4123481388028380a'5bc'4
+171362658181680a'6bc'3+249276379960a'7bc'2+863831800a'8bc+262143a'9b
-1263448897495785000ab'3c'7-19359533995062030000a'2b'3c'6-10327999258370871000a'3b'3c'5
-2425033032966621000a'4b'3c'4-26102341761249600a'5b'3c'3-12207324890376000a'6b'3c'2
-211207541391000a'7b'3c-903190365000a'8b'3+2034152724968213850000ab'5c'6
+2954067582162935700000a'2b'5c'5+14980490302915714500a'3b'5c'4+333262597045843080000a'4b'5c'3
+3343704006955851000a'5b'5c'2+1384321564653340000a'6b'5c+1754184385107000a'7b'5
-14529662321201527500000ab'7c'5-1912371859938389850000a'2b'7c'4
-86311553814549063000a'3b'7c'3-1645838791362265500000a'4b'7c'2-130111014478044600000a'5b'7c
-3326421776277600000a'6b'7+5448623370450572812500ab'9c'4+6162693189569566650000a'2b'9c'3
+2279813906398945920000a'3b'9c'2+3232069795894539600000a'4b'9c+146020166931714075000a'5b'9
-11491642017677571750000ab'11c'3-1037144834200480005000000a'2b'11c'2
-2758637873711394675000a'3b'11c-21423882877649752500a'4b'11+13701573174923258625000ab'13c'2
+87045288485394819500000a'2b'13c+12358281687185684250000a'3b'13-8612417424237476850000ab'15c
-2870805808079158950000a'2b'15+2216430954766997718750ab'17>
```

Fig. 13. The quantity f_{19} as printed by the CAMAL program of Fig. 12. Here the correspondence $a \equiv \mu$, $b \equiv \sigma$ and $c \equiv \epsilon$ has been employed.

```
F(12):

A↑6 - A↑5*(195195*B↑2 - 2031*C)

+ A↑4*(58108050*B↑4 - 12072060*B↑2*C + 164610*C↑2) - A↑3

*(1640266630*B↑6 - 1122971850*B↑4*C + 159729570*B↑2*C↑2 - 2480958*C↑3)

+ A↑2

*(9820036125*B↑8 - 13315121820*B↑6*C

+ 5298643350*B↑4*C↑2 - 618918300*B↑2*C↑3 + 9951525*C↑4)

- A

*(13749310575*B↑10 - 19462808375*B↑8*C + 21709437750*B↑6*C↑2

-6385128750*B↑4*C↑3 + 638512875*B↑2*C↑4 - 9823275*C↑5)
```

Fig. 14. The quantity f_{12} as printed by the IAM system using the program of Fig. 7. Here the correspondence $A \equiv \mu$, $B \equiv \sigma$ and $C \equiv \epsilon$ has been employed.

```
*COMMENT THE KEPLER EQUATION;

*COMMENT FIRST INTRODUCE THE SIMPLIFICATION RULES FOR SIN AND COS;

*FOR ALL U,V LET COS(U)*COS(V)=(COS(U+V)+COS(U-V))/2;

*FOR ALL U,V LET SIN(U)*SIN(V)=(COS(U-V)-COS(U+V))/2;

*FOR ALL U,V LET SIN(U)*COS(V)=(SIN(U+V)+SIN(U-V))/2;

*FOR ALL U,V LET COS(U)*SIN(V)=(SIN(U+V)+SIN(V-U))/2;

*FOR ALL U LET SIN(U)↑2=(1-COS(2*U))/2;

*FOR ALL U LET COS(U)↑2=(1+COS(2*U))/2;

*COMMENT THE STARTING VALUE IS A=0;

*A←0;

A ← 0

*FOR K←0:3 DO BEGIN
*     LET E↑(K+2)=0;
*     WRITE A←E*SIN(U)*(1-A↑2/2+A↑4/24)+E*COS(U)*(A-A↑3/6+A↑5/120);
*     CLEAR E↑(K+2)
*END;
```

Fig. 15. The solution of the Kepler equation (2.3.1 – 1) by the algorithm (2.3.1 – 4) using REDUCE.

Table 1 A comparison of various algebra systems computing f and g series

SYSTEM	MACHINE	WORD LENGTH IN BITS	CYCLE TIME IN μ SECS	MULTIPLICATION TIME IN μ SECS	ORDER OF f AND g	COMPUTING TIME IN SECONDS	MEMORY REQUIRED IN UNITS OF ONE THOUSAND WORDS	NOTES
ALTRAN Hall (1971)	GE 625/635	36	1·5	6	19	158	51	
CAMAL Bourne and Horton (1971b)	TITAN	48	4	7	19	6·4	3·8	For this program only f_{19} and g_{19} were printed
CLAM D'Inverno and Russell-Clark (1971)	CDC 6600	60	0·8	1	15	10·6	30	
FORMAC Xenakis (1971)	IBM 7094	36	2	10	12	58·2	Not available	
Korsvold's system (1965)	IBM 7094	36	2	10	12	178·2	Not available	
MATHLAB Engelman (1971)	PDP 10	36	1	10	12	20	Not available	
PM Collins (1966)	IBM 7094	36	2	10	27	105	Not available	This program printed all the results
REDUCE Hearn (1970)	PDP 10	36	1	10	10	20	50	A large part of the time was spent printing all the results
SAC-1 Collins (1971a)	CDC 1604	48	4·8	36	12	75·9	21	Standard FORTRAN system
SAC-1 Collins (1971a)	CDC 1604	48	4·8	36	12	38·5	21	The list processing routines were written in assembler code and the remainder was written in FORTRAN

Table 2 The computing time and memory requirements of CAMAL when computing f and g series on the TITAN computer whose word length is 48 bits and whose store cycle time is 4 micro seconds. These figures include 2000 words for the program

ORDER OF f AND g	COMPUTING TIME IN SECONDS	MEMORY REQUIRED FOR EXPRESSIONS IN UNITS OF ONE THOUSAND WORDS	NOTES
10	0·4	0·3	
11	0·4	0·4	
12	1·0	0·5	
13	1·0	0·6	At this point double length fixed point arithmetic is required
14	1·4	0·7	
15	2·0	0·9	
16	3·0	1·1	
17	4·0	1·4	
18	5·0	1·6	
19	6·4	1·8	
20	8·0	2·0	At this point single length floating point arithmetic is introduced
25	17·0	3·0	
30	28·4	4·0	

```
A - E*SIN(U)

A - (E*(E*SIN(2*U) + 2*SIN(U)))/2

                  2                 2
A - ( - E*(E *SIN(U) - 3*E *SIN(3*U) - 4*E*SIN(2*U) - 8*SIN(U)))/8

                  3                 3                 2                 2
A - (E*(E*E *SIN(4*U) - 4*E *SIN(2*U) - 3*E *SIN(U) + 9*E *SIN(3*U)

        + 12*E*SIN(2*U) + 24*SIN(U)))/24
```

Fig. 16. Successive approximations to the solution of the Kepler equation (2.3.1 – 1) produced by the REDUCE program displayed in Fig. 15.

```
PROGRAM 0 A[10]
A[0]=0
FOR K=0:1:5
        %AXORDER=K+1
        A[K+1]=SUBSTITUTE(u,A[K],u,sin(u);K+1)
| The SUBSTITUTE function replaces u by us[K] in the
| expression sin(u). The series is expanded to K+1 terms only
        PRINT['A[',K+1,'] ',A[K+1],CRLF]
REPEAT
STOP
START
```

Fig. 17. The solution of the Kepler equation (2.3.1 – 1) by the algorithm (2.3.1 – 4) using CAMAL. An array of 11 elements named A is declared on the first line of the program and successive approximations to the solution are stored there.

should be noted that in order to make use of this device with the SCRATCHPAD system, it would be necessary to use the conventional program features provided by that system and to abandon the elegant mathematical formulation shown in Fig. 6. The program facilities of SCRATCHPAD are very similar to those of the IAM system (Fig. 7). It is important to understand that the decision to make free the memory occupied by the expressions f_{n-1} and g_{n-1} must be taken by the user of the algebra system, because the systems themselves have no way of discovering that the expressions are no longer required. This aspect of programming for an algebra system has no parallel in ordinary numerical programming where all the variables occupy the same small amount of computer memory. However, it is a point that the algebra programmer must learn to keep in mind at all times during the programming of his problems because it will frequently make the difference between the success and failure of his calculation. The CAMAL system provides a facility to ensure that this idea is easy to implement. In the CAMAL program (Fig. 12) the variables $F[N-1]$ and $G[N-1]$ are written $F:[N-1]$ and $G:[N-1]$ in the assignment to $G[N]$. This notation instructs the CAMAL system to discard the corresponding expressions after they have been used in the particular assignment, and is a shorthand designed to ensure that the meaning of the program remains clear. Of course the responsibility of where to place the colons rests with the user. As an example of the difference that this simple device makes to the performance of an algebra program the CAMAL system requires a total of 3·8K words of memory to generate f_{19} and g_{19} but if all the series are retained in memory it can only manage f_{14} and g_{14} in the same space. Of course, should all the earlier f and g series be required they can always be output to backing store by the various systems and the immediate copy discarded to save memory in the manner described above.

Another difficulty that arises in the computation of f and g series is that of the layout of the results. **Fig. 13** shows the CAMAL output for f_{19}. This is not easy to understand but it is doubtful if it would be significantly improved by a two-dimensional layout. The printout of f_{12} obtained from the IAM system and reproduced in **Fig. 14** is better simply because blank lines are introduced between the terms. It is a subjective decision which of these forms a particular individual prefers but, in the authors' opinion, two-dimensional output is not of very great value for the practical problems of science at present.

It was mentioned in the introduction that there are vast differences between the computation time required for various algebra systems attempting the same problem. In **Table 1** we give a comparison of several systems for the generation of the f and g series together with the names of the machines on which the experiments were conducted. In **Table 2** we give details of the CAMAL system for the generation of f and g series to different orders to aid a fair comparison. The huge range of times indicated in Table 1 is quite typical over the whole range of problems and systems discussed in this paper.

2.3. *Simple repeated approximation*

We turn now to the study of two simple examples of the solution of equations by the method of repeated approximation using an algebra system. We show how it is possible to derive, for each of our examples, an approximation procedure and how this may be programmed for various algebra systems. The methods used would not be suitable for hand calculation because they would involve the mathematician in extensive algebra that he could easily avoid by the use of more powerful mathematical techniques. However, they are entirely suitable for machine calculation because they are much easier to program than the more powerful methods, and the computational labour is undertaken by the computer. The great simplicity of the methods illustrated here demonstrates that manipulative algebra systems are very powerful tools indeed for this type of calculation. Our objectives in presenting these examples are to illustrate the nature of the work involved in solving each problem, and to indicate those parts of the total labour that are performed by the machine. For each problem it is necessary to devise, by hand and without reference to a computer, a certain approximation procedure to solve the equation. Once the procedure is known it can be programmed for a particular algebra system; that program is then run and the algebraic results obtained. The derivation of the approximation procedure is frequently a very complicated task in interesting cases and at present very little work has been done towards automating the process.

2.3.1. *The Kepler equation*

Here we are concerned with the solution for E, as a function of u and e, of the implicit equation

$$E = u + e \sin E, \qquad (2.3.1-1)$$

where e is to be regarded as a small quantity. Equation (2.3.1 – 1) is known as the Kepler equation. The problem is capable of formal solution in terms of Bessel functions in the form

$$E = u + 2\sum_{n=1}^{\infty} \frac{J_n(ne)\sin(nu)}{n}. \qquad (2.3.1-2)$$

In order to obtain E as a function of u and e correct to the 10th order in e it would be perfectly possible to sum the first 10 terms of this series using a computer. However, consider what is involved in this procedure. We must write a program to generate the individual Bessel functions and arrange to ignore those terms of order $k > 10$ that arise in e. Then we must write a program to form the sum indicated above. Such a program is not difficult to write but it is even easier if we adopt a repeated approximation procedure. From the Kepler equation it is clear that $E = u$ to the zero order in e. Suppose $E = u + A_k$ is the solution correct to order k in e. Then clearly

$$A_{k+1} = e \sin(u + A_k) \qquad (2.3.1-3)$$

where the right-hand member of equation (2.3.1 – 3) is to be taken only to order $k + 1$ in e. Thus an approximation algorithm may be stated as follows

$$\left.\begin{aligned} &E = u + \lim_{n\to\infty} A_n \\ &\text{where} \\ &A_0 = 0 \\ &\text{and} \\ &A_{k+1} = \left[e \sin u \left\{ 1 - \frac{A_k^2}{2!} + \frac{A_k^4}{4!} - \ldots \right\} \right. \\ &\qquad \left. + e \cos u \left\{ A_k - \frac{A_k^3}{3!} + \ldots \right\} \right]_{k+1} . \end{aligned}\right\} \qquad (2.3.1-4)$$

```
A[1]   e*sin(u)

A[2]   e*sin(u)
       +1/2e**2*sin(2u)

A[3]   <e-1/8e**3>*sin(u)
       +1/2e**2*sin(2u)
       +3/8e**3*sin(3u)

A[4]   <e-1/8e**3>*sin(u)
       +<1/2e**2-1/6e**4>*sin(2u)
       +3/8e**3*sin(3u)
       +1/3e**4*sin(4u)

A[5]   <e-1/8e**3+1/192e**5>*sin(u)
       +<1/2e**2-1/6e**4>*sin(2u)
       +<3/8e**3-27/128e**5>*sin(3u)
       +1/3e**4*sin(4u)
       +125/384e**5*sin(5u)

A[6]   <e-1/8e**3+1/192e**5>*sin(u)
       +<1/2e**2-1/6e**4+1/48e**6>*sin(2u)
       +<3/8e**3-27/128e**5>*sin(3u)
       +<1/3e**4-4/15e**6>*sin(4u)
       +125/384e**5*sin(5u)
       +27/80e**6*sin(6u)
```

Fig. 18. Successive approximations to the solucion of the Kepler equation (2.3.1 – 1) produced by the CAMAL program displayed in Fig. 17.

```
COMMENT INTRODUCE THE SIMPLIFICATION FOR PRODUCTS OF COSINES;
FORALL U,V LET COS(U)*COS(V)=(COS(U+V)+COS(U-V))/2;

COMMENT GIVE THE FIRST APPROXIMATIONS TO RHO AND C;
RHO-A*COS(V);
C-1;

COMMENT AND OUR PARTICULAR VALUE OF THETA;
THETA-1+2*M+M**2*COS(2*U)+M**4*COS(4*U)+M**6*COS(6*U);

FOR K-0:6 DO BEGIN
   COMMENT ARRANGE TO KEEP ONLY TERMS OF ORDER LESS THAN K+2 IN M;
   LET M**(K+2)=0;
   Y-DF(RHO,U,2)+2*C*DF(DF(RHO,U),V)+C**2*DF(RHO,V,2)+THETA*RHO;

   COMMENT DEAL WITH THE CORRECTION TO THE EIGENVALUE;
   LET COS(V)=0; TEMP-Y; CLEAR COS(V);
   TEMP-Y-TEMP; Y-Y-TEMP;
   C-C+TEMP/(2*A*COS(V));

   COMMENT NOW DEAL WITH THE OTHER RESONANCE TERMS;
   LET COS(-2U+V)=0; TEMP-Y; CLEAR COS(-2U+V);
   TEMP=Y-TEMP; Y-Y-TEMP;
   RHO-RHO-TEMP/(4*M);

   COMMENT FINALLY DEAL WITH THE NORMAL TERMS;
   FORALL N LET COS(2*N*U+V)=COS(2*N*U+V)/(4*N*(N+1));
   TEMP=Y; CLEAR COS(2*N*U+V); Y-TEMP
   WRITE "K+1,"TH APPROXIMATION ", RHO-RHO+Y, "WITH EIGENVALUE ",C;
   CLEAR M**(K+2)
END;
```

Fig. 19. The solution of the Hill equation (2.3.2 – 1) by the algorithm (2.3.2 – 6) using REDUCE. For this example we have chosen

$$\theta = 1 + 2m + \sum_{n=1}^{3} m^{2n} \cos 2nu\,.$$

```
                    PROGRAM 1 I[6]
! e -> e
! rho -> Z
! theta -> X
! The initial values of the approximation
Z=acos(v)
C=1
X=1+2e+e**2cos(2u)+e**4cos(4u)+e**6cos(6u)
FOR K=0:1:6
   ! Arrange to keep only terms of order less than K+2 in e
   SELECT=<e**(K+1)
   ! =u means differentiate wrt u, **v is second derrivative wrt v
   Y=Z**u+2C(Z*u)+v*CCZ**v+XZ
   ! We now take the expression Y to pieces term by term
1: -2 IF Y=0
   ! Get the NEXT Harmonic term,remove it from Y and place it in B
   B=HNEXT(Y)
   ! And find the coefft of u and v,that of u is 2n and that of v is +-1
   HPARSE(B,I[0])
   ! I[1] contains 2n, the coefft of u, I[2] contains +-1, the coeff of v
   ! The next line arranges to deal with the transformation that CAMAL
   ! always makes namely cos(-x)=cos(x)
   I[1]=-I[1] IF I[2]<0
   ! Calculate N using integer division by 2
   N=I[1]%2
   ! Deal with special cases
   -3 IF N=0; -4 IF N=-1
   ! And correct rho in all other cases
   Z=Z+B/(4N(N+1))
   -1 IF Y=0
2: PRINT(K+1,'th approximation to rho',CRLF,CRLF,Z,'
with eigenvalue ',C,CRLF,CRLF)
REPEAT
STOP

! Correct the eigenvalue
3:C=C*(1/2)*DOEFFT(B)/A; -1

! Correct Z IN THE CASE OF THE RESONANCE term in cos(-2u+v)
4:Z=Z-B/(4C); -1
START
```

Fig. 20. The solution of the Hill equation (2.3.2 – 1) by the algorithm (2.3.2 – 6) using CAMAL. In the first line of the program the index array *I* of seven elements is declared. This array is used by the function HPARSE (*B*, *I*[0]) one of whose arguments *B* must have a value of the form

$$P \begin{matrix}\sin\\ \cos\end{matrix} (iu + jv + kw + i'x + j'y + k'z)$$

where *P* is a polynomial coefficient, *i*, *j*, *k*, *i'*, *j'* and *k'* are integers and *u*, *v*, *w*, *x*, *y* and *z* are symbolic variables. The result of HPARSE is to load the integers *i*, *j*, *k*, *i'*, *j'* and *k'* into the array *I*. For this example we have chosen

$$\theta = 1 + 2m + \sum_{n=1}^{3} m^{2n} \cos 2nu$$

and used the correspondence

$$e \equiv m\,.$$

Here the large parentheses and the subscript $k + 1$ indicate that terms of degree greater than $k + 1$ are to be ignored in the calculation. It is normal to express the result of this computation as a linear expression in the trigonometric functions and consequently when the calculation is performed by a computer it is necessary to carry out the linearisation of these functions given by

$$\sin u \sin v = \tfrac{1}{2}(\cos(u - v) - \cos(u + v))$$
$$\cos u \sin v = \tfrac{1}{2}(\sin(u + v) - \sin(u - v)) \quad (2.3.1 - 5)$$

and

$$\cos u \cos v = \tfrac{1}{2}(\cos(u + v) + \cos(u - v))$$

for all u and v.

Let us now consider the programming of the algorithm (2.3.1 – 4) for two algebra systems beginning with REDUCE. The REDUCE system, in common with many others, is aware of the trigonometrical functions and implements automatically such simplifications as $\sin(0) = 0$, $\cos(0) = 1$, $\sin(-u) = -\sin u$ and $\cos(-u) = \cos u$. However it is not aware of the relations (2.3.1 – 5) and so we define the simplifications required on these functions by means of the LET statements at the beginning of the program presented in **Fig. 15.** This ability to define a simplification procedure is an extremely powerful feature of REDUCE that has been duplicated in several other systems. In the next part of the program we set the initial approximation A to zero and then begin the iterative loop. The first operation in the loop is to arrange for the rounding down to order $k + 1$ in e to occur automatically, and this is done by means of another simplification using the LET construction, namely LET $E \uparrow (K + 2) = 0$. Following this statement, whenever terms in e of order greater than $k + 1$ are constructed they are automatically set to zero until the CLEAR instruction is encountered. The body of the program simply arranges to compute the right-hand member of the third equation of (2.3.1 – 4) in the obvious way and to write out the results shown in **Fig. 16.** When the algorithm (2.3.1 – 4) is implemented using the CAMAL system the program is slightly shorter since CAMAL implements the simplification rules between cosine and sine (2.3.1 – 5) automatically and also provides an automatic implementation of the expansion of $\sin(u + A_k)$ in equation (2.3.1 – 3). The rounding down to the particular order in e is performed by the MAXORDER statement in the CAMAL program **Fig. 17** and the expansion of $\sin(u + A_k)$ is performed by the SUBSTITUTE routine. The results produced by CAMAL appear in **Fig. 18.**

2.3.2. *The Hill equation*

In this example the problem is to determine a periodic solution to Hill's equation

$$\ddot{\rho} + \theta\rho = 0\,. \qquad (2.3.2 - 1)$$

where dot denotes differentiation with respect to time.

For the purpose of the example we may assume that

$$\theta = \theta_0 + \sum_{n=1}^{\infty} \theta_n \cos(2nt)\,,\ \theta_0 = 1 + 2m + 0(m^2)\,,$$

and that θ_n are known polynomials in m of degree $2n$. Here m is a small quantity. We seek a solution in the form

$$\rho = \sum_{k=-\infty}^{\infty} A_k \cos(2kt + ct + \xi) \qquad (2.3.2 - 2)$$

involving arbitrary constants A_0 and ξ and a disposable parameter c. The quantity c appears as an eigenvalue of the problem and is used to ensure that a periodic solution is obtained. We shall proceed in powers of m and observe that to order zero in m we have

$$\ddot{\rho} + \rho = 0$$

with solution

$$\rho_0 = A_0 \cos(c_0 t + \xi)$$

and so

$$c_0 = 1 .$$

Assume now that the solutions ρ_k and c_k satisfy equation (2.3.2 – 1) to order k in m and write

$$\rho_{k+1} = \rho_k + \varepsilon ,$$
$$c_{k+1} = c_k + \eta .$$

Further, write u for t and v for $ct + \xi$ then (2.3.2 – 1) may be written

$$\varepsilon_{,uu} + 2\varepsilon_{,uv} + \varepsilon_{,vv} + \varepsilon = 2\eta \cos(v) - Y_{k+1} \qquad (2.3.2 - 3)$$

where

$$Y_{k+1} = [\rho_{k\,uu} + 2c_k\rho_{k,uv} + c_k^2\rho_{k,vv} + \rho_k\theta]_{k+1} , \qquad (2.3.2 - 4)$$

$\varepsilon_{,\alpha\beta} = \partial^2\varepsilon/\partial\alpha\partial\beta$ and the parentheses and subscript mean that terms of order greater than $k + 1$ are to be omitted. From equation (2.3.2 – 4) it is clear that

$$Y_{k+1} = \sum_{n=-\infty}^{\infty} B_n \cos(2nu + v) \qquad (2.3.2 - 5)$$

where the B_n may be easily determined from ρ_k and c_k. The solution to equation (2.3.2 – 3) is then obviously

$$\varepsilon = \sum_{n=-\infty}^{\infty} \frac{B_n \cos(2nu + v)}{4n(n+1)}$$

and this summation must exclude the resonance terms corresponding to $n = 0$ and $n = -1$. The $n = 0$ term may be removed by setting

$$\eta = \tfrac{1}{2}B_0/A_0$$

and this yields the correction to the eigenvalue c. The remaining resonance term may be removed by adding to ρ_k a complementary function $\mathscr{A}\cos(2u - v)$. This term, of degree k in m, will not affect the solution to order k but will contribute an amount $4m\mathscr{A}\cos(2u - v)$ to Y_{k+1}, as a little algebra shows. Thus choosing $\mathscr{A} = B_{-1}/4m$ ensures that the resonance terms are eliminated. We have therefore, the following algorithm for the solution of the Hill equation.

$$\left.\begin{aligned} \rho_0 &= A_0\cos(t + \rho) ,\ c_0 = 1 && \text{a}\\ \rho_{k+1} &= \rho_k + \sum_n \frac{B_n\cos(2nu+v)}{4n(n+1)} - \frac{1}{4m}B_{-1}\cos(2u-v), && \text{b}\\ c_{k+1} &= c_k + \tfrac{1}{2}B_0/A_0 && \text{c}\end{aligned}\right\} \qquad (2.3.2 - 6)$$

and the terms in the summation exclude the cases $n = 0$ and $n = -1$. It should be noted that this algorithm ensures that the result is correct to degree k in m only when the terms of degree $k + 1$ have been partly determined.

To implement the algorithm given the initial values of ρ_0 and c_0 (2.3.2 – 6a) it is simply necessary to compute Y_1 from (2.3.2 – 4) and arrange it in the form (2.3.2 – 5). The B_i are then known and so the expressions (2.3.2 – 6b and c) may be easily constructed. Repeated application of the algorithm yields the result to any desired degree in m. Let us now examine the REDUCE program for the implementation of this algorithm shown in **Fig. 19.** As is usual in problems of this type we begin with the simplification rules for cosine. Then the program proceeds in the obvious way making use of the statement LET $M \uparrow (K + 2) = 0$ to remove unwanted terms. However, a difficulty arises when we need to generate from Y_{k+1}, equation (2.3.2 – 5) and variable Y in Fig. 19, the corrections to ρ and c. It is necessary to separate the terms of the summation, treat the B_0 and B_{-1} terms in their particular ways and reassemble the remainder according to equation (2.3.2 – 6b). This operation can be performed using the LET facility of REDUCE and this is the means adopted in the program. A copy of the variable Y with the term in cos (v) suppressed is obtained by the sequence LET COS $(V) = 0$; TEMP ← Y; CLEAR COS (V). Then the term in cos (v) can be obtained

```
1th approximation to rho
        acos(v)

with eigenvalue          <1+c>


2th approximation to rho

        acos(v)
        +1/16ae^2cos(2u+v)
        -1/8acos(2u-v)

with eigenvalue          <1+c-1/2e^2>


3th approximation to rho

        acos(v)
        +<1/16ae^2-1/32ae^3>cos(2u+v)
        -<1/8ae+1/16ae^2>cos(2u-v)
        -1/128ae^3cos(4u-v)

with eigenvalue          <1+c-1/2e^2+15/32e^3>


4th approximation to rho

        acos(v)
        +<1/16ae^2-1/32ae^3+1/32ae^4>cos(2u+v)
        -<1/8ae+1/16ae^2-15/512ae^3>cos(2u-v)
        +17/768ae^4cos(4u+v)
        -<1/128ae^3-13/256ae^4>cos(4u-v)

with eigenvalue          <1+c-1/2e^2+15/32e^3-19/32e^4>


5th approximation to rho

        acos(v)
        +<1/16ae^2-1/32ae^3+1/32ae^4-93/2048ae^5>cos(2u+v)
        -<1/8ae+1/16ae^2-15/512ae^3+29/1024ae^4-391/16384ae^5>cos(2u-v)
        +<17/768ae^4-37/4608ae^5>cos(4u+v)
        -<1/128ae^3-13/256ae^4-463/8192ae^5>cos(4u-v)
        -17/6144ae^5cos(6u-v)

with eigenvalue          <1+c-1/2e^2+15/32e^3-19/32e^4+1695/2048e^5>


6th approximation to rho

        acos(v)
        +<1/16ae^2-1/32ae^3+1/32ae^4-93/2048ae^5+655/12288ae^6>cos(2u+v)
        -<1/8ae+1/16ae^2-15/512ae^3+29/1024ae^4-391/16384ae^5>cos(2u-v)
        +<17/768ae^4-37/4608ae^5+97/13824ae^6>cos(4u+v)
        -<1/128ae^3-13/256ae^4-463/8192ae^5-105/4096ae^6>cos(4u-v)
        +333/73728ae^6cos(6u+v)
        -<17/6144ae^5-59/3072ae^6>cos(6u-v)
```

Fig. 21. Successive approximations to the solution of the Hill equation (2.3.2 – 1) produced by the CAMAL program displayed in Fig. 20. The correspondence $e \equiv m$ is employed.

independently by the assignment TEMP ← Y − TEMP; A similar technique is used to obtain the term in cos $(2u - v)$ and hence *RHO* and *C* are updated to their new values by elementary methods. An alternative technique is employed by CAMAL because that system provides facilities which enable the several terms of the series Y_{k+1} to be removed and examined independently. The annotated CAMAL program appears in **Fig. 20** and the results in **Fig. 21.**

3. The application of algebra systems

3.1. *Introduction*

Two essentially different paths towards the construction of algebra systems have been followed during the past decade. On the one hand computer scientists developed systems to perform algebraic manipulation with the intention of addressing a wide problem field. These workers soon discovered the limitation of computers in the area. They identified the major problems of implementation, and their work then began to concentrate on the resolution of these problems. Meanwhile a quite separate group of people, drawn from disciplines for which the computer is no more than a necessary tool, encountered problems in their own fields that required huge amounts of algebraic manipulation for their solution. These people set out to build algebra systems to solve those problems and, in those few cases where the major problems of machine algebra were encountered at all, avoided the difficulties by any device at their disposal including fundamental restatement of the initial problem. Subsequently these workers have either returned to their original science when their problem was solved, or in some cases have attempted to generalise their algebra systems to a wider problem area.

The two groups have always learned from each other and each have had their systems improved and extended by the experience obtained from the alternative approach. However, this process is far from complete and currently there is no single algebra system that can successfully perform all the calculations that have been described in the literature. It has been demonstrated by physicists that algebra systems are capable of calculations that enable significant original work to be undertaken in three main fields. These fields are quantum electrodynamics, celestial mechanics and general relativity and

reviews of the applications of algebra programs to these disciplines appear respectively in Hearn (1971), Jefferys (1971a) and Barton and Fitch (1971) together with a review of all three fields in Barton and Fitch (1972). However, while the problem area of many general purpose algebra systems in principle includes some or all of the above fields there is currently no single system that is able to make real progress with the more difficult problems of all three. There remains at present a considerable gap between the capabilities of portable general purpose systems and the requirements of the user community.

A complete list of references to all of the individual applications of algebra programs easily contains many hundreds of entries and we shall confine ourselves here simply to references to reviews of such work. Applications that lie outside the three areas of physics named above are difficult to classify usefully and about all that can be said is that they are drawn from many separate disciplines in the fields of physics, mathematics, engineering and chemistry. As we mentioned in the introduction, reviews of these applications may be found in Sammet (1966a, b, 1967, 1968, 1971), Hyde (1964), Collins and Griesmer (1966), Tobey (1966a, b), Raphael *et al.* (1968), Marks (1968), Bond and Cundall (1968) and Brown (1969a). If we are to generalise then it is clear that the broad field covered by the above reviews represents the applications area of general purpose systems such as FORMAC and ALTRAN while the more specialised fields covered by the reviews of Hearn (1971), Jefferys (1971a) and Barton and Fitch (1971, 1972) represent the applications area of systems designed with a more specific problem field in view. The former reviews cover a broad class of applications giving rise to relatively small quantities of algebraic manipulation while the latter deal with more specific but commonly huge algebraic calculations.

It is clear from the literature that applications have originated either around a particular system or within a particular problem area. In the first case the existence of general purpose systems has led to a proliferation of separate applications, while in the second a particular problem area has led to a proliferation of algebra systems designed to deal with problems in that area. Over the broad applications field FORMAC (Xenakis, 1971) and ALTRAN (Hall, 1971) have been most successful. In quantum electrodynamics REDUCE (Hearn, 1970) is clearly in a pre-eminent position with about a hundred publications acknowledging its use while about a dozen other systems have been constructed to perform the algebra involved in this field. These are listed in full in Barton and Fitch (1972). In celestial mechanics the position is less clear. CAMAL (Bourne and Horton, 1971b) is probably the most powerful system although the major work in the field has been performed using two little known systems namely MAO and ESP (Rom, 1969, 1971) and also using a system developed by Chapront (Kovalevsky, 1968). Again about a dozen other systems have been developed that are listed in Barton and Fitch (1972). Finally in general relativity where a general purpose efficient system is essential, FORMAC (Xenakis, 1971) has been successfully used but the major applications were undertaken with CAMAL (Barton *et al.*, 1970a) and ALAM (D'Inverno, 1970). Although a little slow, REDUCE has been successfully used for calculations in both celestial mechanics and relativity.

In order to produce an algebra system that on the one hand provides a comprehensive range of facilities and so is capable of application over the entire problem field while on the other ensuring that it is not prohibitively expensive we must seek both to improve the runtime efficiency of the general systems and also to extend the power of the efficient systems. However, the difficulty here is, in part at least, one of measurement. Workers in the field generally have their own opinions as to which system is in some sense the most efficient but hitherto these views have rarely been supported by real evidence. Tobey (1971) has firmly pointed out the importance of making exact measurements of the runtime activities of algebra systems in order that some comparative judgement of their efficiency may be obtained and indeed his remarks have already initiated research in this field (Fitch and Garnett, 1972). One way in which we might seek to measure the performance of an algebra system is by running a series of bench mark tests using programs drawn from a wide but well-defined problem class. However, in this type of measurement a serious problem occurs because before such measurements can be undertaken it is necessary to define a suitable problem base and to assemble an appropriate suite of programs. The danger is that in assembling such programs there is a strong tendency to concentrate on elementary problems of the type discussed in Section 2 simply because they are so easy to understand, and to ignore the huge manipulative problems of physics with their attendant data processing and store management difficulties. An objective of this section is therefore to present two problems together with appropriate programs that can be used in any subsequent examinations of system efficiency. These problems have been chosen because they are representative of the calculations performed by workers in the relevant disciplines (celestial mechanics and general relativity); because they may be made very simple by suitable choice of input data, and therefore can be run with any system providing the elementary manipulative facilities that they require; because they can alternatively be made very complicated by a different choice of the input data, and are therefore capable of placing severe strains on an algebra system; because to some extent comparisons are already available for these problems; and, finally, because they can be easily expressed in terms of a few equations or a simple program and they do not require an extensive understanding of the physics from which they are drawn. The programs presented for these computations are respectively written for REDUCE and SCRATCHPAD because these systems are widely known and the programs can be easily understood.

3.2. *A problem for a Poisson series processor*

The first problem presented here is drawn from celestial mechanics and features a special type of algebraic manipulative system known as a Poisson series processor. A Poisson series is a series of the form

$$Q(x, y) = \sum_j P_j \begin{matrix}\cos \\ \sin\end{matrix} (j \cdot y)$$

where x is a vector of *polynomial variables*, y is a vector of *trigonometric variables* and j is a vector of integers. P_j is a polynomial indexed by j in the polynomial variables x. The coefficients of these polynomials may be either rational numbers or floating point numbers, and sometimes P_j is allowed to include negative exponents. These Poisson series are closed under the operations of addition, subtraction, multiplication, differentiation and, if the polynomial variables are treated as small quantities and certain assumptions are made, under division and substitution. Further, provided that there are no side relationships between the variables, every Poisson series can be written in a unique canonical form. Thus the really difficult problems of symbol manipulation do not arise when manipulating Poisson series. Nearly all of the truly enormous calculations of celestial mechanics fall into this area of manipulation and it is for this reason that Poisson series manipulators are important.

As an example of the use of a Poisson series manipulator we present a calculation drawn from the theory of the motion of the Moon. The notation that we adopt will be that conventionally used in celestial mechanics but we shall avoid the use of technical expressions by presenting the problem in terms of equations only. The fundamental problem is to obtain the expansion of a certain function R in terms of certain variables.

The variables concerned are called e, e', γ, and a/a' and these are all to be thought of as polynomial variables. Consequently in subsequent work they may appear only as the arguments of polynomials. These variables are also to be thought of as small quantities and their numerical values are approximately 1/20, 1/60, 1/11 and 1/400. We therefore regard e, e' and γ as quantities of the first order of smallness and a/a' as a quantity of the second order. The function R also depends on six trigonometrical variables namely l, g, h, l', g' and h' and these variables may only occur in linear combinations as the arguments of the functions cosine or sine. Now initially the function R is given in terms of other quantities by the infinite sum

$$R = \frac{a^2}{a'^3}\frac{a'}{r'}\left\{\left(\frac{r}{a}\right)^2\left(\frac{a'}{r'}\right)^2 P_2(S) + \frac{a}{a'}\left(\frac{r}{a}\right)^3\left(\frac{a'}{r'}\right)^3 P_3(S) + \ldots\right\} \qquad (3.2-1)$$

where P_i $(i = 2, 3, \ldots)$ are the well-known Legendre polynomials. We shall see below how to compute r/a, a'/r' and S in terms of the variables we actually require. Now the entire problem is to compute R in terms of the variables specified above to the nth order in small quantities where n is a positive integer.

To compute r/a we must use the two equations

$$l = E - e\sin E \qquad (3.2-2)$$

and

$$\frac{r}{a} = 1 - e\cos E\,. \qquad (3.2-3)$$

We have already seen in Section 2.3.1 how the Kepler equation (3.2 – 2) may be rewritten in the form $E = l +$ (Poisson series in e and l) to a given degree in e. When this result is substituted into (3.2 – 3) and a suitable expansion performed for cos E we obtain a Poisson series for r/a. We are now in a position to obtain a/r in terms of e and l using the relation

$$\frac{a}{r} = \frac{dE}{dl} \qquad (3.2-4)$$

and the expression for a'/r' is obtained by writing primed variables for the unprimed ones in the latter expression. It remains now to demonstrate how S is obtained as a Poisson series. S is defined using certain trigonometric variables f and f' by the equation

$$S = (1-\gamma^2)\cos(f+g+h-f'-g'-h') + \gamma^2\cos(f+g-h+f'+g'+h') \qquad (3.2-5)$$

and f is related to e and l by the equation

$$\frac{df}{dl} = \left(\frac{a}{r}\right)^2 (1-e^2)^{:\,1/2} \qquad (3.2-6)$$

It is clear that we can obtain df/dl as a Poisson series in terms of e and l simply by writing a suitable power series approximation to $(1-e^2)^{\frac{1}{2}}$ in (3.2 – 6). Integration of this Poisson series with respect to l is obviously trivial because of the manner in which the Poisson series involves l. A slight difficulty occurs when we apparently have to integrate the term independent of l as this destroys the Poisson series structure. In fact we are able to avoid it because it may be shown that $f = l +$ (terms periodic in l) obtained by integration of (3.2 – 6). Once we have obtained f in the form $f = l +$ (Poisson series) we can substitute this into (3.2 – 5). Substitution of an equivalent form in terms of primed variables must also be performed for f' and thus we obtain S in the required form. Finally a direct substitution for r/a, a'/r' and S into (3.2 – 1) yields the desired expression for R.

If the calculation is performed to degree n in the small quantities e, e', γ and a/a' then for $n \leqslant 2$ it is easily done by hand, for $2 < n \leqslant 4$ it becomes a substantial hand calculation, for $4 < n \leqslant 6$ it has been done by hand but the calculation probably required about a man-year of effort. For $6 < n \leqslant 8$ the calculation has been performed only by machine. A number

Fig.22 (*a*)

```
ARRAY P(10);
ORDER E,B; FACTOR COS;
COMMENT FIRST INTRODUCE THE SIMPLIFICATION RULES FOR SIN AND COS;
FOR ALL U,V LET COS(U)*COS(V)=(COS(U+V)+COS(U-V))/2;
FOR ALL U,V LET SIN(U)*SIN(V)=(COS(U-V)-COS(U+V))/2;
FOR ALL U,V LET SIN(U)*COS(V)=(SIN(U+V)+SIN(U-V))/2;
FOR ALL U,V LET COS(U)*SIN(V)=(SIN(U+V)+SIN(V-U))/2;
FOR ALL U LET SIN(U)↑2=(1-COS(2*U))/2;
FOR ALL U LET COS(U)↑2=(1+COS(2*U))/2;

COMMENT N IS THE ORDER OF THE DERIVATION;
N←2;

COMMENT CALCULATE THE LEGENDRE POLYNOMIALS BY RODRIGUES' FORMULA AND STORE THEM IN THE ARRAY P
USING A METHOD SIMILAR TO THAT DISPLAYED IN FIGURE 3;
FOR I←2:N/2+2 DO BEGIN  P(I)←(H*H-1)↑I ; FOR J←1:I DO P(I)←DF(P(I),H)/(2*J) END;

COMMENT SOLVE THE KEPLER EQUATION 3.2-2 BY A METHOD SIMILAR TO THAT DISPLAYED IN FIGURE 15,
CALLING THE SOLUTION EE. THIS IS STEP 1;
EE←0;
FOR K←1:N DO BEGIN
    LET E↑(K+1)=0; EE←E*SIN(U)*(1-EE↑2/2+EE↑4/24)+E*COS(U)*(EE-EE↑3/6+EE↑5/120); CLEAR E↑(K+1)
END;

COMMENT ARRANGE TO TRUNCATE EXPRESSIONS TO ORDER N;
WTLEVEL N;

COMMENT CALCULATE R/A A/R AND F, CALLING THEM BB,AA,AND FF USING EQUATIONS 3.2-3,3.2-4 AND 3.2-6
THIS IS STEPS 2 AND 3.
FIRST WE NEED TO CALCULATE (1-E↑2)↑(1/2) BY BINOMIAL EXPANSION FOR USE IN EQUATION 3.2-5.;
DD←-E*E; HH←3/2; J←1; CC←1;
FOR I←1:N/2 DO BEGIN J←I*J; HH←HH-1; CC←CC+HH*DD↑I/J END;
COMMENT NOW USE THE THREE EQUATIONS 3.2-3,3.2-4 AND 3.2-6;
LET E↑(N+1)=0; BB←1-E*COS(U)*(1-EE↑2/2+EE↑4/24)+E*SIN(U)*(EE-EE↑3/6+EE↑5/120);
AA←1+DF(EE,U); FF←AA*AA*CC; CLEAR E↑(N+1);
FORALL Z LET COS(Z)=0; TEMP←FF; CLEAR COS(Z); FF←FF-TEMP;
COMMENT INTEGRATE FF WRT U USING THE LET COMMAND;
FORALL N LET COS(N*U)=SIN(N*U)/N; FF←FF; CLEAR COS(N*U);

COMMENT CALCULATE THE QUANTITIES R' A' A'/R' AND F' BY DIRECT SUBSTITUTION , AND CALL THEM UU,VV,WW;
LET E=B,U=V; UU←BB; VV←AA; WW←FF; CLEAR E,U;

COMMENT CALCULATE BY DIRECT SUBSTITUTION THE EXPRESSION S DEFINED BY EQUATION 3.2-5 AND STORE IT IN XX.
THIS IS STEP 4.;
YY←FF-WW; ZZ←FF+WW;
XX←(1-D*D)*(COS(U-V+W-X-Y+Z)*(1-YY↑2/2+YY↑4/24)-SIN(U-V+W-X-Y+Z)*(YY-YY↑3/6+YY↑5/120))
      +D*D*(COS(U+V+W+X+Y-Z)*(1-ZZ↑2/2+ZZ↑4/24)-SIN(U+V+W+X+Y-Z)*(ZZ-ZZ↑3/6+ZZ↑5/120));

COMMENT THE REMAINDER OF THE CALCULATION IS SIMPLY STEP 5 AND IT CONCLUDES BY PRINTING THE RESULTS.
CALCULATE ZZ←(RA')/(AR') AND YY←(R/A)↑2(A'/R')↑3;
ZZ←BB*VV; YY←ZZ*ZZ*VV;

COMMENT SUBSTITUTE THE EXPRESSION S=XX INTO THE LEGENDRE POLYNOMIALS TAKING CARE OVER
THE ORDER OF THE EXPRESSIONS;
H←XX ; FOR I←2:N/2+2 DO BEGIN WTLEVEL N+4-2I;  P(I)←P(I) END;

COMMENT FINALLY CALCULATE R FROM EQUATION 3.2-1 OMITTING THE FACTOR (M'(A↑2/A'↑3));
WTLEVEL N; FOR I←N/2+2 STEP -1 TO 3 DO BEGIN P(N/2+2)←A*A*ZZ*P(N/2+2)+P(I-1) END;
WRITE YY*P(N/2+2)
```

Fig.22 (*a*)

```
3/4*COS(V)*B - 1/2*COS(U)*E - 3/4*COS(V + U)*E*B - 3/4*COS(V - U)*E

*B + 9/8*COS(2*X + V + 2*Y - U - 2*Z - 2*W)*E*B - 3/8*COS(2*X + V

 + 2*Y - 3*U - 2*Z - 2*W)*E*B - 3/8*COS(2*X + V + 2*Y - 2*U - 2*Z

                                                      2
 - 2*W)*B + 51/8*COS(2*X + 4*V + 2*Y - 2*U - 2*Z - 2*W)*B  -63/8*

COS(2*X + 3*V + 2*Y - U - 2*Z - 2*W)*E*B + 21/8*COS(2*X + 3*V + 2*Y

 - 3*U - 2*Z - 2*W)*E*B + 21/8*COS(2*X + 3*V + 2*Y - 2*U - 2*Z - 2*

W)*B - 9/4*COS(2*X + 2*V + 2*Y - U - 2*Z - 2*W)*E + 3/4*COS(2*X + 2

                             2
*V + 2*Y - 4*U - 2*Z - 2*W)*E  + 3/4*COS(2*X + 2*V + 2*Y - 3*U - 2*

                                                             2
Z + 2*W)*E - COS(2*X + 2*V + 2*Y - 2*U - 2*Z - 2*W)*(15/8*E  +3/2*

 2           2                                         2
D  + 15/8*B  - 3/4) + 3/2*COS(2*X + 2*V + 2*Y - 2*Z)*D  + 15/8*COS(

                            2                          2
2*X + 2*V + 2*Y - 2*Z - 2*W)*E  + 3/2*COS(2*U + 2*W)*D  + 9/8*COS(2

    2                    2                                     2
*V)*B  - 1/8*COS(2*U)*E  + 3/8*COS(X + V + Y - U - Z - W)*A  + 5/8*

                                          2         2         2
COS(3*X + 3*V + 3*Y - 3*U - 3*Z - 3*W)*A  + 3/8*E  - 3/2*D  + 3/8*

 2
B  + 1/4

*
***

*!C
```

Fig. 22. **(*a*) The computation of the Delaunay disturbing function by the method of Section 3.2 using REDUCE. The results are presented to the second order of small quantities. (*b*) The second order terms of the lunar disturbing function as printed by the REDUCE program displayed in (*a*).**

```
(I,J,K,L,P) IN (0,1,2,3).
G<I,J>=0
G<1,1>=EXP(P(R))
G<2,2>=R**2
G<3,3>=(R*SIN(U))**2
G<0,0>=-EXP(Q(R))
X<0>=T
X<1>=R
X<2>=U
X<3>=V
G<I,I>
DF<U>SIN(U)=COS(U)
DF<U>COS(U)=-SIN(U)
DF<R>EXP(R)=EXP(R)
DF<R>P(R)=PP(R)
DF<R>Q(R)=QQ(R)
DF<R>PP(R)=PPP(R)
DF<R>QQ(R)=QQQ(R)
XDF=1
GAM<I,J,K>=GAM<J,I,K> IF I < J
GAM<I,J,K>=(DF<X<I>>G<J,K>+DF<X<J>>G<I,K>-DF<X<K>>G<I,J>)/2 IF I >= J
GAM<1,1,K>
G<;I,J>=0
G<;I,I>=1/G<I,I>
XSUM=1
GAM<I,J;K>=GAM<J,I;K> IF I < J
GAM<I,J;K>=SUM<P>G<;K,P>*GAM<I,J,P>_
       IF I >= J
GAM<1,1;0>
GAM<1,1;1>
GAM<1,1;2>
GAM<1,1;3>
R<I,J,K,L>=-R<J,I,K,L> IF I < J
R<I,J,K,L>=-R<I,J,L,K> IF K < L
R<I,J,K,L>=R<K,L,I,J> IF I+4*J < K+4*L
R<I,J,K,L>=0 IF (I = J) OR (K = L)
R<I,J,K,L>=DF<X<K>>GAM<J,L,I>-DF<X<L>>GAM<J,K,I> _
                   +SUM<P>GAM<J,K;P>*GAM<I,L,P>    _
                   -SUM<P>GAM<J,L;P>*GAM<I,K,P>    _
IF I > J  & K > L & I+4*J >= K+4*L
R<I,J>=R<J,I> IF I < J
R<I,J>=SUM<P>SUM<L>G<;P,L>*R<I,P,J,L> IF I >= J
```

Fig. 23. **The computation of the Ricci tensor using SCRATCHPAD. This program makes use of the equations defining R_{ij} presented in Section 3.2. Using SCRATCHPAD the quantity A^i_j is presented as $A\langle j\,;i\rangle$. Here the user has typed all the text including the definition of differentiation on sine, cosine, exp and the unknown functions $p(r)$ and $q(r)$. When the program is run it prints out g_{ij}, [11, k] and $\left\{\begin{matrix}k\\1\ 1\end{matrix}\right\}$.**

However the results are too long to be included in this figure.

of papers discuss this calculation (Barton, 1966, Jefferys, 1970, and Broucke, 1970), and it is probably fair to say that if a system can complete the case $n = 8$ in a reasonable time then it could also do the Lunar theory completely. The calculation of this function R is very large indeed in the case $n = 8$ when compared with the examples of Section 2 but it is essential to make it clear that the computation is only a trivial first step in solving the current problems of celestial mechanics and unless a system is able to perform the manipulation indicated it is not likely to be of great value in that field. Certainly it will not be competitive with Roms' systems (Rom, 1969, 1971).

To complete this example we now set down the steps required to solve the above problem and present (in **Fig. 22***a*, *b*) the REDUCE program and result for the solution. The steps are as follows:

1. Solve the Kepler equation (3.2 – 2) by the method described in Section 2.3.1.
2. Substitute into (3.2 – 3) to give r/a in terms of e and l.
3. Obtain a/r from (3.2 – 4) and f from (3.2 – 6).
4. Substitute for f and f' into S using (3.2 – 5).
5. Substitute for a'/r', r/a and S into R using (3.2 – 1).

3.3. *A problem for a general purpose system*

The problem we present in this section is drawn from the theory of general relativity but it involves a great deal of algebraic manipulation that, from a human's viewpoint, is essentially very simple. The algebra potentially involves all the elementary functions and consequently if this calculation is to be performed automatically it must be programmed for a system that is able to manipulate these functions and take advantage of their various inter-relationships in the course of its simplification procedure. It is in this area of simplification that the problem tests out a general purpose algebra system as well as testing its runtime efficiency by simply causing a huge amount of algebra to be performed. Once again we shall try to avoid the use of technical expressions by presenting the problem in terms of a set of mathematical equations but we shall refer to the various quantities computed by their usual names.

The problem is essentially concerned with a $4 \wedge 4$ symmetric array of functions g_{ij} $i = 0, 1, 2, 3$ and $j = 0, 1, 2, 3$. There are 10 of these functions and they depend on four independent variables x^i $i = 0, 1, 2, 3$. The functions g_{ij} are the elements of a matrix whose inverse matrix has elements that are written g^{ij}. Clearly we have

$$g^{ij}g_{jk} = \delta^i_k$$

where the repeated superfix and suffix j implies summation over the possible values of j namely 0, 1, 2, and 3. The array δ^i_k is the well-known Kronecker delta array with zeros off the diagonal and ones on it. The summation convention introduced above will be used in what follows without further comment. It is essential to understand that all that is implied by this economical notation for repeated summation are the operations of multiplication and addition. The first part of the problem is to determine the array g^{ij} and to do this we must effectively solve a set of linear equations with algebraic coefficients, and therefore we can normally expect the algebraic form of either the array g^{ij} or the original array g_{ij} to be substantially more complicated than the other unless it should happen that both are diagonal. Furthermore, it is clear that the inverse array g^{ij} must contain the operation of division in the expressions for its elements unless some fortunate coincidence makes this operator redundant. The algebraic expressions are therefore almost bound to be complicated.

We are now able to define a three-dimensional array of functions called the Christoffel symbols of the first kind. These quantities are written $[i, j, k]$ and are given in terms of the first derivatives of the functions g_{ij} by

$$[i, j, k] = \tfrac{1}{2}\left\{\frac{\partial g_{ik}}{\partial x^j} + \frac{\partial g_{jk}}{\partial x^i} - \frac{\partial g_{ij}}{\partial x^k}\right\}$$

where i, j and k range over 0, 1, 2, 3. We note that this array of Christoffel symbols is symmetric in the indices i, j and so there are at most 40 distinct algebraic expressions here. A second array, the Christoffel symbols of the second kind, is defined in terms of the first by

$$\left\{\begin{matrix} k \\ i\ j \end{matrix}\right\} = g^{kp}[ij, p]$$

and satisfies the same symmetry condition. Consider now the Riemann tensor that we define by

$$R_{ijkl} = \frac{\partial}{\partial x^k}[jl, i] - \frac{\partial}{\partial x^l}[jk, i] + [il, p]\left\{\begin{matrix} p \\ j\ k \end{matrix}\right\} - [ik, p]\left\{\begin{matrix} p \\ j\ l \end{matrix}\right\}$$

where i, j, k and l range over 0, 1, 2, 3. This tensor obviously depends on the second derivatives of the original array g_{ij} and a naïve inspection indicates that it is a four-dimensional array with 256 elements. It is clear that the elements of the Riemann tensor are likely to be algebraically complicated and so it is worth attempting to reduce the effort required for its computation as much as possible. In fact it may be shown that owing to a great number of symmetries that are exhibited in the program presented in **Fig. 23** there are only 21 distinct elements in the Riemann tensor. If we are to calculate the tensor on a computer it is absolutely essential to take advantage in our program of these symmetries because we shall then perform far less manipulation and require far less computer memory. Of course cheaper computing machines would make these gains less obvious, but at present there can be no doubt that they are well worth while, even when the extra programming effort required is taken into account. From the Riemann tensor we immediately obtain the Ricci tensor R_{ij} given by

$$R_{ij} = g^{pq}R_{qipj}$$

which is again symmetric in the indices i and j. Next we easily obtain the Ricci scalar R where

$$R = g^{pq}R_{pq}$$

and finally the Einstein tensor given by

$$G_{ij} = R_{ij} - \tfrac{1}{2}Rg_{ij}$$

The Einstein tensor is symmetric in i and j. We observe from these definitions that the Einstein tensor G_{ij} may be computed from the elements of the original array g_{ij} by the operations of addition, subtraction, multiplication and at most two differentiations. The division operator is required only during the inversion of the original array.

The entire algebraic problem is to compute all of the quantities defined above and print them out when once an initial array g_{ij} is set down. In many cases the array g_{ij} is diagonal and although there would be considerable interest in non-diagonal arrays the lengthy algebra involved has normally prohibited their serious study. We here present two distinct sets of functions g_{ij}. For the first of these the above computation is very easy indeed while for the second it is a little more difficult. Several really difficult arrays are given in Harrison (1959). The first array is

$$g_{00} = e^{q(x^1)},\ g_{11} = -e^{p(x^1)},\ g_{ij} = 0 \text{ when } i \neq j,$$
$$g_{22} = -(x^1)^2,\qquad g_{33} = -(x^1 \sin x^2)^2,$$

where $p(x^1)$ and $q(x^1)$ are arbitrary (unknown) functions of x^1. The second array is

$$g_{00} = \frac{V e^{2\beta}}{x^1} - U^2(x^1)^2 e^{2\gamma},\qquad g_{22} = -(x^1)^2 e^{2\gamma}$$

$$g_{33} = -(x^1 \sin x^2)^2 e^{-2\gamma},\qquad g_{01} = g_{10} = e^{2\beta}$$

and

Table 3 A comparison of various algebra systems computing the Ricci tensor and Einstein tensor for the second array displayed in Section 3.3

SYSTEM	MACHINE	WORD LENGTH IN BITS	CYCLE TIME IN μ SECS	MULTIPLICATION TIME IN μ SECS	COMPUTATION TIME IN SECONDS	MEMORY REQUIRED IN UNITS OF ONE THOUSAND WORDS	NOTES
ALAM D'Inverno (1969)	ATLAS 1	48	3	8	240	50	A large amount of the time is spent printing the results
CLAM D'Inverno and Russell-Clark (1971)	CDC 6600	60	0·8	1	18	40	
FORMAC Clemens and Matzner (1967)	IBM 7094	36	2	10	1,800 approx.	Not available	
GRAD ASSISTANT Fletcher (1965)	IBM 7090	36	2·4	34	1,020 approx.	Not available	
CAMAL Barton *et al.* (1970a)	TITAN	48	4	8	140	18	
REDUCE Hearn (1970)	PDP 10	36	1	10	360	70	

$$g_{02} = g_{20} = U(x^1)^2\, e^{2\gamma},$$

with all the remaining $g_{ij} = 0$. The functions U, V, β and γ are arbitrary (unknown) functions of the coordinates x^0, x^1, x^2. The latter data has been used with programs for the above computation on several algebra systems and we reproduce in **Table 3** some statistics obtained by D'Inverno (1969) for these systems. We have augmented the table with the appropriate entries for CAMAL and REDUCE. A program for the above computation using the first set of data and taking advantage of all the symmetry available is presented in Fig. 23. This program is written for the SCRATCHPAD system (Griesmer and Jenks, 1971). Once again it must be emphasised that for an algebra system to be really useful in this field it must be capable of calculations at least an order of magnitude more difficult than either of those mentioned above.

4. Manipulative algebra

4.1. *Introduction*

This section is confined to a discussion of several of the more interesting and difficult problems that impede the development of manipulative algebra systems. There is insufficient space here to enter on a detailed review of all the technical problems of symbol manipulation and unfortunately it has been necessary to omit a great deal of material that is important to the discipline. We have included only those developments that appear to us to be of immediate importance to the applications area or those theorems that indicate the ultimate theoretical bounds of the subject. Much of our material is drawn from the Proceedings of the Second Symposium on Symbolic and Algebraic manipulation held in Los Angeles in 1971. The reader should refer to those proceedings and to the ACM Communications for August 1971 and the Journal for October 1971 for an excellent and complete review of the state of the art.

In Section 1 we mentioned the importance of the areas of simplification, gcd computation, symbolic integration and the input and output of expressions to the applications field. We now discuss these subjects in more detail.

4.2. *Simplification*

Simplification is an active and important area of research in symbol manipulation that frequently presents severe difficulties in the applications area. These difficulties manifest themselves either by causing algebra programs to suffer blow-up and hence to fail to run to completion, or by producing answers that because of their unsimplified nature are incomprehensible. The first difficulty can be avoided in some cases by casting the computation in terms of functions for which a canonical form exists, but not every calculation can be so expressed and some of those for which a recasting is possible are then forced into an unnatural form. The unnatural form is perhaps not important if the result itself is of great value, but if the algebra system is to be used as a desk calculator every effort must be made to ensure that expressions are printed in as comprehensible a fashion as possible. Research into simplification therefore follows two paths. The first seeks to ensure that communication between the user and the machine takes place in a natural manner, and that any results presented to the user can be easily understood. The second seeks to extend the class of functions for which canonical forms are available.

Moses (1971a) has classified the various philosophies of simplification adopted by algebra systems in a particularly graphic manner. Those systems that adopt a doctrinaire attitude towards the internal representation of expressions, by forcing them in all circumstances into a well-defined internal canonical form, he terms *radical*. Radical systems are on the whole early systems—either polynomial, Poisson series, or rational function manipulators, and typical among them are MAO (Rom, 1969), ESP (Rom, 1971), REDUCE I (Hearn, 1968), CAMAL's polynomial and Poisson series modules (Bourne and Horton, 1971a; Barton *et al.*, 1968), ALPAK (Brown *et al.*, 1964), SAC-1 (Collins, 1971a, b) Chapront's system (Kovalevsky, 1968), MATHLAB's rational function system (Martin, 1967) and MACSYMA's rational function system (Martin and Fateman, 1971). The chief advantage of radical systems is simply that they are very efficient and have proven problem solving capability. Their disadvantage is that they frequently find themselves in trouble with the unnecessary expansion of expressions resulting in blow-up. However, their success has stimulated much research into the discovery of canonical forms for larger classes of functions in order to take advantage of the undoubted power of the radical approach. Exactly the opposite view of simplification is adopted by the *conservatives* of Moses' classification. Conservative systems will in no circumstances

attempt to simplify or transform an expression unless explicitly instructed to do so by the user, thus placing the burden of simplification on someone who is probably able but unwilling to bear it. Consequently, the conservatives FAMOUS (Fenichel, 1966) and FORMULA ALGOL (Perlis *et al.*, 1966) have fallen out of use. Between these extremes we have *liberal* systems. The liberals' view is that while some simplification transformations are almost always worthwhile, the majority of complex transformations must be left for the user to initiate. The liberals however appear to have begun to alter their policy, and while a number of user modifiable simplification procedures of a liberal character are in existence (Goldberg, 1959; Hart, 1961; Wooldridge, 1963; Korsvold, 1965), there are few algebra systems of a distinctly liberal flavour left in general use. In their place a party that Moses calls the *new left* has appeared whose view of simplification is a little more liberal than the radicals. New left systems, such as REDUCE II (Hearn, 1970), CAMAL's functions module (Barton *et al.*, 1970a), and ALTRAN (Hall, 1971) give the user a little control over the simplification algorithm by allowing him to prevent some large expansions from being performed but retain most of the transformation power themselves. Because they insist on a nearly standardised internal representation of expressions they are able to conserve most of the efficiency of the radicals while affording a measure of control over blow-up. Finally there are the *catholics* whose aim is to please everybody by the inclusion of multiple simplification algorithms and multiple representations of expressions in their systems. With the intention of addressing a larger problem field they throw efficiency to the wind and they have yet to demonstrate their capabilities on the larger of the problems discussed in Barton and Fitch (1972). Catholic systems indicate a new and exciting future for algebraic manipulation and typical of these systems are SCRATCHPAD (Blair *et al.*, 1970, Griesmer and Jenks, 1971), IAM (Christensen and Karr, 1971) and MACSYMA (Martin and Fateman, 1971).

New left and catholic systems represent the major areas of work today. From the catholics we can expect a vastly improved interface between the scientific user and the algebra system. We have already seen examples of this in the elegant programs for SCRATCHPAD presented in Sections 2 and 3. Progress towards improved formatting of output can also be expected and this will mean far more than simply using a two-dimensional layout. We can expect that machine printed expressions will be ordered in a manner akin to that used in conventional mathematical notation and that the bracketing of the expression will be structured in a natural way. There are huge gains in comprehensibility to be made in the area of machine/user communication but the technical problems are very difficult. Problems no less difficult but rather better defined are encountered by the new left. Their aim is to extend the class of expressions that can be conveniently represented in a canonical form and their search has made considerable progress during the last few years; progress from which both the new left and the catholics can expect to benefit. It is already clear that canonical forms exist for polynomials, Poisson series and rational functions of polynomials, and research has been diverted to more complicated classes of functions. So far there are few absolutely solid results in the field but if certain conjectures of theorems of pure mathematics are adopted a number of useful algorithms can be constructed. The nature of the conjectures is such that they are widely accepted and even if they prove to be false they are unlikely to render the algorithms derived from them useless in everyday circumstances.

In an important paper Brown (1969b) has described a simplification algorithm for the set of nested exponential expressions. These rational exponential or REX expressions are generated from the constants $i = \sqrt{-1}$ and π together with the finite set of variables $x_1, x_2, \ldots, x_n$. In the set of REX expressions the normal arithmetic operations are allowed, namely addition, subtraction, multiplication and division and further the exponential of a REX expression is a REX expression. Thus the trigonometric, hyperbolic and exponential functions are all REX expressions. Brown makes the conjecture that, if $\theta_1, \ldots, \theta_p$ are REX expressions that together with the quantity $i\pi$ are linearly independent over the rationals, then the quantities $e^{\theta_1}, e^{\theta_2}, \ldots, e^{\theta_p}, x_1, x_2, \ldots, x_n$ and $i\pi$ are algebraically independent over the rationals. Using this conjecture Brown demonstrates a simplification algorithm that will always reduce a zero REX expression to zero, but his algorithm as stated does not define a canonical form. Moses (1971a) gives an example of two non-zero REX expressions which, although equal, are reduced to distinct simplified forms by Brown's algorithm and subsequently Moses proves an existence theorem for a canonical reduction of REX expressions. Unfortunately no efficient form of this algorithm for canonical reduction exists at present.

Without making any conjecture at all Caviness (1970) has proved an algorithm for a canonical reduction of a subset of REX expressions. In this set Caviness allows no division and he restricts the functions to contain only one variable x. The constant π is excluded from the set and exponentiation is allowed only to a depth of one. Subsequently Caviness makes the conjecture that if $c_1, \ldots, c_p$ are distinct constants in the set then the quantities $e^{c_1}, \ldots, e^{c_p}$ are linearly independent over the rationals. Using this conjecture he is able to remove the restriction on the depth of nesting of the exponential function and still prove an algorithm for a canonical reduction of the resulting larger set of expressions. Moses (1971a) has proved, again subject to a conjecture, that REX expressions that do not contain i or π and for which no nested exponentials occur may be reduced to a canonical form.

Richardson (1971) has studied the problem of zero determination for a class of functions generated from the independent variable x, the reals and the functions e^x, $\sin x$, $\cos x$ and $\log |x|$. The operations of addition, subtraction, multiplication and division are allowed in Richardson's set and the four functions may be arbitrarily nested to any depth. The expressions so constructed are, however, subject to the restriction that they must be totally defined on the interval being examined, and this means that they remain bounded at all finite points of that interval. Richardson is able to reduce the zero determination problem to that of determining whether or not a constant expression is zero and, as Moses points out, this can normally be done by evaluation. However, the constant problem has not been formally proved decidable and consequently this part of the work remains as a conjecture. Moses *et al.* (1972) have shown that Richardson's work can be extended to include functions defined as solutions to first order differential equations and this work is of importance to the manipulation of the Spence functions that arise in quantum electrodynamics. Johnson (1971) also discusses the problem of zero determination. Other work on simplification theory that may well be of importance to applications in physics appears in Caviness (1967), Bajo *et al.* (1969), Fateman (1971) and Fitch (1971). Finally the bounds of simplification theory have been drawn by Richardson (1968), who proved that the zero determination problem for a large set of functions is undecidable. Richardson's set contains the rational numbers, π, $\sqrt{2}$ and the variable x. Further members of the set are generated by addition, multiplication, the sine function, the logarithm of the absolute value function and nested combinations of these. Richardson's proof is based on a theorem of Davis *et al.* (1961) concerning the solution of exponential diophantine equations. Caviness (1970) has shown that a similar theorem may be proved for the above set of functions excluding $\sqrt{2}$ and replacing the logarithm function by the absolute value function. His proof is based on the unsolvability of Hilbert's Tenth Problem which has recently been demonstrated (Matijasevic, English translation, 1970).

4.3. *GCD algorithms*

In this section we are concerned with some of the problems that confront the designer of a system for the manipulation of rational functions of polynomials. During a computation involving rational functions it is conceptually obvious that it is desirable to remove the greatest common divisor (gcd) from any rational expression that occurs. Clearly few people would be content with a printed result in which this simplification had not taken place but it is by no means clear that each and every intermediate expression obtained in the course of calculation should be so reduced because of the computer time that this would require. Nevertheless, the reduction must be undertaken for many intermediate expressions because failure to extract gcd's from rational functions leads to incomprehensibility in the printed results and blow-up in the intermediate calculations. This point is simply illustrated by the example of the product R_1R_2 where

$$R_1 = \frac{x^3 - 6x^2 + 11x - 6}{x^4 + x + 21}$$

$$R_2 = \frac{x^5 - x^4 + x^2 + 20x - 21}{x^3 - 4x^2 + x + 6}$$

The unreduced product R_1R_2 is

$$\frac{x^8-7x^7+17x^6-16x^5+20x^4-130x^3+340x^2-351x+126}{x^7-4x^6+x^5+7x^4+17x^3-83x^2+27x+126}$$

while the reduced result is

$$R_1R_2 = \frac{x^2 - 2x + 1}{x + 1}$$

Our problem then is to discover an efficient algorithm for the computation of the gcd of two polynomials in one or many variables and it is of very great importance to the applications field. Many calculations in celestial mechanics that are most naturally expressed in terms of rational functions cannot be conveniently solved in that form in the absence of a gcd algorithm, simply because they result in blow-up. It is true that by reformulating the problems a solution can frequently be obtained using a polynomial manipulator, but generally this is at the expense of much greater programming effort on the part of the user. An efficient gcd algorithm is also essential in order that symbolic integration programs can function with efficiency. Extensive use is made of gcd extraction algorithms in programs for the integration of rational functions and in other more general integration procedures. In a continuous research over the past decade Collins, and independently Brown, have proposed steadily improved algorithms for the discovery of gcd's and at present the major difficulties have been substantially overcome. Their major effort will be rapidly absorbed into other algebra systems and their work can be expected to expand substantially the applications field.

The starting point for the work of Brown and Collins is the well-known Euclidean algorithm for the determination of gcd's. This algorithm, when applied to the integers operates as follows. Let I_1 and I_2 be two positive integers with $I_1 \geqslant I_2 > 0$. Construct a sequence of decreasing positive integers I_i, $i = 3, 4, \ldots$ by

$$I_i = I_{i-2} \bmod I_{i-1} .$$

The sequence obviously terminates and the last non-zero element may be shown to be the gcd of I_1 and I_2. This algorithm works well for small integers, but if the integers are large, and several words of computer memory are required for their storage, then the repeated divisions become expensive. Knuth (1969) has shown how this problem may be considerably reduced. With a little modification the Euclidean algorithm may be restated for polynomials over a field. Let P_1 and P_2 be two polynomials with degree $(P_1) \geqslant$ degree $(P_2) > 0$. Construct a sequence of polynomials P_i, $i = 3, 4, \ldots$ by

$$P_{i-2} = H_i P_{i-1} + P_i$$

where H_i is a polynomial and degree $(P_i) <$ degree (P_{i-1}). Again the series terminates and the last element of non-zero degree is the gcd of P_1 and P_2. This algorithm does not work very well. If the polynomials are over the rational field it is generally found that as the calculation proceeds the rational coefficients of the polynomials P_i became the ratios of either large integers in the case of one variable, or polynomials in the multivariable case. Thus it becomes necessary to compute the gcd's of these coefficients, and even then the problem is not totally eliminated. The algorithm is therefore very expensive in terms of computer time. A further serious objection to the algorithm is that it applies only to polynomials over a field. What is needed is an algorithm for polynomials over an integral domain because one then avoids working with rational coefficients and consequently the multivariate case becomes substantially easier. The coefficient growth referred to above was studied and ultimately controlled by Collins (1967), and subsequently he was able to eliminate the problem altogether. Independently Brown achieved a similar result and Knuth (1969) describes their so-called *modular* algorithm.

Once again we do not attempt a complete revision of work in this field but refer the reader to Jordan *et al.* (1966) to Brown's tutorial paper (Brown, 1971) and to Horowitz (1971a) for more detailed information. Essentially two lines of attack on the gcd problem have been developed. One based on a theory of subresultants and representing a development of the Euclidean algorithm (Collins, 1971b; Brown and Traub, 1971), and the other based on the use of modular arithmetic (Knuth, 1969) that maps the given polynomials into polynomials over a simpler domain. The first of these methods controls the coefficient growth that is the principal defect of the Euclidean algorithm and in the multivariate case it eliminates the need for recursive application of the gcd procedure. The modular algorithm reduces the problem to that of computing the gcd of two polynomials in one variable over $GF(p)$ for some prime p. This latter calculation is easily performed by the Euclidean algorithm since $GF(p)$ is a field in which coefficients cannot grow. Brown (1971) concludes that the two methods are of equal value, the subresultant algorithm giving faster results in the case of a large gcd and the modular algorithm proving more economic in the case of mutually prime polynomials.

4.4. *Symbolic integration*

The problem of symbolic integration has been extensively studied during the past decade by Moses (1967, 1969), Risch (1968a, b; 1969a, b; 1971), Manove *et al.* (1968), Horowitz (1971b) and Tobey (1967). Very substantial progress has been made in this field and it is now possible to integrate a great variety of complicated expressions automatically. The most successful program to date is Moses' symbolic integration program SIN that followed Slagle's program SAINT (Slagle, 1961). SIN emulates in many ways the strategy adopted by a mathematician in performing an integration, and this method has a great deal to recommend its use. When the program is successful it produces results that are expressed in terms similar to those of the integrand. Thus SIN produces the 'expected' results, and these are easily comprehensible. On the other hand when an algorithmic approach to integration is adopted simple integrands are generally transformed prior to integration, and because the results are expressed in unexpected terms they are frequently difficult to understand. However, SIN is unable to decide whether or not a general function is integrable in closed form and so it must sometimes fail. In order to deal with the situation Moses has recently begun to program an algorithmic integration procedure into the latter stage of SIN which can take over when the heuristic methods fail. The algorithmic

integration procedure is due to Risch who has formalised a method of Liouville. However, only parts of the method have currently been implemented.

SIN proceeds in three stages to integrate a given elementary expression. In the first stage the integral is expanded to its fullest extent and then the integration operator is commuted with addition, thus

$$\int (x + \sin x)^2\, dx \rightarrow \int (x^2 + 2x \sin x + \sin^2 x)\, dx$$
$$\rightarrow \int x^2\, dx + \int 2x \sin x\, dx + \int \sin^2 x\, dx$$

These simplified integrals are then examined to see if any of them match the form

$$\int cf(u(x))\, u'(x)\, dx$$

for constant c. Here $u(x)$ is any elementary expression and $f(x)$ is any elementary function. If a match is achieved the integral is trivial. If however, no such match occurs the second stage of SIN is entered. During this stage SIN determines on the basis of the type of expression occurring in the integrand which of 11 integration methods might be suitable for the problem at hand. Having made the selection the appropriate method is implemented and this generally causes a transformation that yields a simple integral. For example, given

$$\int \sqrt{\frac{x+1}{2x+3}}\, dx$$

SIN will transform according to

$$y = \sqrt{\frac{x+1}{2x+3}}$$

and obtain

$$\int \frac{y^2}{(2y^2-1)^2}\, dy\,.$$

If SIN is successful in obtaining a rational expression to integrate as a result of one of the transformations then Manove's implementation of the Hermite algorithm for the integration of rational functions is entered (Manove *et al.*, 1968). Otherwise, the new integral is submitted to the first stage of SIN again. If both the first and second stages of SIN fail then the final stage is entered. Here Moses originally anticipated the Risch algorithm and used his so called educated guess or EDGE heuristic. The purpose of EDGE was to inspect the integrand and, on the basis of the form and combination of the functions present, to make a guess at the form of the integral. This done, EDGE inserted into the guess a number of constants and after differentiating the guessed solution tried to discover suitable values for those constants. Later versions of SIN have replaced EDGE with parts of the Risch algorithm.

A central and important part of the second stage of SIN is the Hermite algorithm for the integration of rational functions. This well-known algorithm will not be reproduced here but two remarks must be made about its application. Our first point is concerned with gcd extraction. Given an integrand $R(x)/S(x)$ where R and S are polynomials in x with degree (R) < degree (S) it may be shown that we can express the integral in the form

$$\int \frac{R}{S}\, dx = f(x) + \sum_{j=1}^{k} \int \frac{V_j(x)}{W_j(x)}\, dx$$

by partial factorisation and integration by parts. Here $f(x)$ is a known rational function of x, $V_j(x)$ and $W_j(x)$ are known polynomials in x with degree V_j < degree W_j and W_j has only simple roots. In the course of this reduction frequent extensive use is made of a gcd algorithm for polynomials, and it is here that a major application of the methods of Collins and Brown, discussed in Section 4.3 occurs. Our second point is concerned with the factorisation of the quantities $W_j(x)$. If these are already linear functions then the integration problem obviously becomes trivial. In other cases it is known that the polynomials W_j contain only simple roots, and consequently the integration algorithm depends on our being able to discover the factors of such polynomials. Manove uses the Kronecker factorisation procedure to determine linear factors (Van der Waerden, 1949). Berlekamp (1971), Brown and Graham (1969) and Zimmer (1971) have discussed alternative factorisation algorithms. However, as Tobey (1967) indicates, the factorisation problem is unavoidable in most circumstances, and of course in general a polynomial of degree $\geqslant 5$ cannot be factored in terms of radicals. These difficulties of the rational integration algorithm detract little from the power of SIN and Moses' program is able to integrate most of the examples normally presented to sophomores. It is unnecessary to say that it is very much quicker.

Turning now to the algorithmic approach to integration that is embodied in the Risch papers, we encounter a contribution to symbol manipulation of the greatest importance. The algorithm is based on Liouville's theorem which states that if f is a function in a field of elementary functions F then

$$\int f\, dx = V_0 + \sum_{i=1}^{n} C_i \log V_i$$

where $V_i \in F$, $(i = 0, 1, \ldots, n)$ and the C_i are constants. Risch's algorithm is complex and we shall try to give the flavour of it by example. Let F be the rational field extended by the simple variable x. Thus F is the field of rational expressions in x. Suppose that we have to integrate $f(x)$, a given rational function of $\log x$ with coefficients in F. This function is not a member of F because it contains the function $\log x$. The function $\log x$ is used to form a new field $\mathscr{F}$, an extension of the field F, and since $\log x$ is not the root of an algebraic equation in F it is said to form a *monomial* extension of F. (New functions that do satisfy algebraic equations in F give rise to *algebraic* extensions of F and are separately treated by the algorithm.) We observe from the function $f(x)$ that, while it is not a member of F it is a member of the *monomial* extension $\mathscr{F}$ of F formed by augmenting the field F by the function $\log x$. The Liouville theorem tells us that

$$\int f(x)\, dx = V_0 + \sum_{i=1}^{n} C_i \log V_i$$

where $V_i \in F$ and C_i are constants. The Risch algorithm takes advantage of the rational property of the function $f(x)$ in $\log x$ over F. Thus we note

$$f(x) = P(\log x) + \frac{Q(\log x)}{R(\log x)}$$

where P, Q and R are polynomials in $\log x$ over F and degree Q < degree R Risch's algorithm now asserts that

$$\int f(x)\, dx = \mathscr{P}(\log x) + \frac{\mathscr{Q}(\log x)}{\mathscr{R}(\log x)} + \sum_{i=1}^{n} C_i \log V_i$$

where $\mathscr{P}$, $\mathscr{Q}$, $\mathscr{R}$ and V_i are polynomials in $\log x$ over F, degree $\mathscr{Q}$ < degree $\mathscr{R}$ and the C_i are constants. The algorithm proceeds to explain how the functions $\mathscr{Q}$, $\mathscr{R}$ and the V_i together with the constants C_i may be determined from Q and R by a method similar to the Hermite algorithm for the integration of rational functions of x. As in the Hermite algorithm, a gcd procedure is necessary here. The Risch algorithm then demonstrates that the polynomial $\mathscr{P}$ can be constructed from the polynomial P *provided only that formal integration in F is possible* and these latter integrals are simpler because they *cannot contain* $\log x$ *in the integrand.* A monomial extension to F can also be produced by the function e^x and the method of integrating expressions containing e^x is similar to that described for $\log x$. Again the determination of the corresponding polynomial $\mathscr{P}(e^x)$

reduces to integration in F. However, in more complex cases a serious difficulty arises here (Moses, 1971b; Risch, 1968a).

Now let us suppose that F extended by $\log x$ to $\mathscr{F}$ is further extended by e^x to $\mathscr{F}'$. The above argument indicates that we can reduce integration in $\mathscr{F}'$ to integration in $\mathscr{F}$ and then integration in $\mathscr{F}$ to integration in F. So we can solve the problem of integration with two monomial extensions. Clearly this procedure can be adopted for any finite number of monomial extensions, and so, given any arbitrary function we need only identify the monomials, arrange them in some order and apply the algorithm recursively. However, Risch's algorithm requires that the monomials are algebraically independent, and the recognition of a suitable set of algebraically independent monomials in general begs the constant problem of simplification introduced in Section 4.2.

The Risch algorithm depends heavily on the properties of the exponential and logarithmic functions under differentiation and it is a defect of the method that in order to make use of it any proposed integrand must first be written explicitly in terms of those functions. Thus the basic structure of a trigonometrical integrand is immediately lost and the integral, while correct, may well be incomprehensible. The exponential and logarithmic functions are not the only functions capable of being used to generate extensions to the basic field and Moses (1971b) gives an example of the error function so employed. Indeed, there is hope that the Spence functions of quantum electrodynamics may be capable of treatment by this method. So far as is known to the present authors no complete implementation of the Risch algorithm exists and the most advanced program is in the third stage of Moses' SIN, but several groups are presently engaged on separate implementations of the procedure and in the near future it is expected that general purpose systems such as REDUCE, CAMAL, SCRATCHPAD, and SIN will contain the full fruits of Risch's work.

4.5. *Mathematical input and output*

Most scientific disciplines employ some form of two-dimensional notation and it is clearly a desirable objective to enable a computer to understand such hand written symbolism and to construct it as output on a suitable terminal. This is the problem to which we now address ourselves. As might be expected the easier problem here is that of constructing automatically two-dimensional symbolic output, and indeed we have already seen examples of this produced by the REDUCE, SCRATCHPAD and IAM systems. The problem of recognition of hand written input is far more difficult, and there are no general purpose systems freely available today that provide this capability. Returning to the output problem the major limitation at present seems to be an economic one. In order to obtain really satisfactory two-dimensional output that faithfully represents what is technically possible from the programming point of view it is necessary to provide the user with an expensive display terminal. On such a terminal characters of arbitrary size may be displayed at arbitrary positions and orientations (Martin, 1967 and Siret, 1971). At present it is quite impossible to produce that degree of flexibility using a conventional typewriter. In practice using a typewriter terminal system such as Millen's CHARYBDIS (1968) two-dimensional output may be obtained that certainly substantially improves the legibility of short expressions, but two-dimensional formatting enables little to be gained in this direction when a long expression is encountered, and most practical systems return to a linear representation in that case. Two-dimensional output is an area in which the techniques are straightforward and are developed beyond the capability of inexpensive hardware; we must wait for the development of an appropriate terminal. The problem of hand written input is a rather different story. Suitable hardware exists in the form of 'tablets' on which the user 'draws' with a 'pen'. The position of the pen on the tablet is discovered by the computer that is able to detect either light, sound or variations of electric field caused by motion of the pen. The drawing then appears on a display screen that may or may not be located below the pen. As might be expected these devices are also expensive and they are still in the experimental stage. However, their existence acts as a stimulus to system designers who must endeavour to translate the two-dimensional input produced by the equipment into an intelligible form in the computer.

Substantial work has been carried out in this area by Anderson (1968, 1971), Blackwell and Anderson (1969), Bernstein and Williams (1968), Bernstein and Howell (1968), Bernstein (1971) and Klerer and Grossman (1967). Apart from the serious problem of the accurate recognition of a hand drawn character two classes of difficulties arise in the input of mathematical expressions. First there are the software problems that arise from ambiguous expressions inherent in the great richness of mathematical notation and then there are the expressions that are badly written by the user on the tablet itself. These problems are difficult but they are by no means technical impossibilities, and it can be expected that in the next few years considerable progress will be made towards their solution. However, we cannot expect to be able to communicate with a computer freely and easily in a two-dimensional format for some years yet. For a complete review of the current state of the art in this area, the reader is referred to Martin (1971).

References

ANDERSON R H 1968 Proc. ACM Symposium on Interactive Systems for Experimental Applied Mathematics (eds M Klerer and J Reinfelds) (New York: Academic Press) 436-59

ANDERSON R H 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation (ed S R Petrick) (New York: Ass Comput Mach) 100-1

BAJO S, MAIOCCHI M, POZZI G and QUILICI U 1969 *Calcolo* **6** 391-401

BARRON D, BROWN H, HARTLEY D F and SWINNERTON-DYER H P F 1967 TITAN Autocode Programming Manual 3rd. ed. University Computer Laboratory Cambridge

BARTON D 1966 *Astr. J.* **71** 438-42

BARTON D, BOURNE S R and BURGESS C J 1968 *Comput. J.* **11** 293-8

BARTON D, BOURNE S R and FITCH J P 1970a *Comput. J.* **13** 32-9

BARTON D, BOURNE S R and HORTON J R 1970b *Comput. J.* **13** 243-7

BARTON D and FITCH J P 1971 *Communs. Ass. Comput. Mach.* **14** 542-7

BARTON D and FITCH J P 1972 Reports on Progress in Physics **35** 235-314

BERLEKAMP E R 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 223

BERNSTEIN M I 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 102-3

BERNSTEIN M I and HOWELL H L 1968 Report TM-3937/000/00 Systems Development Corporation Santa Monica

BERNSTEIN M I and WILLIAMS T G 1968, Proc. 68 IFIP Congress (Amsterdam: North-Holland) C84-9

BLACKWELL F W and ANDERSON R H 1969, Proc. 24th. ACM Nat. Conf. (New York: Ass Comput Mach) 551-7

BLAIR F W, GRIESMER J H and JENKS R D 1970 Proc. ACM Int. Comput. Symp. (Berlinghoven: Gesellschaft fur Datenverarbeitung) **2** 393-420

BOND E R and CUNDALL P A 1968 Symbol Manipulation Languages and Techniques (Amsterdam: North Holland) pp116-32

BOURNE S R and HORTON J R 1971a CAMAL Manual Cambridge University

BOURNE S R and HORTON J R 1971b Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 134-43
BROUCKE R A 1970 *Celestial Mech.* **2** 9-20
BROWN W S 1963 *Bell Syst. Tech. J.* **42** 2081-2119
BROWN W S 1969a Bell Telephone Laboratories Inc. Report AL-69-1.10
BROWN W S 1969b *Am. math. Mon.* **76** 28-34
BROWN W S 1971 *J. Ass. Comput. Mach.* **18** 478-504
BROWN W S and GRAHAM R L 1969 *Am. math. Mon.* **76** 795-7
BROWN W S and TRAUB J F 1971 *J. Ass. Comput. Mach.* **18** 505-14
BROWN W S, HYDE J P and TAGUE B A 1964 *Bell Syst. Tech. J.* **43** 785-804
CAVINESS B F 1967 Ph.D. Thesis Carnegie-Mellon University Pittsburg
CAVINESS B F 1970 *J. Ass. Comput. Mach.* **17** 385-96
CHRISTENSEN C and KARR M 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 115-27
CLEMENS R and MATZNER 1967 University of Maryland Tech. Report No 635
COLLINS G E 1965 Research Report RC-1436 IBM Watson Research Center
COLLINS G E 1966 *Communs. Ass. Mach.* **9** 578-89
COLLINS G E 1967 *J. Ass. Comput. Mach.* **14** 128-42
COLLINS G E 1971a Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 144-52
COLLINS G E 1971b *J. Ass. Comput. Mach.* **18** 515-32
COLLINS G E and GRIESMER J H 1966 SICSAM bulletin No: 4 (New York: Ass Comput Mach)
DAVIS M, PUTNAM H and ROBINSON J 1961 *Ann. Math.* **74** 425-36
D'INVERNO R A 1969 *Comput. J.* **12** 124-7
D'INVERNO R A 1970 ALAM Programmer's Manual
D'INVERNO R A and RUSSELL-CLARK R A 1971 CLAM Programmer's Manual King's College London
ENGELMAN C 1971 Proc. 2nd Symp. on Symbolic and Algebraic Manipulation 29-41
FATEMAN R J 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 311-23
FENICHEL R 1966 Ph.D. Thesis Harvard University
FITCH J P 1971 Ph.D. Thesis Cambridge University
FITCH J P and GARNETT D J 1972 Proc. ACM Int. Comput. Symp. April Venice.
FLETCHER J G 1965 University of California Lawrence Radiation Lab Report UCRL-14624-T
GOLDBERG S H 1959 M.S. Thesis MIT
GRIESMER J H and JENKS R D 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 42-58
HALL A D 1971 *Communs. Ass. Comput. Mach.* **14** 517-21
HARRISON B K 1959 *Phys. Rev.* **116** 1285-96
HART T 1961 Project MAC Artificial Intelligence Group Memo 27, MIT
HEARN A C 1968 Proc. ACM Symposium on Interactive Systems for Experimental Applied Mathematics (eds. M Klerer and J Reinfelds) (New York: Academic Press) 79-90
HEARN A C 1970 REDUCE Users' Manual, Stanford Artificial Intelligence Project Memo AIM-133
HEARN A C 1971 *Communs. Ass. Comput. Mach.* **14** 511-6
HOROWITZ E 1971a Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 188-94
HOROWITZ E 1971b Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 411-57
HYDE J P 1964 *Bell Syst. Tech. J.* **43** 1547-62
JEFFERYS W H 1970 *Celestial Mech.* **2** 474-80
JEFFERYS W H 1971a *Communs. Ass. Comput. Mach.* **14** 538-41
JEFFERYS W H 1971b *Celestial Mech.* **3** 390-4
JOHNSON S C 1971 *J. Ass. Comput. Mach.* **18** 559-65
JORDAN D E, KAIN R Y and CLAPP L C 1966 *Communs. Ass. Comput. Mach.* **9** 638-43
KLERER M and GROSSMAN F 1967 Proc. fall jt. Computer Conf. 675-87
KNUTH D E 1969 The Art of Computer Programming Vol 2 (Reading Mass.: Addison-Wesley)
KORSVOLD K 1965 Stanford Artificial Intelligence Project Memo AIM-37
KOVALEVSKY J 1968 *Astr. J.* **73** 203-9
MCCARTHY J, ABRAHAMS P W, EDWARDS D J, HART T P and LEVIN M I 1965 LISP 1.5 Programmer's Manual 2nd. edition MIT Press
MANOVE M, BLOOM S and ENGELMAN C 1968 Proc. 1966 IFIPS Conf. on Symbolic Manipulation Languages (ed D G Bobrow) (Amsterdam: North-Holland) 86-102
MARKS P 1968 Proc. of Summer Institute on Symbolic Mathematical Computation Cambridge Mass. (ed R. G. Tobey) (Gaithersburg: IBM) 21-38
MARTIN W A 1967 Project MAC Report MAC-TR-36 MIT
MARTIN W A 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 78-87
MARTIN W A and FATEMAN R J 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 59-75
MATIJASEVIC J V 1970 (trans.) *Soviet Math. Dokl.* **11** 354-7
MESZTENYI C K 1971 FORMAL user's Manual University of Maryland
MILLEN J K 1968 Proc. ACM Symposium on Interactive Systems for Experimental Applied Mathematics (eds M. Klerer and J. Reinfelds) (New York: Academic Press) 155-66
MOSES J 1967 Project MAC Report MAC-TR-47
MOSES J 1969 Tutorial Session July 2 Summer Institute in Dynamical Astronomy MIT
MOSES J 1971a *Communs. Ass. Comput. Mach.* **14** 527-37
MOSES J 1971b *Communs. Ass. Comput. Mach.* **14** 548-60
MOSES J, ROTHSCHILD L P and SCHROEPPEL R 1972 in preparation
PERLIS A, ITTURIASA R and STANDISH T A 1966 *Communs. Ass. Comput. Mach.* **9** p549
RAPHAEL B, BOBROW D G, FEIN L and YOUNG J W 1968 Symbol Manipulation Languages and Techniques (Amsterdam: North-Holland) p1-54
RICHARDSON D 1966 Ph.D. Thesis University of Bristol
RICHARDSON D 1968 *J. Symb. Log.* **33** 511-20
RICHARDSON D 1971 *Z. math. Logik* **17** 133-6
RISCH R H 1968a SDC Report SP-2801-002

RISCH R H 1968b Proc. Summer Institute in Symbolic Mathematical Computation Cambridge Mass. 133-48
RISCH R H 1969a *Trans. Am. math. Soc.* **139** 167-89
RISCH R H 1969b IBM Corp. Report RC 2402
RISCH R H 1971 Bull. Am. math. Soc. to be published
ROM A 1969 *Celestial Mech.* **1** 301-19
ROM A 1971 *Celestial Mech.* **3** 331-45
SAMMET J E 1966a *Computg. Rev.* **7** BIB 11 BI-17
SAMMET J E 1966b *Communs. Ass. Comput. Mach.* **9** 555-69
SAMMET J E 1967 *Adv. Comp.* **8** 47-102
SAMMET J E 1968 Symbol Manipulation Languages and Techniques (Amsterdam: North-Holland) pp55-63
SAMMET J E 1971 Mathematical Software (New York and London: Academic Press) pp295-330
SCONZO P, LE SCHACK A and TOBEY R 1965 *Astr. J.* **70** 269-71
SHAW J C 1964 Proc. AFIPS fall jt. Computer Conf. (Baltimore: Spartan Books Inc) 455-64
SIRET Y 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 90-9
SLAGLE J 1961 Ph.D. Thesis MIT
TOBEY R G 1966a *Communs. Ass. Comput. Mach.* **9** 589-97
TOBEY R G 1966b *Communs. Ass. Comput. Mach.* **9** 742-51
TOBEY R G 1967 Ph.D. Thesis Harvard University
TOBEY R G 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 1-16
VAN DER WAERDEN B L 1949 Modern Algebra, Vol. 1 (New York: Ungar)
WOOLDRIDGE D 1963 Stanford Artificial Intelligence Project memo AIM-11
XENAKIS J 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 105-14
ZIMMER H 1971 Proc. 2nd. Symp. on Symbolic and Algebraic Manipulation 172-9

Correspondence

To the Editor
The Computer Journal

Sir

The letters by Flavell (1972) and Dewar (1972) commenting on my earlier note (Chambers, 1971) are encouraging signs of continued interest in FORTRAN definition, at least on the part of some users. However, a few of the points of detail raised need further comment, in order to avoid confusion.

Mr. Flavell's claim that array assignments as proposed would require dynamic temporary storage blocks would be true *if* one were to add new array operations, such as those in APL. Such a proposal would be objectionable, since it involves FORTRAN in **heap** storage of the ALGOL68 variety. However, using only the existing operators in an element-by-element manner requires storage only for the scalar elements of the left-side array involved (plus the usual requirements for the implied DO-loop); for example, for A(1, 1) in the example, A = A/A(1, 1). This extension I find fully compatible with FORTRAN philosophy.

Mr. Dewar is quite correct that copy-in and copy-out as practised by OS/360 is allowed by ANSI standard. The question is whether we can improve on the current standard here. My hope is that either call-by-address could become the standard or (more realistically, perhaps) that a new standard recognise the two possible methods explicitly and include a provision for forcing one or the other when necessary. This is a particularly pernicious ambiguity at present, as its consequences are subtle, undetectable at compilation, and occasionally disastrous.

It is to be hoped that further consideration of FORTRAN will lead to some practical revised standard for a language which seems to be occupying a growing portion of algorithm sections throughout the world.

Yours faithfully,
J. M. CHAMBERS

Bell Laboratories
600 Mountain Avenue
Murray Hill
New Jersey 07974
USA
25 April 1972

References

CHAMBERS, J. M. (1971). Another round of FORTRAN, *The Computer Journal*, Vol. 14, No. 3, pp. 312-314.
DEWAR, R. B. K. (1972). *Ibid.* Vol. 15, No. 1, p. 7.
FLAVELL, A. J. (1972). *Ibid*, Vol. 15, No. 1, p. 92.