ACM

SIGSAM Bulletin

A Quarterly Publication of the
Special Interest Group on Symbolic & Algebraic Manipulation

Volume 17, Number 3&4 August & November 1983 (Issue Number 67&68)

ORTOCARTAN - A PROGRAM FOR ALGEBRAIC CALCULATIONS IN GENERAL RELATIVITY

Andrzej Krasiński
N. Copernicus Astronomical Center
Polish Academy of Sciences
Bartycka 18, 00 716 Warszawa, Poland

1. Why was this program created and what is it good for?

The supply of computer programs for algebraic calculation seems at present sufficient to satisfy the needs of any potential user (1). It would thus seem natural for a future user to ask for a copy of one of the existing programs. However, it turns out to be quite difficult in practice. Some algebraic systems are not distributed at all (MACSYMA (2) was until recently an example). Some others (e.g. SHEEP (3, 4)) exist only for definite types of computers, and adapting such a program to another computer is, for an inexperienced programmer, a task comparable in difficulty to writing a program from scratch. Several specialized programs will not be suited to the particular application intended. Finally, establishing a local group of experts in algebraic programming in a place where this activity was not represented before is an obvious (*) convenience for the users. Therefore we decided in 1977 to write our own program.

The program calculates the Riemann, Ricci, Einstein and Weyl tensors from a given metric tensor using an ORThOnormal set of CARTAN forms. This calculation is performed in the theory of relativity in order to find the left-hand side of the Einstein's field equations. It consists of repeated application of differentiations, multiplications and additions to polynomials of gradually increasing complexity, and of inverting a single 4×4 matrix with algebraic elements (5). The terms of the polynomials may in general contain any functional expressions, including arbitrary functions. This is why every algebraic program for use in the relativity theory must contain very general and efficient subroutines for algebraic simplification and differentiation. If they are good enough for this calculation, they will be useful for many other applications. ORTOCARTAN could thus be relatively easily extended for other specialized programs. The extensions, apart from the program for inverting matrices of arbitrary rank, did not require much ingenuity in programming, so they will not be discussed here (see (6)). This article will describe the computer science aspects of ORTOCARTAN. For a review oriented towards users see (5) and the user's manual (7). A detailed reference manual describing all the functions in the program is also available (8).

(*) The author is at present alone responsible for the maintenance and distribution of ORTOCARTAN. However, the program was created by an informal team whose other members have shares at least equal to the author's share in the final outcome. The team included M. Wardęcki (we adapted his simplifying subprogram written for another purpose), M. Perkowski (he wrote the first complete code of the program and helped the author to learn LISP) and Z. Otwinowski (his brilliant ideas resulted in reducing the execution time by the factor 5).

2. The language.

ORTOCARTAN is written in LISP. Recursion, the characteristic feature of LISP, is used in the program in an essential way in several places (e.g. in simplification, substitutions, differentiations, matrix operations and printing). It would thus be very difficult if possible at all to implement the same algorithm in another programming language. The program is available in four different machine implementations of LISP: 1. The Cambridge LISP for IBM computers (9); 2. The University of Texas LISP 4.1 for CDC computers (10); 3. The SLISP/360 for IBM computers (11); 4. A version for UNIVAC computers (12). The author was himself involved in the first three versions only. The program was originally written in U.T. LISP. In order to rewrite it into the Cambridge LISP, it was

necessary to DEFINE 12 additional simple functions to make up for a few system functions of U.T. LISP and to redefine or rename 7 others which expected a different format of their arguments or had different names. More problems were caused by unclear error messages and discrepancies between the reference manual and the actual working of SLISP/360. Difficulties resulted also from different "philosophies" of the various implementations. For instance, in U.T. LISP the PROG- and global variables practically do not differ in any way, while Cambridge LISP and SLISP/360 make a distinction between GLOBAL, FLUID and LOCAL bindings. Also, the very different treatment of vectors and arrays in U.T. LISP and in Cambridge LISP was a source of some trouble. In the end, however, it seems to be a better policy to risk facing problems of this kind by taking the advantages of each implementation to the extreme than to avoid problems with portability by sticking to the "Standard LISP" (13). The "Standard LISP" is rather simple and obeying its rules would force the programmer to give up many useful conveniences available in U.T. LISP. Besides, this "Standard" is so far only a proposal which was not exactly followed in any implementation (even in SLISP/360 in spite of the manual's title (11)), so obeying it does not guarantee portability either.

3. Types of variables.

There are 5 types of atomic variables in ORTOCARTAN:

1. Constants.
2. Coordinates (the independent variables for differentiation).
3. Arbitrary functions.
4. Symbols for explicitly defined expressions (to be used as abbreviations in all operations except differentiation which acts on the expressions themselves).
5. Elementary functions: times, plus, expt, minus, exp, log (to base e), cos, sin, tan, cot, cosh, sinh, tanh, arctan, arcsin, arcsinh, arccosh, arctanh, der (the derivative) and int (the indefinite integral which cannot be calculated, but can be differentiated).

The first four types are declared by the user. The elementary functions are stored in the program and must not be declared. Each atomic variable is assigned a priority, i.e. a number which is stored in its property list with the indicator PRIOR. Numbers have higher priorities than variables, lists (which represent functional expressions) have lower priorities than atoms. Of two lists, the one has higher priority whose CAR has higher priority. If both CARs have equal priorities, the CDRs are compared in the same way. Two objects can have the same priority only if they are identical. In this way, each list of expressions (atoms or lists) can be uniquely ordered according to descending priorities. Such ordering is performed in sums and products, and this simplifies further operations on them.

Each function is declared by the user together with the list of its arguments. The arguments can be themselves the names of other functions or of symbols. Such composite functions (functions of functions of functions...) are allowed to any arbitrary depth and will be correctly differentiated. During the calculation, each function has its arguments stored in the property list with the indicator DEPS. Similarly, each symbol has the expression it represents stored in the property list with the indicator CONTENT.

4. Internal representation of numbers and algebraic expressions.

Rational numbers are represented in the program as dotted pairs, (1 . 2) meaning 1/2 (in user's data the dot is omitted, i.e. the numbers are two-element lists). Floating-point numbers are not allowed because non-rational numbers must be dealt with precisely. The program can simplify and multiply rational numbers and simplify their rational powers. The following principles are used in simplifying fractional powers: 1. Make the exponent positive by the identity $a^{-p} = (1/a)^{p}$; 2. Change the exponentiated number to an integer by the identity $(m/n)^{p/q} = n^{-p} (mn^{q-1})^{p/q}$; 3. Change the exponentiated integer so that the numerator of the exponent equals 1 by the identity $n^{p/q} = (n^{p})^{1/q}$; 4. Try to make the denominator of the exponent as small as possible by detecting simplifications like $9^{1/4} = 3^{1/2}$. The program would not perform more sophisticated operations like e.g. $2^{1/2} \cdot 6^{1/2} = 2 \cdot 3^{1/2}$. This did not cause any trouble so far, and the program SHEEP does very well even without as much arithmetics as ORTOCARTAN has (14). The user can always use substitutions to smooth out such failures.

Algebraic expressions are represented in the prefix notation, (PLUS A B), (TIMES A B) and (EXPT A B) representing A+B, A·B and A^{B}, respectively. Formally, nonrational numbers are algebraic expressions.

5. The algorithm for algebraic simplification and substitutions.

The goal of simplification cannot be uniquely defined for a completely arbitrary expression (15). However, a program will work reasonably well if, in addition to a fixed set of rules, it will obey the ad hoc commands of the user. ORTOCARTAN follows the following rules:

1. Each sum or product is ordered in a unique way as described in sec. 3.
2. Exponentiations of sums are expanded if the exponent does not exceed 3 (the user can change this limit or suppress exponentiation altogether).

3. Products which contain sums as factors are fully expanded with use of the distributivity rule (in accord with rule 2; the user can avoid it by defining an atomic symbol to represent any given sum).

Other rules, like summing up exponents in products, are quite obvious.

To each algebraic operation there exists in ORTOCARTAN a specialized simplifying function which operates only on the top level of the expression and makes the assumption that

(A) All the sub-expressions were already simplified before.

For instance, the function SPLUS will sum up numerical coefficients of terms in a sum, deleting a term if its coefficients sum up to zero, but will not analyse the terms themselves, even if they contain sums on deeper levels. The function STIMES will sum up the exponents of factors in a product, will expand the product if some terms are sums (calling SPLUS to simplify the result), but will not analyse the factors. There are 6 specialized functions, with self-explanatory names: SMINUS, SPLUS, SLOG, STIMES, SEXPT and SEXP.

A thorough simplification on all levels is done by the function SIMPLIFY which is recursive. It goes into its first argument (the expression to be simplified) until it encounters a list whose all elements are atoms or numbers. On that level, it recognizes by the first element of the list which specialized simplifying function is to be called and calls it. It goes then back to the previous level, having thus guaranteed that the deeper level was already simplified, and so on until the top level. Such recursive simplification must be performed only once, on the user's input data. Later, the assumption (A) is guaranteed to be fulfilled. The function SIMPLIFY can operate in two modes: in the "hard" mode in which it modifies the expression, and in the "soft" mode in which it produces a simplified copy of the original expression. The mode is fixed by the second argument which is T or NIL.

In U.T. LISP the function SUBST is available which replaces every occurrence of its second argument S2 in the third argument S3 by the first argument S1. However, SUBST copies S3 on every call, so many calls fill the core with a great number of list structures. Moreover, it replaces every S2 by always the same copy of S1, so that only the soft mode of the function SIMPLIFY could be used, resulting in further losses of core. We designed our own function SUBSTITUTE which copies S3 only when S2 is actually found in it, and does so only upwards from the levels on which S2 was found. At the same time, it replaces every occurrence of S2 by a different copy of S1, so the simplifying functions can safely modify the resulting structure. Each actual replacement is followed by the recursive algebraic simplification, also only upwards from the levels on which the replacement occurred. The last point is important: one performs substitutions just in order to cause additional simplifications, so simplification should not wait until all substitutions are finished. Again, however, SHEEP is doing well ignoring this principle (14).

The substitutions are literal, with no pattern-matching. They can replace, though, a part of a sum or of a product (not necessarily the whole sum or product). How they are coded in the data and then stored and processed in the program will not be described here, since we would have to go into too much detail of the Einstein's field equations, but see (7-8).

With the simplifying subprogram given, differentiation is quite easy to design.

6. The algorithm for inverting matrices.

The single matrix which is inverted in every run of ORTOCARTAN is always of rank four, so its determinant could be calculated by the Laplace algorithm in a reasonable time. However, we wanted to make this subprogram directly accessible to the users. It was necessary then to use a faster algorithm, applicable also in higher dimensions. (For a matrix of rank n, the Laplace algorithm requires n! sub-determinants to be calculated. It is definitely too much already with n = 8).

We used the algorithm which first makes the matrix triangular. It requires roughly $n^3/3$ multiplications and additions to be performed on the elements of the matrix, and the determinant is then equal to the product of the n elements on the main diagonal. This is still realistic with $n < 40$, provided the elements do not become long sums in the process (with a completely general symbolic matrix, the determinant is a sum of n! terms, so this algorithm is no better).

The calculation proceeds as follows. The element M11 of the matrix M is picked out. If it is zero, the program goes along the first column of the matrix until it finds the first nonzero element. If no such element is found, the determinant obviously equals zero. If such an element is found, say it is $Mk1$, then the rows 1 and k are interchanged, $Mk1$ playing now the role of M11, and the sign of the result will be later reversed. Suppose then $M11 \neq 0$. In this case, the first column is surveyed for other nonzero elements. If there are none, the determinant equals M11 times the sub-determinant (Mij), $i = 2, \ldots, n$; $j = 2, \ldots, n$; of rank n-1. The subdeterminant is calculated recursively by the same method. If another nonzero element, $Mk1$, is found in the first column in the row k, then each element Mkl in the k-th row is replaced by $(Mkl - Mk1 \cdot M1l/M11)$, $Mk1$ being thus replaced by zero. This procedure, which does not change the value of the determinant, is repeated until M11 remains as the only nonzero element in the first column. The sub-determinant of rank

n-1 is then to be calculated. The calculation is repeated recursively until the rank goes down to 1.

There is a pitfall in this algorithm. Since the simplifying procedure automatically expands products involving sums, it would find (A+B){1-B/(A+B)} to equal {A+B
2
-AB/(A+B)-B /(A+B)} and would not recognize that this equals simply A. Therefore a precaution must be taken for the case when M11 is a sum. In that case, M11 is replaced by an atomic symbol G11 generated in the program. Only after the whole determinant is calculated, the corresponding sums are substituted back for the symbols in a reversed order. The order must be reversed, since the later replaced sums contain the earlier generated atoms in their terms. In this way, negative powers of each symbol are cancelled before the symbol is replaced by the appropriate sum. The generation of the atomic symbols and the two-way substitutions are performed automatically and in full conspiracy so that the user does not notice them.

It would be logical now to invert the triangular matrix and use the result to reconstruct the desired inverse matrix. Unfortunately, this reconstruction relies on factorization of polynomials. The polynomials are quite simple and guaranteed to factorize, but our simplifying procedure cannot factorize at all and would have to be extended. It seemed more economical to invert the matrices by the old rule "minor divided by determinant".

The algortihm for calculating determinants could be further improved. It may happen that some other column than the first one has the most zeros. If it is interchanged with the first column, the "triangularization" of the matrix is done faster. Inspired by an optimistic report (16) the author generalized the algorithm to include this trick. The new algorithm worked faster indeed, but failed on a curious 9×9 matrix. The matrix contained two columns with 7 zeros each, while every other column contained less zeros. Both of these 7-zero columns seemed equally good to start with. However, if the "triangularization" was started at the first of them, most other zeros in the matrix were killed and even a long time limit was not sufficient to complete the calculation. If it was started at the second 7-zero column, many zeros were preserved and the calculation succeeded in 710 seconds (the SLISP/360 version). The algorithm could not recognize the best one among columns with equal numbers of zeros, and started the work always at the leftmost of them. It succeeded only after the columns were interchanged "by hand".

The matrix is available from the author. The problem arose in research on galaxy formation (17).

7. Input format and prints.

The user writes his data in a format which reminds that of FORTRAN: multiplications are denoted by asterisks (which must not be omitted), exponentiations by double asterisks. These data are translated into the LISP prefix format by a simple subprogram. The results are printed in the normal mathematical format, with exponents above the main line and subscripts below it. The program for printing is able to print formulae in which exponents carry their own exponents or subscripts. It is guaranteed to work correctly only if the expressions to be printed are of the format produced by the function SIMPLIFY. For instance, (TIMES (EXPT A 2) (EXPT B 2))
2 2
will be printed correctly, as A B . However, (EXPT (TIMES A B) 2) which is the same expression, but not SIMPLIFYied,
2
would appear in print as TIMES (A, B).

The printing program analyses the form of the expression to be printed and decides in which line to place each atom encountered. The atom becomes the first element of a dotted pair. Its second element is the number of the column in which the first character of the atom should be placed. The dotted pair is inserted into the appropriate list, each list corresponding to one line of print: the line for subscripts, the main line, the first line of superscripts, the second line of superscripts, and so on. The program keeps track of the first free column and compares it with the beginning of the right margin (adjustable by the user, default 133). If the next atom would not fit into the space left, a special break-sign is inserted into all the lines which causes carriage return during printing, and the column counter is reset to 4.

This is a pure LISP program, it uses no printer-driving commands apart from the functions PRINT, PRIN1, TERPRI and OTAB.

8. Core-saving methods.

Drawing from other programmers' experience we have put a strong emphasis on using as little core as possible. On the other hand, we did not want to achieve this at the cost of users' convenience or of abilities of the program. This led us to several ideas. Those which resulted in the greatest savings are described below.

The objects calculated in the relativity theory carry two to four indices, each ranging from 1 to 4 or from 0 to 3. To calculate all the components of such an object, all possible sets of its indices must be referenced. In the beginning, we generated the values of the indices by a function similar to the ALGOL statement FOR i=<1> STEP <s> UNTIL <u> DO <function> and formed sets of indices by executing LIST on two, three or four arguments. This resulted in choking both the full word space (with numbers) and the free space (with lists). Since however the indices

run always through the same four values, they need not be generated. One can define the global variable LIND by (SETQ LIND (QUOTE (0 1 2 3))) and call (MAPC LIND (FUNCTION (LAMBDA (I) <function-body>))) instead of calling DO. Similarly, instead of generating always new lists of indices, one can define three global variables with the initial values LIJ = (NIL NIL), LIJK = (NIL NIL NIL) and LIJKL = (NIL NIL NIL NIL). These are working lists of indices. Their elements can be manipulated through the function RPLACA. In this way one uses the same 4-element list 256 times instead of generating it 256 times anew. Although this method made the code of the program longer by about 10%, it reduced the execution time by the factor 2 and the garbage collections by more than 10 times.

In U.T. LISP it is easy to handle files and overlays. We used this facility to further reduce the core-requirements. The program was divided into 3 parts. Part 1 contains the printing program and forms a separate overlay. Part 2 contains the programs for algebraic simplification and differentiation which are needed during the whole calculation, and also forms a separate overlay. Parts 1 and 2 can (and should) be compiled. Part 3 contains LISP-definitions of all the other functions (input analysis, inverting matrices, the Einstein's equations, etc.). The functions from part 3 are used only during brief definite periods of the calculation.

The overlay with part 2 is first loaded into core, and the definitions from part 3 are read until the program is able to do its first job (translate the user's input into the LISP notation for instance). The job is then done, and the functions which did it will not be useful anymore. Their definitions are then erased, by (REMPROP (QUOTE <the function-name>) (QUOTE EXPR)). Further definitions are read only when they are first needed. In this way the program gradually learns while it works, and at the same time gradually destroys itself. At any moment, at most half of the code actually resides in core. This has one small disadvantage: the program may be called only once in each machine-job.

The results of the calculation are not immediately printed, but stored on the file PRINTS in their unreadable LISP-format until part 2 is ready with its work. The overlay with part 1 (the printing program) is then loaded into core and overwrites part 2. The file PRINTS which was an output file for part 2 becomes the input file for part 1, and the results are now neatly printed on the standard output.

U.T. LISP has also a very convenient virtual-memory facility which allows for further savings of core. Unfortunately, we could not use it because our copy of LISP 4.1 contained a bug. The idea is as follows. During the calculation, ORTOCARTAN creates several arrays whose elements are large expressions. The arrays are stored in core till the end of the calculation. However, only a few components of them may be needed simultaneously at a time (at most five), while there are more than 400 of them altogether. The expressions could be thus stored on a random-access disk file while only their disk-addresses would be kept in core. Frequent referencing the virtual memory would make the calculation a little slower, but 5 expressions could use the space occupied before by 400.

9. Actual and possible applications of ORTOCARTAN.

So far ORTOCARTAN was applied to problems in the gravitation theory, mostly by the author himself. It was used to investigate a generalized cosmological model (18) and symmetry properties of the curvature tensor (19). At the University of Cologne it was applied to smaller problems in a generalized theory of gravitation, the Poincare gauge field theory (20). It was also used by several colleagues of the author to various small calculations.

It is not likely that ORTOCARTAN would appear well suited to problems outside the gravitation theory. For instance, it cannot calculate any integrals what is necessary in perturbative calculations of quantum electrodynamics. However, it could be extended for other specialized procedures and other general abilities at a moderate effort. These are for example: processing complex numbers and functions, processing truncated power series, substitutions by pattern-matching, the possibility to write programs in ORTOCARTAN without resorting to LISP. These extensions would enlarge the area of application in the gravitation theory, but so far there was no pressure in this direction from the users.

There are no plans to include calculating indefinite integrals and simplifying rational functions. These would require more concentrated efforts and it is not likely that anything useful could be produced in the half-amateur spare time mode we worked so far.

10. Existing implementations, availability and documentation.

ORTOCARTAN is available at the following sites:

1. The Cambridge LISP version: on an IBM computer at the University of Cambridge (England) and on IBM-compatible Amdahl and Siemens computers at the Max-Planck-Institute of Physics and Astrophysics in Garching/Munich (W. Germany).

2. The U.T. LISP version: on CDC Cyber computers in regional computer centers in Warsaw and Cracow (Poland) and at the University of Cologne (W. Germany).

3. The SLISP/360 version: on a Siemens 4004 (IBM-incompatible) computer at the University in Konstanz (W. Germany).

4. The UNIVAC version: at the University in Istanbul (Turkey).

The program is at present in active use only in Warsaw and in Cologne, and only the author can promise to respond to re-

quests from interested users. Magnetic tape records of the first three versions are available from the author.

Also available is the user's manual (7) which contains sample prints from simple tests and is intended for those who do not know LISP and do not wish to learn it, but are familiar with Einstein's relativity theory. The manual refers only to version 1. New edition which will describe the later extensions of the program and its other computer implementations is at present being prepared in Cologne. It will be stored on a magnetic tape. A detailed reference manual of ORTOCARTAN (8) describes the working of all the functions in the program. An updated version thereof is being prepared in Cracow, and is also to be stored on a magnetic tape.

11. Comparison with other programs.

Comparisons between various programs for algebraic calculations are not really objective, even in the points concerning measurable quantities like the time required to perform a standard calculation or the core memory needed for this. For instance, the four versions of ORTOCARTAN follow, of course, the same algorithm. In spite of that, the times taken to perform the same calculation are, respectively, in the ratios 1:15:30:180. The differences result from the different efficiencies of the corresponding LISP implementations. Cambridge LISP has a readily usable compiler, so the version 1 was always run in the compiled form and without the overlay-handling described in section 8. In U.T. LISP compilation is not straightforward (e.g. large function definitions must be split into small parts (9)), and with our difficult conditions of access to the computer in Warsaw we preferred not to attempt it. Instead, Z. Otwinowski rewrote the definitions of a few most frequently called functions into the CDC assembler code, optimizing them by-hand. For these individual functions, the gain in speed was by a factor of up to 100, in the whole program this factor was between 5 and 10. Such partly assembled version was used for the comparison below. In SLISP/360 the compilation should theoretically be as easy as it is in Cambridge LISP. However, difficulties are encountered of which the manual (11) does not warn. After many experiments, only 1/6 of the body of ORTOCARTAN was found to be compilable. The version 4 was not compiled at all.

A comparison is further complicated by the fact that the data taken from literature come from different times, and one is comparing the most recent version of his own program with older versions of other programs. The subjective impressions of users as to which programs are easier to learn and apply, although very important, cannot be compared as there is no user who would have the same amount of experience with a number of different programs.

Unreliable as they are, comparisons must be made. The table below compares versions 1 and 2 of ORTOCARTAN (data from 1982 and 1979) to programs written in five

TABLE: Times taken and core-memory used
by 6 different algebraic programs to calculate the Einstein's tensor

Language or program	Time (sec.)		Core	
	Problem 1	Problem 2	Problem 1	Problem 2
LAM (24)	104	210	400 kbytes	x
ALTRAN (25)	255	691	350 kbytes	x
FORMAC (26)	162	321	300 kbytes	x
REDUCE (27)	234	856	500 kbytes	x
SYMBAL (28)	35	47	33 000 words (*)	x (*)
ORTOCARTAN in Cambridge LISP	35	98	800 kbytes	800 kbytes
ORTOCARTAN in U.T. LISP	536	1104	ca 53 000 words	ca 53 000 words

x - data missing
(*) This large memory requirement must have resulted from the author's inappropriate handling of the Amdahl/Siemens computers in Garching. ORTOCARTAN processed tests of similar complexity on the IBM in Cambridge at 300 kbytes.

general-purpose languages for algebraic calculation. The data on these other programs come from a fairly objective comparison made by I. Cohen, Ö. Leringe and Y. Sundblad in 1976 (21). The authors coded the same algorithm in each of the languages, and ran each program on the same tests. The table favors ORTOCARTAN because the latter was not written in a higher-level language for algebraic programming, but in LISP on which some of the other languages are based.

Two calculations were chosen for the comparison. In the problem 1 (22) the programs just calculate the Einstein tensor for a reasonably complicated metric tensor. In problem 2 (23), they calculate the Einstein tensor up to linear terms in a small parameter for another metric tensor. Problem 2 thus tests a program's ability to handle user-generated substitutions.

Unfortunately, no data were available on SHEEP (3, 4) which is a more advanced program of similar type as ORTOCARTAN and has been used by a larger number of users, all of which gave it enthusiastic opinions. SHEEP is interactive what is its strong advantage. ORTOCARTAN was not made interactive because the terms of access to the CDC computer in Warsaw do not allow for reasonable work in the interactive mode. Working in the batch mode with a short turnaround time is not a great obstacle, however.

ACKNOWLEDGMENTS. The author wishes to thank J. Richer and A. Norman for implementing ORTOCARTAN in Cambridge LISP and kindly supplying him with the product, and F. Hehl and M. Kwaśniewski for their cooperation in updating the documentation. Sincere thanks go to M. Perkowski without whose colaboration the program would never have been written, and to Z. Otwinowski for a lot of good suggestions how to improve the algorithms. This work has been partly supported by the Alexander von Humboldt Foundation and by Deutsche Forschungsgemeinschaft.

REFERENCES

(1) V. P. Gerdt, O. V. Tarasov and D. V. Shirkov, Sov. Phys. Uspekhi 23, 59 (1980) (Russian version 130, 113 (1980)).
(2) A. Bers, J. L. Kulp and C. F. F. Karney, Computer Phys. Commun. 12, 8 (1976).
(3) I. Frick, The computer algebra system SHEEP, what it can and cannot do in general relativity. University of Stockholm report 77-14 (1977).
(4) I. Frick, SHEEP user's guide. University of Stockholm report 77-15 (1977).
(5) A. Krasiński and M. Perkowski, Gen. Rel. Grav. 13, 67 (1981).
(6) A. Krasiński, in: 10th International Conference on General Relativity and Gravitation, Padova 1983, p. 433.
(7) A. Krasiński and M. Perkowski, The system ORTOCARTAN - user's manual. Report of the N. Copernicus Astronomical Center (1980).
(8) A. Krasiński, M. Perkowski and Z. Otwinowski, The system ORTOCARTAN for analytic calculations, detailed description. Report of the N. Copernicus Astronomical Center (1979).
(9) Cambridge LISP reference manual (distributed on tape by A. Norman, Cambridge).
(10) E. M. Greenawalt, J. Slocum and R. A. Amsler, U.T. LISP reference manual. University of Texas at Austin 1975 (distributed on tape).
(11) J. Fitch, Manual for Standard LISP on IBM system 360 and 370. University of Utah, Salt Lake City 1978 (distributed on tape).
(12) G. Üçoluk, private communication.
(13) J. B. Marti, A. C. Hearn, M. L. Griss and C. Griss, Standard LISP report. University of Utah, Salt Lake City 1978 (distributed on tape).
(14) I. Frick, private communication.
(15) M. Genesereth, in: Symbolic and algebraic manipulation. Proceedings of EUROSAM '79. Edited by E. W. Ng. Lecture Notes in Computer Science no 72, Springer Verlag, Berlin-Heidelberg 1979, p. 23.
(16) J. Smit, page 74 in Ref. 15.
(17) M. Corona, private communication.
(18) A. Krasiński, Gen. Rel. Grav. 13, 1021 (1981).
(19) A. Krasiński, page 290 in Ref. 6.
(20) P. Baekler and F. Hehl, in: Gauge theory and gravitation. Lecture Notes in Physics vol. 176 p. 1. Springer Verlag, Berlin - Heidelberg 1983.
(21) I. Cohen, Ö. Leringe and Y. Sundblad, Gen. Rel. Grav. 7, 269 (1976).
(22) H. Bondi, M. van den Burg and A. Metzner, Proc. Roy. Soc. A269, 21 (1962).
(23) H. Levy, Proc. Cambridge Phil. Soc. 64, 1081 (1968).
(24) R. A. d'Inverno, Comput. J. 12, 124 (1969).
(25) A. D. Hall, Commun. ACM 14, 517 (1971).
(26) C. Fike, PL/1 for scientific programmers. Prentice Hall, Englewood Cliffs, N.J. 1970, chap. 12.
(27) A. C. Hearn, REDUCE2 user's manual. Salt Lake City, Utah 1979.
(28) M. E. Engeli, Ad. Inf. Sys. Sci. 1, 117 (1969).