CLAM—Its function, structure and implementation

R. A. d'Inverno* and R. A. Russell-Clark†

CLAM, a successor to ALAM, is a LISP-based system for carrying out algebraic manipulation. It was designed especially for use in the field of general relativity where it lays claim to being the fastest such system.

(Received June 1973)

1. Introduction

CLAM (CDC LISP Algebraic Manipulator) is a LISP-based algebra system and represents the final form in the development of ALAM (*see* d'Inverno, 1969). Both ALAM and CLAM were designed especially to aid research work within the field of general relativity and, to date, they have been used successfully on numerous problems, details of which can be found in d'Inverno (1970), d'Inverno and Russell-Clark (1971) and Russell-Clark (1973).

In the past, the most common uses of computer algebra systems within relativity have been what we shall term 'metric applications'. These are calculations which take ten algebraic expressions as their initial data (specifically, they are the ten independent components of a covariant metric tensor, g_{ab}) and, from these, by the processes of differentiation and algebraic manipulation, evaluate a number of other important quantities of geometrical significance. CLAM has been designed primarily with this type of application in mind.

A CLAM program consists simply of a series of commands. Each command is made up of a CLAM function name followed by a list of arguments (in fact, the simplest form of LISP doublet). The principal function is METRIC which takes as its argument a list of the ten algebraic expressions comprising g_{ab} and its action is to compute and print out all the independent components of the quantities

$$\{g_{ab}, g, g^{ab}, \Gamma^a_{bc}, R_{abcd}, R_{ab}, R, G_{ab}\}$$

which are various combinations of the metric tensor and its derivatives up to the second order. There are also functions for defining variables, functional dependence and substitutions; functions for halting the action of METRIC at various points in its process and for calculating further required expressions. A description of the types of commands available to the programmer is given in succeeding sections of this paper. The resulting system is flexible and extremely easy to use for metric applications and the command language has been found to be quite adequate for such purposes.

A two-part manual (d'Inverno and Russell-Clark, 1971; 1972) has been written for the CLAM system. Part 1 of the manual describes a minimal subset of the command language sufficient for the reader with no previous programming experience to write programs to calculate the curvature tensor and some related quantities of a restricted, but large, class of metrics. It has been found that this is possible after only about half an hour's study. The attainment of this goal has been one of our principal aims from the outset. Part 2 explains how to deal with common factors and substitution in general and describes some additional facilities which have been found to be most commonly needed in practice.

The following sections of this paper are concerned with various aspects of the CLAM system. We describe the simplification processes in CLAM and how substitutions and common factors are dealt with. In particular, Section 10 gives a brief account of METRIC and the associated functions which affect its action. In the final section we attempt a critical analysis of the system. One of the appendices contains a sample program and its output.

2. Algebraic expressions and simplification

The format in which algebraic expressions are input is the same as in ALAM. However, the definition of an algebraic expression given in d'Inverno (1969) is not complete and a corrected version is presented in Appendix 1.

Our attitude towards simplification has been dictated by the needs of relativity on the one hand and on the other by the need to economise on store as much as possible whilst using the LISP list structure. As a result, no attempt is made to extract common factors or perform division. The simplification process is at its most efficient when the calculations involve only constants, variables, arbitrary functions and the trigonometric, exponential and logarithmic functions. However, when common factors occur which have to be differentiated and/or simplified or when substitutions are required, then extra simplification functions are called into use and more time is consumed in their execution.

The simplification function is SIMP which takes as its argument an algebraic expression and returns as its value the simplified form of that expression. When no substitutions are present, SIMP consists solely of a call to each of the three functions ZERM, EXPD and EDIT in turn. The action of these functions has been described in detail in d'Inverno (1969) but the following summary will be sufficient for the purposes of this paper. ZERM rids an algebraic expression of zeros after which EXPD expands the expression out using the laws of distributivity and associativity. EDIT then passes through the resulting expression simplifying each term and collecting like terms together.

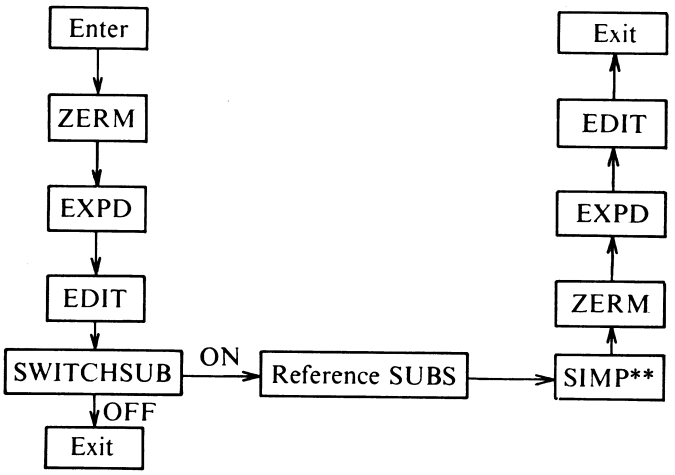
3. Substitutions

Substitutions are defined by the programmer by using the function SUBSTITUTE whose first argument is the expression to be substituted for and its second is the expression to be substituted. Both arguments are formed into a dotted pair and this is added onto the end of the substitution list which is held as the APVAL of the atom SUBS.

Although substitutions may have been defined in this way, they will not be used by SIMP unless the function SWITCHSUB has been used. The command SWITCHSUB (ON) will patch SIMP in such a way that, on all subsequent calls, a branch is made after the ZERM-EXPD-EDIT sequence to another set of instructions which references SUBS, calls the extra simplification function SIMP** (see Section 5) and finally passes through the ZERM-EXPD-EDIT sequence once more. The substitution mechanism is straightforward. The

*Department of Mathematics, University of Southampton, Southampton SO9 5NH.
†The Computer Laboratory, University of Cambridge, Corn Exchange Street, Cambridge CB2 3QG.

first dotted pair on the substitution list ($(\alpha_1.\beta_1)$ say; α_1, β_1 are algebraic expressions) is accessed and the algebraic expression is searched for every occurrence of α_1 using EQUAL (both α_1 and β_1 will have been canonicalised on input, see Section 6). Every time one is found, a copy of β_1 replaces it by direct modification of the list structure. The next pair on the substitution list is then accessed and the algebraic expression is searched again and so on until the end of the list. The full action of SIMP can be displayed in the form of a block diagram:



SIMP can always be patched back into its original form by SWITCHSUB (OFF). A set of substitutions already defined can be lost by the command LOSE (SUBS) which merely sets the APVAL of SUBS to NIL; whereupon, a new set can then be defined if so wished.

4. Common factor expressions

If the algebraic expression $(a + b)(a + b)^{-1}$ were to be given to SIMP, it would be expanded out to

$$a(a + b)^{-1} + b(a + b)^{-1}$$

by EXPD and the subsequent call to EDIT would leave it in this form and not reduce it to unity. However, if the factor $(a + b)$ were to be represented by the atomic symbol A, then the example expression would now be AA^{-1} and SIMP will simplify this to unity. It is in precisely this way that CLAM deals with common factors.

Common factors are defined by the function FACTOR which takes two arguments; the first is the chosen atomic symbol which is to represent the factor expression and the second is the expression itself. FACTOR proceeds by calculating all derivatives up to the second order (METRIC only requires derivatives up to this order) of both arguments, at the same time forming a list of dotted pairs of the corresponding derivatives. This list, which includes the original argument pair as well, is then added onto the end of the factor list which is attached as APVAL to the atom FACTORS. For example, if the atom A is to represent the factor expression α which is a function (either implicitly or explicitly) of the variables x^0 and x^1 then the command FACTOR (A α) will produce the list:

$$(((\$\ A\ 1\ 1).\alpha_{11})\ ((\$\ A\ 1).\alpha_1)\ ((\$\ A\ 0\ 1).\alpha_{01})\ ((\$\ A\ 0\ 0).\alpha_{00})\ ((\$\ A\ 0).\alpha_0)\ (A.\alpha))$$

which is then NCONC'ed onto the APVAL of FACTORS.

Within algebraic expressions, common factors defined by FACTOR can therefore be represented by their corresponding atoms and SIMP treats these just like arbitrary functions. Thus, valuable store is saved and required cancellations will take place. Unlike substitutions, the actual values of the factor expressions and their derivatives are only substituted on output and the process forms part of the function PRT (see Section 9). The command SWITCHFAC (ON) patches PRT so that the factor list is referenced in a manner similar to that in which the substitution list is referenced in SIMP and the values of the common factors and their derivatives are substituted into the argument of PRT. This intermediate result is then given to SIMP** and SIMP before a branch is made back to the main body of PRT which prepares the final simplified expression for output. PRT can always be patched back into its normal state by SWITCHFAC (OFF).

5. Extra simplification functions

SIMP in its normal state makes no attempt to simplify mantissae of exponential expressions or multiply out expressions raised to integral powers. However, if such simplifications are required (for instance, after common factor substitutions and/or ordinary substitutions have taken place) then either SIMP or PRT (or both) can be made to reference the extra simplification function SIMP** by using SWITCHSUB or SWITCHFAC respectively. SIMP** works by searching an algebraic expression for every subexpression headed by the operator **, whereupon the subexpression is handed to EDIT* which performs the quick, elementary simplifications. If necessary, the value of EDIT* is given to EDIT** which performs the more complicated simplifications associated with exponentiation.

Two functions, POWER and EXPAND, affect the action of EDIT**. POWER patches EDIT** so that numbers raised to an integer power whose modulus is $\leq k_N$ (the argument of POWER) are multiplied out. EXPAND similarly patches EDIT** so that sums of terms raised to an integer power of modulus $\leq k_S$ (the argument of EXPAND) are likewise expanded out. The default setting of k_N is 3 and that of k_S is 1 (i.e. no expansion). For example, with the default settings, the expressions

$$(\tfrac{2}{3})^4,\ (a + b)^2,\ (a + b)^{-2}$$

would be unaffected. However, if the commands POWER (4) and EXPAND (2) are interpreted, the above expressions would subsequently be simplified by EDIT** to

$$16/81,\ a^2 + 2ab + b^2,\ (a^2 + 2ab + b^2)^{-1}$$ respectively .

No attempt is made in CLAM to combine or expand out expressions involving sine, cosh, log, etc. The arguments of these functions are never simplified and thus it is left to the programmer to decide in what form expressions involving these functions should be input. A consequence of this is that a substitution making the argument of cosine zero, say, will not result in a simplification to unity; that is, the substitution defined by SUBSTITUTE (X3 0) will only reduce (COS X3) to (COS 0). However, SUBSTITUTE ((COS X3) (1 1))† will produce the required simplification. As mentioned in d'Inverno (1969), it is this kind of restricted capability which has enabled ALAM/CLAM to be reasonably efficient in the calculations it has been called upon to do.

6. Canonicalisation

All algebraic expressions read into the system through the standard input functions (these are METRIC, METRIC2, SUBSTITUTE, FACTOR and TETRAD; see Section 10) are canonicalised. SIMP1 is the canonicalisation function and it works by recursively calling itself throughout an expression using EXPD and EDIT where necessary to multiply out subexpressions. SIMP1 canonicalises all arguments of trigonometric and logarithmic functions and all mantissae and exponents. All expressions within a term are put into canonical order so that 'like terms' can be tested for equality by using EQUAL. On the other hand, the order of the terms within a

†Rational numbers in CLAM are represented by a list of two positive integers. The first integer is the numerator of the fraction and the second the denominator. Thus, in this case, (1 1) represents the number 1.

sum is not canonicalised unless the sum is the argument of a function or is a mantissae or exponent.

Within calculations, further canonical ordering only takes place when new mantissae or exponents are formed. The actual ordering itself is not important and in practice a natural one has been chosen so as to make the output the more readable; for instance

$$(1/3)UR^{-1}E^{2B-2G}B_1 \text{ instead of } B_1R^{-1}(1/3)E^{-2G+2B}U\ .$$

7. Syntax analysis

Upon entry through any of the standard input functions (see Section 6) an algebraic expression is given first of all to SYNTAX which uses the definition in Appendix 1 to check whether the expression is a legal one. Any illegal expression is printed out and is followed by a message informing the programmer of the fact. SYNTAX then aborts the job.

Some CLAM functions check their arguments to determine whether they are of the correct type. Other errors, such as misspelling and incorrect bracket pairing, are trapped by the LISP input routines.

8. Differentiation

Variables and arbitrary functions are represented by atoms which have the properties VAR and DEP respectively on their property lists. For a variable the property itself is a member of the set {0, 1, 2, 3} denoting whether it is the x^0, x^1, x^2 or x^3 variable. For an arbitrary function the property is a list of numbers denoting the variables on which the function depends. These properties are defined to the system by using two functions, VARIABLES and FUNCTION. For instance, VARIABLES (U R THETA PHI) declares U to be the x^0 variable, R to be the x^1 variable and so on; and FUNCTION (A (U THETA)) will define A to be a function of U and THETA, i.e. give A the DEP of (0 2).

The function which performs differentiation is DIFF; its first argument is the algebraic expression to be differentiated and its second is an integer denoting the variable with respect to which it is to be differentiated. If the first argument is atomic DIFF searches its property list for a VAR or DEP property and acts accordingly. If neither is found the atom is assumed to represent a constant.

If the argument is neither an atom nor a CLAM number, then DIFF recovers the SUBR property of the top-level operator and hands control over to that. By giving the set of operators SUBR's, each of which recursively calls DIFF, all the rules for differentiating products, exponentials, etc. are implemented.

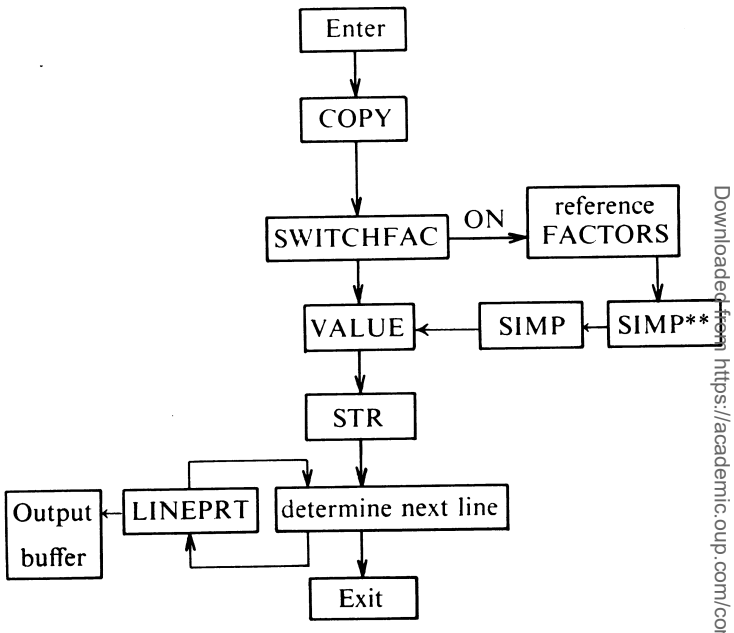
9. Output

In CLAM, two files are used for output purposes. Interpreter, garbage collection and error messages are written to the standard file OUTPUT. However, to keep the results of calculations separate, these are written to the scratch file METRIC which can either be copied to OUTPUT at the end of the job or can have its disposition changed to that of the line-printer.

The function which prints out algebraic expressions in a three line mathematical format is PRT (to our knowledge, the first system to do so).

Upon entry its argument is copied and is then given to VALU which changes from prefix to infix notation delineating superscripts and subscripts by the special atoms // and /*. Every *new* atom which enters the CLAM system is automatically given the property NAM which is a list of the character object atoms which make up its name. VALU references this property when processing atoms so that the PNAME does not have to be decomposed every time the atom is printed out. However, because numbers are not held uniquely in LISP, these have to be decomposed into a list of the corresponding numeric character objects.

After VALU, the function STR flattens out the list structure of its argument so that it becomes simply a list of character objects (apart from // and /*) all at the top level. PRT breaks the resultant list after the last term that can fit on a line, sets a pointer to the remaining part and hands the first part to LINEPRT which converts the character objects into DPC codes (this is trivial as all character objects in the CDC LISP system are held in consecutive locations in store in the correct DPC order) and finally assembles the line, in three line format, in the output buffer. PRT then picks up the remaining list and deals with this in the same way. The full action of PRT can now be displayed in block diagram form (see also Section 4).



10. METRIC and associated functions

Every scalar or tensor quantity which is referred to by the CLAM system has associated with it a unique atomic name. The metric tensor g_{ab}, for instance, is referred to by $G--$ and the Riemann tensor R_{abcd} by R- - - -. The components of non-scalar quantities are stored in a hierarchical list structure which is attached as APVAL to the appropriate atom. Thus the components of $G--$ are stored as

G- -
↳
APVAL $((g_{00}\ g_{01}\ g_{02}\ g_{03})$
$(g_{01}\ g_{11}\ g_{12}\ g_{13})$
$(g_{02}\ g_{12}\ g_{22}\ g_{23})$
$(g_{03}\ g_{13}\ g_{23}\ g_{33}))$

where each g_{ab} represents a pointer to the corresponding algebraic expression for that component. As g_{ab} is symmetric, the double appearance of g_{01}, say, indicates the same pointer. In this way, valuable store is saved by making full use of any symmetries which a tensor may possess. (For example, of the 256 components of R_{abcd}, only 20 independent components are constructed and stored.) The value of a scalar is stored as the APVAL of the relevant atom; a vector is simply a linear list of its four components and, for quantities with more than two indices, the structure is the logical extension of the type shown above.

Given the initial data $\{g_{ab}\}$, the simple action of METRIC is to calculate and print out all the independent components of the set $\{g_{ab}, g^{ab}, \Gamma^a_{bc}, R_{abcd}, R_{ab}, R, G_{ab}\}$ in that order. This order is important as each quantity is then dependent only on the components of those previously calculated. Special functions are used by METRIC for manipulating and constructing

the hierarchical structures described above. When a quantity is no longer required in ensuing calculations, METRIC automatically sets its APVAL to NIL so that, when the next garbage collection occurs, store is reclaimed.

Connected with METRIC are some extra functions which enable the programmer to have more control over its action.

(*a*) NOPRINT can be used to prevent METRIC from printing out any quantity which is not required.

(*b*) STOPAFTER provides a mechanism for terminating the action of METRIC after it has calculated a particular quantity.

(*c*) CALCULATE can be used after METRIC has finished to calculate any further quantities, or just certain components, which may be required. As well as those mentioned in the standard set above, CALCULATE can be called upon to determine the mixed components of the Einstein tensor G^a_b, the contravariant components G^{ab}, the Weyl tensor C_{abcd} and also the Ricci tensor R_{ab} directly from the Christoffel symbols Γ^a_{bc} (normally the Ricci tensor is obtained by contracting g^{ab} with R_{abcd}).

(*d*) KEEP enables the programmer to keep quantities in store which would otherwise be disposed of by METRIC.

(*e*) LOSE exists to rid the store of any quantity (or just certain components) which are no longer required for ensuing calculations.

By judicial use of METRIC and these auxiliary functions, the programmer can select only those quantities or components which he requires to be calculated and printed out, thus saving time and store. As there is complete freedom in the use of the substitution and common factor devices anywhere in a program, CLAM provides an extremely flexible way of performing these types of calculations which are frequently required by research workers in general relativity.

Other facilities are provided. The components of a tetrad can be defined by using TETRAD after which CALCULATE can be made to produce the physical components of the Ricci and Einstein tensors. METRIC2 exists for those metrics for which METRIC is unable to construct g^{ab} for itself. METRIC2 therefore takes the ten $\{g^{ab}\}$ as a second set of data and proceeds from there. Two functions LINELENGTH and NEWPAGE are provided for controlling the number of characters output per line and a page throw respectively. In Appendix 2 we display a simple CLAM program which illustrates the use of some of the functions described above; full details of their use can be found in the manual.

11. Additional information

The CLAM system is available for use on the CDC 6600/6400 computers and is written in their machine code assembler, COMPASS, within the LISP system of the University of Texas. In designing ALAM and CLAM, the emphasis had always been on producing a system which would be capable of performing the large algebraic calculations commonly encountered in relativity research. For this reason, neither system was written in LISP itself because it was doubtful whether the required efficiency could be attained.

Two versions of CLAM exist; CLAM 2.0 is the standard version for metric applications using the command language. There is no way for the programmer to reference explicitly those functions dealing with simplification, differentiation, output, etc. All unnecessary code, list structure, buffers and tables of the LISP system have been edited out to save store. The system is approximately 10K words in length and is the one referred to by the manual. A second version, CLAM 3.0 still retains all the LISP interpreter and compiler interface and can be used for non-metric applications as well (it contains CLAM 2.0 as a subset). Thus, programs can be written in LISP using CLAM functions, compiled and then executed. However, we feel the demand for this version is limited as a knowledge of LISP is necessary to use it and hence no manual has been written for it.

12. A critical analysis

In this final section we shall attempt to present an objective appraisal of the CLAM system. The unsuitability of LISP as an input language has already been mentioned in the previous section and the inconvenience of this choice has been minimised by the use of a command structure in the basic (2.0) version. The format in which algebraic expressions are input also leaves something to be desired. As the amount of input to the system is usually small compared with the amount of output, it was originally considered that this format would provide a simple and unambiguous method of input and storage of algebraic expressions in a LISP list format. However, for those applications which have large amounts of data, the input of expressions is very tedious and the resulting program is not easy to read for checking purposes. An admirable solution would be something similar to REDUCE (Hearn, 1970) which, although LISP based, has an ALGOL-like input language and a method of writing algebraic expressions very similar to FORTRAN. This would not be difficult to achieve for CLAM but we feel that it is not worth the effort as the system also suffers from the drawback of the LISP data structure.

LISP list structure provides a natural and simple method of list processing but, for the purposes of algebraic manipulation it is extremely inefficient in store usage. In ALAM/CLAM the CDR of a node invariably holds a pointer to another node whilst the CAR frequently does so as well. In algebraic manipulation it is often the case that two or more pieces of data naturally belong together and hence it would be more efficient to compact the information into consecutive locations in store rather than joining by pointers. The CAMAL system (Barton, Bourne and Fitch, 1970) provides a good illustration of how this can be accomplished. Thus, because of the data structure, some of the large calculations which ALAM and CLAM have accomplished have extended the resources of the machine to the limit. In CLAM, the wastage of store is even greater as CDC LISP has an extra CSR field (18 bits within a 60-bit word) which is not used at all except in property lists. Thus we feel that the time spent on providing good input facilities for CLAM would be better spent on researching into a more efficient data structure which, of course, would mean an entirely new system.

However, despite the above criticisms, ALAM and CLAM have proved to be extremely useful aids to research in relativity. For metric applications, the basic version, we claim, is by far the easiest system to use. This fact has been established from feedback from users and no knowledge of computing is necessary to use the system successfully.

CLAM is also extremely fast. The measurement of statistics of algebra systems poses many problems (*see* Fitch and Garnett, 1972 for instance) but we think it is true to say that there are no systems in existence which are appreciably faster (*see* Barton and Fitch, 1972, p. 285). As an example, we present the statistics for a standard test problem, the B.V.M. metric. To calculate and print out in a reasonable format all the independent components of $\{g_{ab}, g, g^{ab}, \Gamma^a_{ab}, R_{abcd}, R_{ab}, R, G_{ab}\}$ for the B.V.M. metric given only the $\{g_{ab}\}$ as initial data, CLAM takes 18 seconds CPU time in 40K words of store on the CDC 6600 (although the program will run in less store).

Acknowledgements

We should like to thank Professor F. A. E. Pirani of King's College, London, for his help and encouragement during the

course of this work. We should also like to thank Professor M. V. Wilkes and the referee for their suggestions which have improved the presentation of this manuscript.

Appendix 1

Presented below is the definition of an algebraic expression in CLAM. The notation is basically BNF. Within lists, symbols are understood to be separated by LISP blanks and three dots are taken to mean that any number of symbols of the type on either side of the dots may occur, including none at all.

⟨algebraic expression⟩ ::= ⟨AE⟩
⟨AE⟩ ::= 0|
⟨atomic-symbol⟩|
(⟨integer⟩ ⟨integer⟩)|
($ ⟨atomic-symbol⟩⟨varint⟩
⟨varint⟩ . . . ⟨varint⟩)|
(** ⟨AE⟩ ⟨AE⟩)|
(⟨infop⟩ ⟨AE⟩ ⟨AE⟩ ⟨AE⟩ . . .
⟨AE⟩)|
(⟨monop⟩ ⟨AE⟩)

⟨integer⟩ ::= ⟨LISP positive integer⟩
⟨varint⟩ ::= 0 | 1 | 2 | 3
⟨infop⟩ ::= + | *
⟨monop⟩ ::= − | SIN | COS | COT | SINH | COSH | LOG | EXP

Appendix 2

As an example we exhibit a program in CLAM to determine the R_{1313} and R_{2323} components of the Riemann tensor for the Schwarzschild metric

$$ds^2 = \left(1 - \frac{2m}{r}\right) dt^2 - \left(1 - \frac{2m}{r}\right)^{-1} dr^2 - r^2 d\theta^2 - r^2 \sin^2\theta d\phi^2$$

The $\{g_{ab}\}$ are printed out as a check on input but the common factor, $A = r - 2m$, is only substituted for when the two Riemann tensor components are printed out. The four independent variables (t, r, θ, ϕ) are input as (T R E P) and the quantities $g, g^{ab}, \Gamma^a_{bc}, R_{abcd}$ are referred to in CLAM by *G*, G++, GAM+−−, R−−−− respectively.

The program will run in well under 15K and takes less than a second to execute on a CDC 6600.

```
CLAM PROGRAM:


VARIABLES (T R E P)
FUNCTION (A (R))
FACTOR (A (+ R (- (* (2 1) M))))
NOPRINT ((*G* G++ GAM+--))
STOPAFTER (GAM+--)
METRIC (( (* A (** R (- (1 1)))) 0 0 0
          (- (* R (** A (- (1 1))))) 0 0
          (- (** R (2 1))) 0
          (- (* (** R (2 1)) (** (SIN E) (2 1))))
       ))
LOSE (G--)
LOSE (G++)
SWITCHFAC (ON)
CALCULATE (R---- (1 3 1 3))
CALCULATE (R---- (2 3 2 3))

CLAM OUTPUT:


      -1
G   =AR
  00


G   =0
  01


G   =0
  02


G   =0
  03


      -1
G   =-RA
  11


G   =0
  12


G   =0
  13


      2
G   =-R
  22


G   =0
  23


      2   2
G   =-R SIN (E)
  33


                2          -1          2
R     =(1/2)RSIN (E)(R-2M)   -(1/2)SIN (E)
  1313

               2
R     =-2MRSIN (E)
  2323
```


References

BARTON, D., BOURNE, S. R., and FITCH, J. P. (1970). An algebra system, *The Computer Journal*, Vol. 13, No. 1.
BARTON, D., and FITCH, J. P. (1972). Application of Algebraic Manipulative Programs in Physics, *Rep. Prog. Phys.*, Vol. 35, pp. 235.
FITCH, J. P., and GARNETT, D. J. (1972). *Measurements on the Cambridge Algebra System*, Proc. ACM Int. Comput. Symp., Venice.
HEARN, A. C. (1970). *REDUCE User's Manual*, Stanford Artificial Intelligence Project Memo. AIM-133.
D'INVERNO, R. A. (1969). ALAM-Atlas Lisp Algebraic Manipulator, *The Computer Journal*, Vol. 12, No. 2, pp. 124-127.
D'INVERNO, R. A. (1970). *The Application of Algebraic Manipulation by Computer to some Problems in General Relativity*, King's College, London (Ph.D. Thesis).
D'INVERNO, R. A., and RUSSELL-CLARK, R. A. (1971). Classification of the Harrison Metrics, *J. Math. Phys.*, Vol. 12, p. 1258.
D'INVERNO, R. A., and RUSSELL-CLARK, R. A. *The CLAM Programmer's Manual*, Part 1, Simple Applications of CLAM to computing curvature tensors and some related quantities, (1971), King's College, London. Part 2, Additional Facilities, (1972), University of Cambridge, Computer Laboratory.
RUSSELL-CLARK, R. A. (1973). *The Application of Algebraic Manipulation by Computer to some Problems in Gravitational Radiation Theory*, King's College, London (Ph.D. Thesis).