An algebra system

D. Barton,* S. R. Bourne† and J. P. Fitch‡

* *Institute of Theoretical Astronomy, Cambridge*
† *Trinity College, Cambridge*
‡ *University Mathematical Laboratory, Cambridge*

This paper describes a computing system that enables problems of manipulative algebra involving a number of elementary functions to be simply and efficiently programmed. The system has been designed with particular reference to the problems involved in the explicit calculation of the Riemann tensor and associated quantities.
(Received March 1969)

In recent years considerable effort has been devoted to the development of general purpose list processing and symbol manipulation languages. Subsequently the languages have been employed to produce packages of programs to perform algebraic manipulation and it is not surprising that many of these systems have been designed with particular objectives in view. More recently the availability of interactive console systems has led to the development of formulae manipulation packages that compute with expressions of sufficient generality to be useful for the day-to-day calculations of mathematics and theoretical physics (see Clapp *et al.*, 1966, Engleman, 1965, and the last reference). However, it is doubtful if such systems could be successfully used on calculations that involve really large quantities of manipulative algebra. It is true that the field of problems giving rise to such calculations is at present small but it is probable that it will expand when the tools of the trade become more generally available.

Today the needs of people working on the above class of problems are met, in part, by the FORMAC System (Bond *et al.*, 1964) and the REDUCE System (Hearn, 1968). More specialised needs are met by the ALBERT program (Thorne *et al.*, 1967), a complete FORMAC program, and by the Clemens and Matzner System, 1967. The LISP list processing language has been used for manipulative algebra and both the ALAM System (d'Inverno, 1968) and GRAD-ASSISTANT (Fletcher, 1965) are LISP based and have been successfully used for varied work including the calculations of Riemann tensors. Very large problems in manipulative algebra have been solved by the ALPAC System (Brown, 1963) and the system of Barton, Bourne and Burgess (1968).

Work with the above systems has indicated that two problems will have to be solved if further real progress is to be possible. The first of these is to derive an algorithm that will, in some sense, simplify an arbitrarily complicated mathematical expression. This is a central problem of some difficulty that has so far been treated only by *ad hoc* techniques. A part of this problem, namely factorisation of an expression, has received more formal treatment in particular cases (see Brown *et al.*, 1963, Jordan *et al.*, 1966, and Collins, 1964). The simplified form of an expression is a subjective concept (see Fenichel, 1966) and is to a large degree dependent on the user and the nature of his problem. Of course the system designer may insist on a canonical representation within the computer, but he should be prepared to translate to some more convenient form for output should this be desired. The second fundamental problem that arises is that of designing facilities that would allow the user of a manipulative package the degree of control over the several steps of his calculation within the computer that he would have were he to compute by hand. This problem appears to have been mentioned only in Barton *et al.* (1968) but its importance is clear since inelegant algebraic calculation will prove more expensive by an order of magnitude in terms of machine time and storage consumption than inelegant numerical programming.

In this paper we describe a package of programs that have been written to perform algebraic manipulation on a fairly large class of elementary functions. The system consists of a simple language in which manipulative problems may be conveniently expressed, together with a package of subroutines that are used to perform the actual manipulation of expressions at run time. The system was designed with two aims in view, (1) to provide a useful system that would enable some of the calculations of General Relativity to be undertaken with greater convenience and (2) to provide a system of sufficient flexibility and power to enable further experimentation upon the two problems discussed earlier. The system is in use on the Titan computer (Prototype Atlas II) in Cambridge.

The subroutine package used by the algebra system

This package consists of a set of closed subroutines that perform elementary manipulative operations on a certain class σ of expressions. We define first the set S of polynomials in a finite number of variables over a given ground field. The set σ is then defined to be the

set of finite sums of products of functions f_i whose arguments are either themselves members of σ or alternatively members of S. The functions f_i are divided into two sets, the 'built-in' functions that are known to the system and on which certain types of simplification are carried out and the 'user' functions that are treated purely formally and subjected to no simplification beyond trivial cancellation. In order to simplify the initial programming problem two arbitrary restrictions have been imposed on the system. A function may have at most four arguments and at present there are only 16 'built-in' functions. The system is capable of extension should it be decided to relax either of these restrictions. At present the following are included as 'built-in' functions:

1. The identity and reciprocal functions.
2. The circular functions and the corresponding inverse functions.
3. The exponential, logarithmic and power functions.
4. Functions to denote formal differentiation and integration.
5. The functions Σ and Π.

Exponentiation is represented either by the appropriate combination of the functions **exp** and **log** or using the power function. The hyperbolic functions have been entirely omitted since their similarity to the circular functions makes their inclusion an unrewarding task.

The following functions are examples of the members of σ:

1. x (represented by the identity function on the polynomial x),
2. log $[x]$,
3. sin $[\log[x]+y]$,
4. cos $[\log[x+y]]+\sin[x]$,
5. *sigma* $[1/n.2, n]$ (meaning $\sum_{n=0}^{\infty} 1/n^2$, other limits may be used and sigma expressed as a function of four arguments).

Thus it will be seen that all elementary combinations of functions are available within σ.

The internal representation of expressions

An expression of σ is represented by the package as a branched chain of blocks of space. The whole of the available free space is initially chained together in units of 4 words and forms a free list for which no garbage collector is employed. In the representation of a member of σ one free unit is allocated to each function present. The first word of this unit contains a pointer to the following function together with a marker to indicate additive or multiplicative combination. The second and third words contain, respectively, the function name and the positive integral power to which it is raised, while the fourth word points to a unit of space that represents a vector of pointers to the arguments of the function. The structure is presented in **Fig. 1**. The vector of pointers to the arguments of a function also contains markers to indicate whether the argument is itself a member of σ with a similar data structure or whether it is a member of S. The members of S are represented by a separate data structure presented in **Fig. 2**. This structure takes the form of a branched list together with a header. The second, third and fourth words of the core unit reserved for the header contain respectively a pointer to the remainder of the data structure, a use count of outstanding references to the polynomial and a code word. We shall refer to these latter items later in this paper. The principle part of the data structure for the polynomial is a list of the units of four words containing, in the first two words, a pointer to the next additive polynomial term and a pointer to a unit of core containing the coefficient of the term, and in the latter two words the exponents of the several variables that compose the term. The data structure for a polynomial is presented in Fig. 2. The examples given in **Fig. 3** and **Fig. 4** should help to

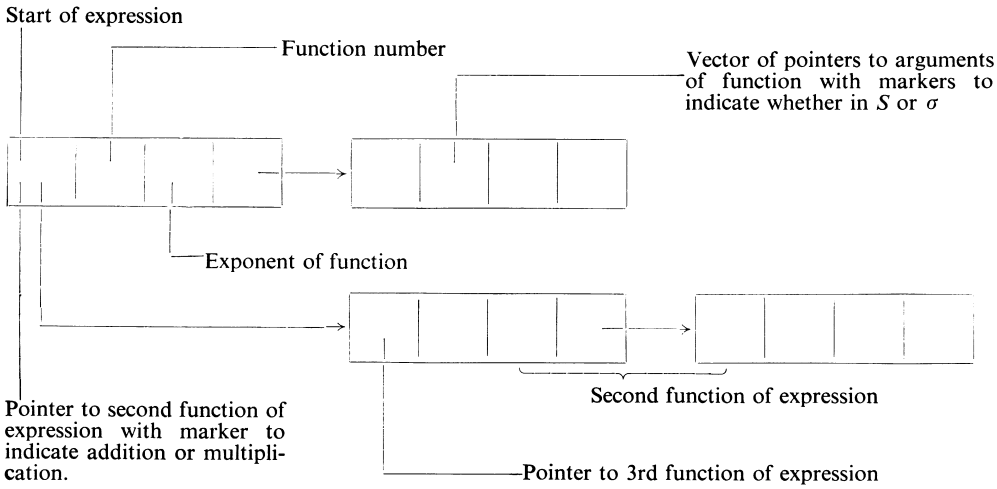

Fig. 1. Basic data structure of a member of σ

explain this complicated structure. It should be understood that the data structures described above are simplified versions of those that are used by the system but that they are similar to them in all essential respects.

In order to try to make efficient use of the machine store we have arranged to store algebraic expressions in a canonical form within the computer and we have associated a 'use count' with each stored expression, Collins (1966); this number is a count of the number of outstanding references to that expression. Associated with each member of S stored in the computer is a codeword obtained by combining the internal representation of the several parts of the expression according to some simple rule. While two identical expressions will have the same codeword the converse is not true. However, by suitable adjustment of the rule defining a codeword the probability of two different expressions having the same codeword can be made small and hence the labour of discovering if a newly obtained expression is already present in the machine may be substantially reduced. It thus becomes possible to scan the set of all stored expressions whenever a new expression is produced in order to increment a use count if this should prove possible.

An expression is in canonical form when the various elements in the sums and products of the expression are arranged in a well defined order. All routines compute with arguments held in canonical form and arrange to output their results in this form. No explicit program is required for transformation to canonical form but a program is included that will decide if two members of σ are 'equal' in the sense of representational equivalence within the computer. In the canonical form the expressions of σ are reduced to the maximum extent by elementary algebraic cancellation. Expressions are further reduced by the application of various combinatorial simplification rules upon the 'built in' functions. The system does not pretend to provide an adequate simplification algorithm for the 'built in' functions included. However, combinations of the functions are reduced in the following ways automatically after any arithmetic operation, before printing and otherwise at the convenience of the system.

(*a*) Under addition: identity, arcsin, arccos, log, formal differentiation, formal integration, and Σ have their arguments combined. e.g. $\log(x + y) + \log(a + b) = \log(ax + ay + bx + by)$.

(*b*) Under multiplication: identity, sine, cosine, exponential and Π have their arguments combined; e.g. $\sin(x)\cos(y) = (\sin(x + y) + \sin(x - y))/2$.

(*c*) Wherever possible such reductions as
$$\sin(\arcsin(x)) = x$$
are performed.

(*d*) Wherever possible the reduction $\exp(n \log x) = x^n$ is applied for integral n.

The algorithm performs all possible reductions of the above types and, if the result is found to occupy less store, it is regarded as a simpler form. Otherwise the original form is retained. It is clear that this algorithm is inadequate since the rules can only be applied in one direction and all at once.

While the list of 'built-in' functions can, in principle, be arbitrarily extended and new elementary simplification rules included, the system has not been designed so that the user can extend the simplification algorithm. It is possible, however, for the user to extend the list of functions with his own 'user' functions but the system will perform only elementary cancellation on these. The reason for this restriction is essentially one of economics

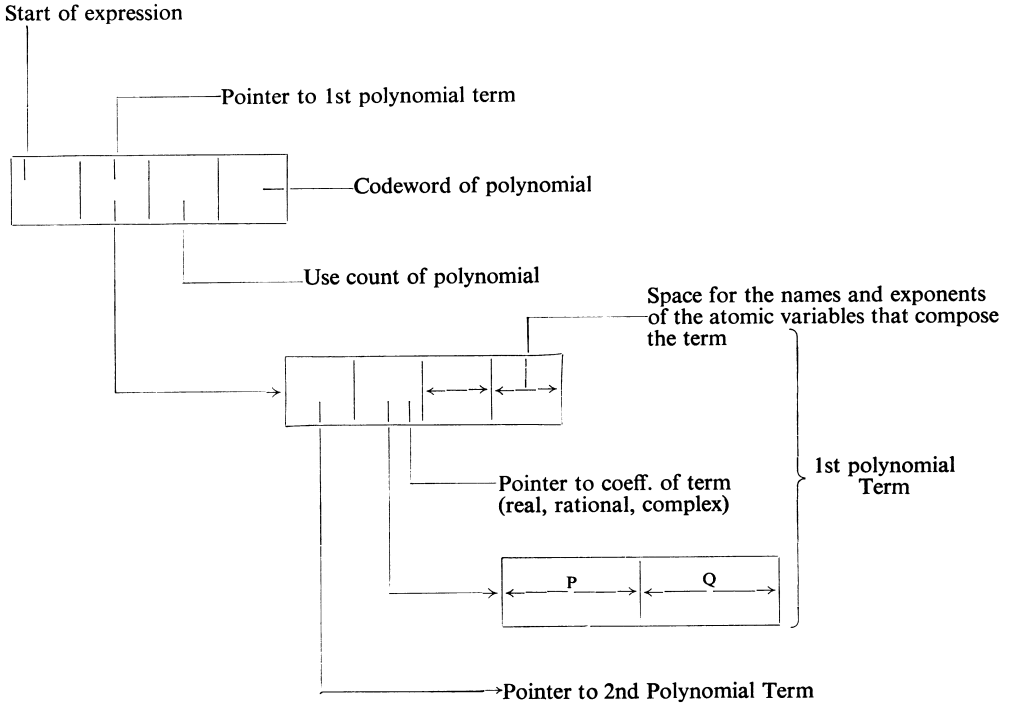

Fig. 2. Basic data structure of a member of S

and it is our intention to discover the possible advantages and disadvantages of the technique.

With the simplification rules described above it is only possible to discover if two expressions are similarly represented in the core and this is a much stronger condition of equality than algebraic identity. This implies that the system may well contain a complicated and extensive representation of the zero expression and calculations with this lead to the proliferation of unnecessary list structures in the machine. While the trivial occurrences of zeros are removed the system will not recognise more complex cases, e.g.

$$\log \tan \left(\frac{x}{2} + \frac{\pi}{4}\right) - \sinh^{-1} \tan x \equiv 0.$$

This problem will only be overcome by the provision of an adequate simplification algorithm. The mechanism of 'use counts' is intended to try to relieve the difficulty but the problem remains and is, of course, the major weakness of the system.

For the purpose of the trivial simplification that is all we discuss here any definition of order between members of σ will be sufficient. However, it is probable that if complex simplification procedures were to be used the particular definition of order employed would have a considerable effect upon the efficiency of the simplification algorithm.

The manipulative subroutines provided in the package

The arithmetic operations between expressions that have been programmed have been written so that they may be entered recursively and hence it is only necessary for each routine to deal with the present outer level of sums and products of functions. Arithmetic with the arguments of functions is only stimulated by the application of simplification rules and the appropriate operations are performed by recursive entry to the arithmetic routines. Hence the operations of addition and multiplication reduce to elementary merging of ordered lists with cancellation and term by term multiplication followed by the merging operation.

The various manipulative routines have been written with a particular regard to the efficient use of core store.

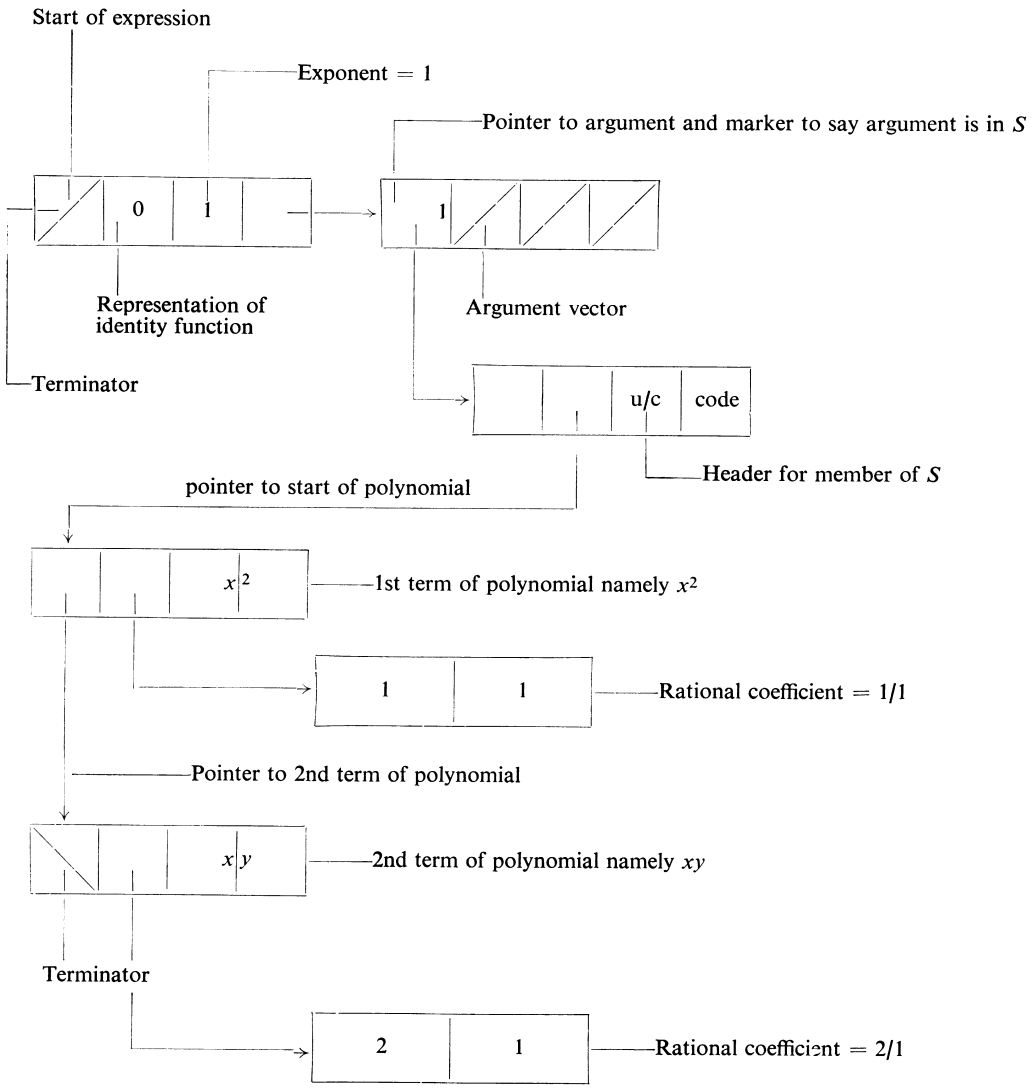

Fig. 3. The representation of the expression $x^2 + 2xy$

Consequently the routines arrange to produce their results in the space occupied by their operands wherever this is possible. The mechanism of 'use counts' prevents this occurring in many cases and a program such as addition must check the use counts of the outer level of its operands and produce a copy of that level before performing the merge if either count is greater than one. The production of such a first level copy raises the use counts of the various arguments of the functions occurring at that level.

It will be clear then that even the elementary operation of addition cannot guarantee to make no demands on the available free space and selection facilities have, therefore, been incorporated that operate at the time expressions are generated by the arithmetic routines. These facilities allow the user to call for that part of the result of an arithmetic operation that satisfies certain prescribed conditions and in this case no attempt is made to store the complete result. Instead each part of the result is inspected when it is produced and, if it is not required, is discarded.

The manipulative routines that have so far been programmed are as follows:

1. $\alpha = -\alpha$ (negate α).
2. $\alpha = \partial\alpha/\partial x_i$ (differentiate α w.r.t. x_i).
3. $\alpha = \int \alpha dx_i$ (integrate α w.r.t. x_i).
4. $\alpha = \alpha + \beta$ (add α to β).
5. $\alpha = \alpha - \beta$ (subtract β from α).
6. $\alpha = \alpha\beta$ (multiply α and β).
7. $\alpha = \alpha/\beta$ (divide α by β).
8. $\alpha = \alpha(x_0, \ldots, \beta, \ldots, x_n)$ (this routine causes expression β to be substituted for a variable x_i into expression α).

where α and $\beta \epsilon \sigma$ and x_i is an atomic variable of S. It should be noted that the operations of differentiation and integration listed above are explicitly carried out by the system and are quite distinct from the functions for formal differentiation and integration mentioned earlier that are treated as built in functions with their own simplification rules. The explicit integration program is very elementary and proceeds by reference to a table of standard forms and then by integration by parts where this is possible. Whenever the user performs an explicit differentiation the table of standard forms is suitably updated. If the program is still unable to integrate the expression and if the user program is interactive, the system displays the quantity it cannot integrate on the user's console, or if possible on a display screen, and asks the user for help. If the job is not interactive the formal integration function is used.

The programming language for manipulative algebra

A primitive language based on Titan Autocode (Barron *et al.*, 1967) has been designed to enable the manipulative subroutines to be conveniently used. The language, known as B-Code for historic reasons, is compiled into semi interpretative machine code by a load and go single pass compiler. We shall not present here a complete syntactic description of B-Code as the exact structure of the language is not relevant to this discussion, but rather we shall give a few brief details and then demonstrate the system by example.

A B-Code program is composed of the upper and lower case letters together with the digits and various special symbols. The letters A–Z denote the names of the two possible data types and these are:

1. *Indices.* Fixed point signed integers in the range $|n| < 10^6$ called by the names, I, J, . . . , T. Arrays of these indices are also available and are referenced by I[*m*, *n*, . . .], . . . , T[*m*, *n*, . . .].
2. *Expressions.* The members of σ called by the names A-H and U-Z. Arrays of these expressions are also allowed, called by the names A[*m*, *n*, . . .], . . . , H[*m*, *n*, . . .], U[*m*, *n*, . . .], . . . , Z[*m*, *n*, . . .].

An expression of σ is made up from a combination of lower case letters, e.g. alog[*x*] or xsin[*x*] square brackets being used to enclose the arguments of functions. A number of special symbols have been used whose meaning is not immediately obvious and these are:

1.	|		vertical bar is used to introduce comment.
2.	§	(expression) dx	indicates the explicit integral with respect to *x*. (The integration is carried out by the system.)
3.	d	(expression)/dx	indicates the explicit derivative with respect to *x* (the differentiation is carried out by the system).
4.	D	(expression)/Dx	indicates the formal derivative with respect to *x* (the built in function is employed).
5.	§	(expression) Dx	indicates the formal integral with respect to *x* (the built in function is employed).
6.		(expression) . N	indicates the *N*th power of the expression.
7.		pi	represents the symbol π.
8.		i	represents $\sqrt{-1}$.

The B-Code language is based upon Titan Autocode and this language has no formal multiplication operator, consequently the syntax for B-Code is ambiguous in some respects. We have found it more convenient to write our programs to avoid these difficulties rather than include an explicit multiplication operator.

Program control is provided by means of 'FOR REPEAT' loops, that are similar to the FORTRAN 'DO' loops, together with conditional and unconditional jumps to explicit labels. Conditional jumps on the value of an algebraic expression are, of course, only meaningful if the condition is equality and in this case the system is only able to detect representational equivalence. A closed subroutine facility is provided. Input and output are controlled by the INPUT and PRINT statements and comprehensive error detection and control facilities are provided.

We present now three simple examples in the form of complete programs. The first to calculate

$$\int \frac{\partial}{\partial x} \{x^3 \sin (x^2 + xy)\}\, dy.$$

```
B-CODE PROGRAM
1 : A = x.3 sin [x.2 +xy]      | read expression in-
                               |   to A
    B = § (dA/dx)dy            | set B to required
                               |   result
    PRINT (B)                  | print the result
    STOP                       | stop execution
START 1                        | directive to start
                               |   program
```

We may also solve the above problem by the program

```
B-CODE PROGRAM
1 : PRINT (§ (d(x.3 sin [x.2 + xy])/dx)dy)
STOP; START 1
```

A less trivial example of the use of the system is to calculate a Fourier series expansion of the function x. We give below a program to perform this task.

```
B-CODE PROGRAM
1 : A = x                      | set x into A
    B = § (A sin [nx])dx       | calculate an indefi-
                               |   nite
                               | integral
    B = Substitute (B, pi, x)—Substitute (B, —pi, x)
          | put in the limits of integration ± π
    B = B/2pi                  | set B to nth coeff.
                               |   in series
    PRINT (sigma [B sin [nx], n]) | print result
    STOP; START 1
```

The above program produces the result

sigma [2(—1).(n + 1) sin [nx]/n]

The principal purpose in the design of the B-code system was to enable some of the calculations associated with the General Theory of Relativity to be performed with greater convenience. We shall therefore take a final example of the use of our system from this subject. We require to construct and print the Christoffel symbols of the first kind given the components of the metric. For simplicity of presentation we take the static spherically symmetric metric

$$ds^2 = g_{ij}dx^i dx^j = -e^{p[r]}dr^2 - r^2du^2 - r^2\sin^2 udv^2 + e^{q[r]}dt^2$$

The Christoffel symbols are given by

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

The program of **Fig. 6** calculates and prints the distinct non zero Christoffel symbols in a self-evident manner. The results are reproduced in **Fig. 5**.

```
[11,1] = — 1/2p'[a]exp[p[a]]
[12,2] = — a
[13,3] = — a(sin[b]).2
[14,4] = 1/2q'[a]exp[q[a]]
[22,1] = a
[23,3] = — a.2sin[b]cos[b]
[33,1] = a(sin[b]).2
[33,2] = a.2sin[b]cos[b]
[44,1] = — 1/2q'[a]exp[q[a]]
```

Fig. 5. Results of the program to calculate Christoffel symbols

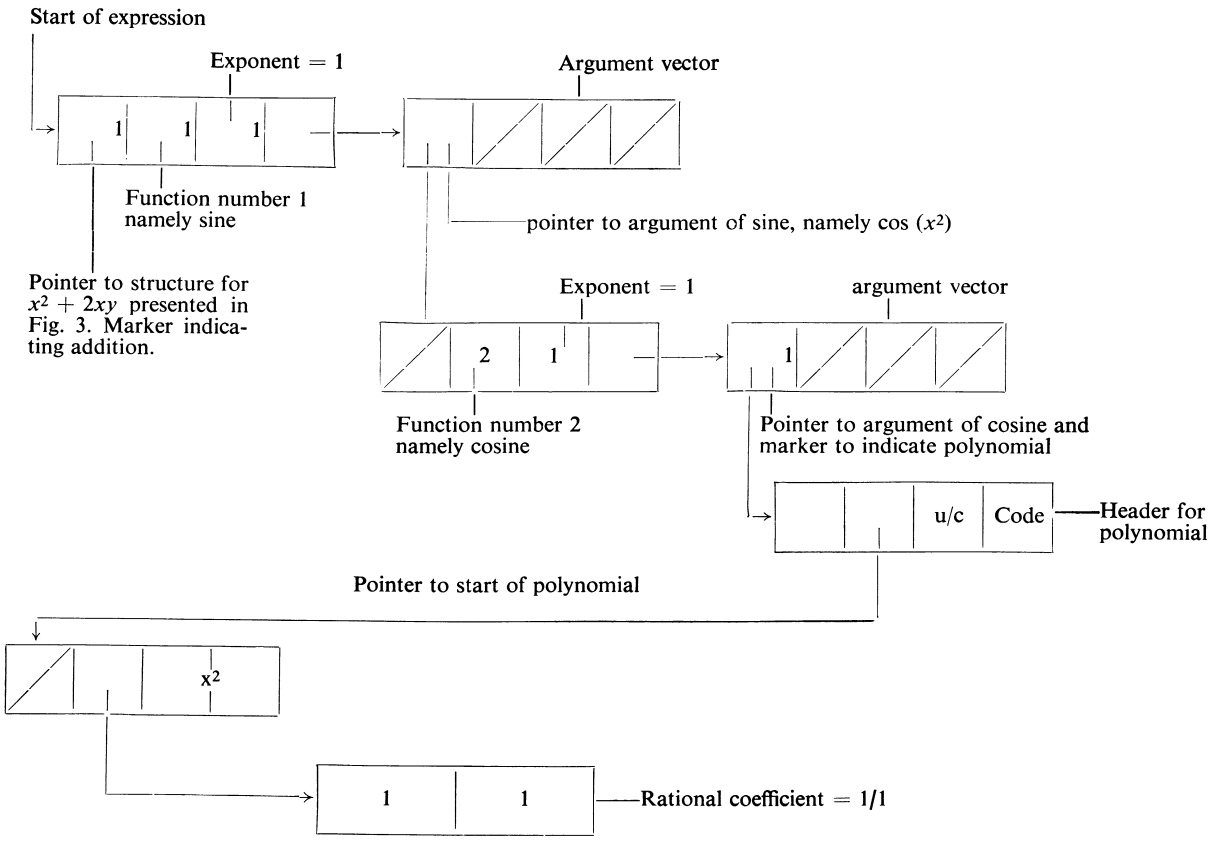

Fig. 4. The representation of the expression *sin* ($cos(x^2)$) + x^2 + 2xy

Possible additional facilities

The algebra system contains two facilities that we feel should be remarked upon. We have previously indicated that the user may introduce new functions into the system and that these are subsequently treated as elementary functions but no simplification is attempted. To do this the user simply writes the new function into his program and calls it by any lower case letter, e.g. A = u[x] will set the variable A to the previously undefined function u[x]. Subsequently the function will be treated exactly as a 'built-in' function except that no combinatorial simplification rules will be applied.

The second aspect of the system that requires comment is concerned with the selection facilities available to the user during a calculation. These facilities may be conveniently divided into two groups; those concerned with the size of an expression for various values of its arguments and those concerned with the exact functional structure of an expression. In the first of these classes we assume that all the elementary variables a, . . . , z are of the same order of smallness for approximation purposes. However, their relative size may be redefined at runtime by the statement a = 0 (expression) which sets the size of a to be that of the largest term in the expression. Thus a = 0 (x.2) sets a to be of order x^2. There is built into the system the knowledge of the behaviour of the 'built in' functions for values of the argument near the origin and infinity, and thus the system is able to determine the relative size of the terms of an expression. The behaviour of the users' functions may be defined by a statement of the form u[x] = 0 (x.2) AS $x \to 0$. Facilities are then available for choosing from a given expression, or dynamically from the result of a calculation, the smallest or largest terms and also to round an expression down to a given order in small quantities.

In order to provide selection facilities dependent upon the functional structure of an expression a special variable named TERMS is provided. This variable is an expression of σ and is subject to computation in a similar fashion to the variables A-H, U-Z. However, if we wish to select from the product AB those terms that contain log [x] as a factor one would write TERMS = log [x]; C = AB. A symbol is provided to indicate independence of argument, and to select all logarithm terms from the product AB one would write TERMS = log [≠]; C = AB. In the expression TERMS addition is interpreted to mean logical *or* while multiplication means logical *and*, hence to select the terms in x log [x] or y log [y] from the product AB one would write TERMS = x log [x] + x log [y]; C = AB.

The part of the system that is concerned with selection facilities is undergoing extensive development and we make no claim to have treated this subject exhaustively. Some of the problems that naturally arise are of extreme difficulty, such as the existence of certain limits and the convergence of series. Others are of less difficulty but are still of considerable importance such as the linguistic description of the user's requirements. However, this is a branch of automatic algebraic manipulation that has been largely neglected although the techniques are frequently employed by the mathematician with pencil and paper.

Finally, it is instructive to see a comparison of the runtime storage requirement and computation times of our system with some others in the field. The test calculation undertaken was that described in d'Inverno (1968) of the construction of a number of quantities necessary in the study of General Relativity. Using the Bondi, Van der Burg, Metzner metric (Bondi *et al.*, 1962) the following were calculated: Christoffel symbols, Curvative tensor, Ricci tensor, Ricci scalar and the Einstein tensor. The comparison is given in **Table 1.**

```
B—CODE PROGRAM G[4,4] D[4,4,4] |CALCULATE THE CHRISTOFFEL SYMBOLS
       |The above line includes a declaration of subscripted variables used to
       |store the metric tensor and its various differential coefficients.

1:     |The start of the program
       |First a routine to set up our particular metric.

       FOR I = 1 : 1 : 4 ; FOR J = 1 : 1 : 4 ; G[I,J] = 0 ; REPEAT ; REPEAT
       G[1,1] = — exp[p[a]] ; G[2,2] = — a.2
       G[3,3] = — (a sin[b]).2 ; G[4,4] = exp[q[a]]

       |Next we calculate the derivatives of the elements of the metric.

       FOR I = 1 : 1 : 4 ; FOR J = 1 : 1 : 4
       D[I,J,1] = dG[I,J]/da ; D[I,J,2] = dG[I,J]/db
       D[I,J,3] = dG[I,J]/dc ; D[I,J,4] = dG[I,J]/de
       REPEAT ; REPEAT

       |The following routine calculates the Christoffel symbols taking
       |symmetry into account.

       FOR I = 1 : 1 : 4
              FOR J = I : 1 : 4          |[ij,k] = [ji,k]
              |[ij,k] = 0 if i ≠ j ≠ k
              → 4 IF I = J ; K = I ; →2→ ; K = J ; →2 → ; 2 → 3
4:            FOR K = 1 : 1 : 4 ; →2→ ; REPEAT
```

Fig. 6

```
3:          REPEAT
        REPEAT
        STOP
2:      |Subroutine to evaluate and print the symbol [IJ,K]
        |Set A = [IJ,K]
        A = (D[I,K,J] + D[J,K,I] − D[I,J,K])/2
        → 5 IF A = 0        |Do not print a zero symbol.
        |Put out text '[ij,k] = ' followed by the value of the symbol.
        TEXT:[: ; PRINT(I) ; PRINT(J) ; TEXT:,: ; PRINT(K) ; TEXT:] = :
        PRINT(A)
5:      RETURN    |End of subroutine.
        START 1
```

Fig. 6 continued

Table 1

SYSTEM	MACHINE	STORE REQUIRED	TIME REQUIRED
Barton-Bourne-Fitch	Titan*	18 K	4 mins.
ALAM (8)	Atlas I	50 K	4 mins.
Clemens-Matzner (7)	IBM 7094	Not available	30 mins.
GRAD.-ASSISTANT (9)	IBM 7090	Not available	17 mins.

* Titan is a prototype Atlas II.

Acknowledgements

In conclusion we would like to thank Professor M. V. Wilkes for his valuable comments on an earlier draft of this paper and Mr. J. R. Horton for his assistance with the display facilities and for his programming effort during the construction of the system. Further, we should like to thank the Director and staff of the University Mathematical Laboratory for their assistance and encouragement throughout the project.

References

BARRON, D., BROWN, H., HARTLEY, D., and SWINNERTON-DYER, H. P. F. (1967). *Titan Autocode Programming Manual*, University Mathematical Laboratory, Cambridge.

BARTON, D., BOURNE, S. R., and BURGESS, C. (1968). A simple algebra system, *Comp. J.*, Vol. 11, No. 3.

BOND, E., AUSLANDER, M., GRISOFF, S., KENNEY, R., MYSZEWSKI, M., SAMMET, J., TOBEY, R., and ZILLES, S. (1964). Formac—An Experimental Formula Manipulation Compiler, *Proc. A.C.M.*, 19, Nat. Conf.

BONDI, H., VAN DER BURG, M. G. J., and METZNER, A. W. K. *Proc. Roy. Soc. A*, Vol. 269, p. 21.

BROWN, W. S. (1963). The Alpac system for non-numerical algebra on a digital computer I, *Bell Tech.* 42, pp. 2081–2119.

BROWN, W. S., HYDE, J. P., and TAGUE, B. A. (1963). Alpac II, *Bell Tech.* 43, 785.

CLAPP, L., JORDAN, D., WAX, E., and WOLF, R. (1966). Magic Paper—An on line system for the manipulation of symbolic mathematics, Computer Research Corp. DDC–A9–643313.

CLEMENS, R., and MATZNER, R. (1967). A system for symbolic computation of the Riemann Tensor, Tech. Report N.635, Univ. of Maryland.

COLLINS, G. E. (1964). Polynomial remainder sequences and determinants, *Notices of Am. Math. Soc.*, Vol. 13, No. 2.

COLLINS, G. E. (1966). PM—A system for polynomial manipulation, *CACM*, Vol. 9, No. 8.

D'INVERNO, R. A. (1968). Alam—Atlas Lisp Algebraic Manipulation, *Comp. J.*, Vol. 12, No. 2, pp. 124–127.

ENGLEMAN, C. (1965). Mathlab—A program for on line machine assistance in symbolic computations, *Proc. A.F.I.P.S.* Fall *J.C.C.*

FENICHEL, R. R. An essay on simplification, *SICSAM Bulletin* No. 6.

FLETCHER, J. G. (1965). GRAD-ASSISTANT—A program for symbolic algebraic manipulation and differentiation, U.C.R.L. Report 14524–T.

HEARN, A. C. (1968). Reduce Users Manual, Stanford Artificial Intelligence Project—Memo. 50.

JORDAN, D. E., KAIN, R., and CLAPP, L. C. (1966). Symbolic factoring of polynomials in several variables, *CACM*, Vol. 9, No. 8.

THORNE, K. S., and ZIMMERMANN, B. (1967). ALBERT—A package of four computer programs for calculating General Relativistic Curvature Tensors and equations of motion, Joint Tech. Report, Cal. Inst. Tech.

A system for interactive Algebraic manipulation (1968). Applied Data Research Inc.