GRG_{EC}: Computer Algebra System for Applications to Gravity Theory

S.I. Tertychniy[†] and I.G. Obukhova[‡]
[†]National Institute of Physical-Technical
and Radiotechnical Measurements
(VNIIFTRI)
Mendeleevo, Russia, 141570
[‡]Moscow State Pedagogical University,
Pirogovskaya 1, Moscow, Russia, 119882

Abstract

We present a general outline of the specialized computer algebra system GRG_{EC} intended for symbolic calculations in the field of the gravitation theory, the classical field theory on a curved background, and the adjacent methods belonging to the differential geometry. The distinctive features of the GRG_{EC} input language are exhibited and the noteworthy elements of the system design are discussed.

1 Introduction

The computer algebra system GRG_{EC} was designed to address a wide spectrum of possible applications to the gravitation theory and the adjacent topics originated from the classical field theory on a curved space-time and the differential geometry.

Concerning the application field mentioned, it should be remembered that, since the early sixties, general relativity has been known as a promising field for successful application of computer algebra systems. A noteworthy development of this direction was stimulated, in particular, by the existence of a considerable number of important problems in the gravity theory, that require involved symbolic manipulations which computer algebra methods were usually developed to deal with. Compared with other approaches, computer algebra often proves to be especially convenient and efficient for tackling these problems.

Nevertheless, in spite of considerable efforts by programmers and system designers, which resulted in a series of packages (e.g., Mathematica/Mathtensor [1], Macsyma [2], SHEEP / CLASSI / Stensor [6], Reduce / Excalc [4], Maple / GRTensor [3] and others), the present state of affairs in the field of the application of the computer algebra to gravity theory seems on the whole to be far less brilliant than one might expect.

In particular, one could not claim that there exists software which would be commonly accepted now as a satisfactory working tool fairly suitable for the majority of theorists, especially non-programmers, specialized in the gravity theory. Although the literature gives some evidence of successful applications of symbolic manipulation packages in the gravity theory, one is under the general impression that computer algebra methods remain more of an enthusiasts' activity than a regular routine tool of current research.

We do not intend to discuss why such a situation has arisen, but only limit ourselves by the statement that, in our opinion, the development of symbolic manipulation packages suitable for treating problems typical for classical field theory including general relativity still remains an important subject of applied programming.

Several different approaches can be suggested for achieving success in specialized scientific software design. A particular one, which underlies the design of GRG_{EC}, assumes that one of the most important points is to provide a maximally convenient interface which should be based on an immediately understandable input language. Furthermore, it is desirable for that language to be as close as possible to the one used for the description of the basic notions and relations in the framework of application field itself.

As far as possible, a user must be released from the duty 'to write a program' — at least, in the sense usually associated with these words. Rather, a user is merely to describe the initial data in an immediately understandable form and then to specify a problem to be solved, operating mostly with 'common' words and expressions and more or less standard mathematical notations.

Of course, workable software should possess many other capabilities, such as, e.g., an implementation of the most important mathematical relations in the application field, providing the efficiency of the algorithms with reasonable run time on real jobs, portability of the source code (as well as the whole system), etc. These and other similar necessary objectives were also borne in mind during the development of GRG_{EC}.

On the other hand, such advanced facilities as a system of menus, graphic window interface and similar tools are not realized in GRG_{EC} yet (which is substantially connected with the current state of the background software utilized, see below). At the same time there should be no conflict between the two aspects, and a clear readable input language could be an important element of the advanced tools of system control.

It seems worthwhile to make an additional remark here. There exists the computer algebra system by V.V. Zhytnikov [11] with almost identical name, GRG, and similar field of applications. The point is that GRG and GRG_{EC} have grown from the same root which had been cultivated by a single team [10]. Having lost the possibility (though not the wish) to work together, we worked independently further

SIGSAM BULLETIN: Communications in Computer Algebra volume 31, number 1, issue 119, March 1997. This article has passed the review process described in "Formal Review of Articles in the BULLETIN", which appeared in volume 30, number 3, issue 117, *pp.* 2–3.

with our own branches of the initially common project. Now these systems are different.

The source code of GRC_{EC} amounts now to approximately 1 Mb. The compiled binary code occupies about 500 Kb. GRC_{EC} is currently regarded as freely distributed software.

2 General characteristic of the system

GRC_{EC} is based on the general purpose computer algebra system *Reduce* developed by A. Hearn (see Refs. [4]). This system is extensively exploited for handling a wide variety of physical and applied mathematical problems.

The language chosen for the realization of GRC_{EC} is however not *Rlisp*, the basic language of the *Reduce* coding, but the STANDARD LISP dialect[1] supported in frames of the *PSL* (*Portable Standard Lisp*) package. At the same time the language used for the communication of a user with GRC_{EC} is a product of independent development. It closely refers to the application field and simulates at the same time the elements of the natural language.

An important feature of the GRC_{EC} system, which is inherited from its base (*PSL* & *Reduce*), is a high degree of portability, being essentially the same as for *Reduce* itself (see Refs. [4] for the characteristic of *Reduce*'s portability). The OS-dependent component of GRC_{EC} is very compact and the system is easy adjustable to a majority of the popular computer platforms including *Unix* workstations and personal computers.

Generally speaking, GRC_{EC} does not utilize the total scope of the *Reduce* facilities exploiting only (i) its algebraic processor, i.e. the routine performing transformations and simplifications of the general mathematical expressions, (ii) the corresponding tools providing the control over the algebraic processor, such as substitution handling routines (enabling one to realize a wide class of pattern matchings) and the flags handler (controlling *Reduce*'s dynamic options), (iii) the routines realizing an output of general mathematical expressions in a form convenient for immediate perception. Thus, being based on *Reduce*, GRC_{EC} nevertheless is not actually inseparably linked to it. Of course GRC_{EC} needs a *PSL* environment for its own run. At the same time the manipulations with formulas could be in principle realized over another symbolic manipulation package which is able to provide a relevant interface with operations equivalent to functions (i)-(iii) listed above.

In principle, one might split GRC_{EC} into the following functional constituents:

- the input interface which is realized in a form of the interpreter of the special language of *problem specifications*;
- a number of subpackages realizing a collection of mathematical methods of modern differential geometry including the exterior forms calculus, the spinor algebra and analysis, the classical Riemannian geometry methods and some other complements which, in fact, enhance the standard capabilities of *Reduce*'s algebraic processor;
- the collection of routines realizing the specific mathematical relations taking place in the general relativity, the classical field theory, etc.

[1] These two languages are basically equivalent. However they support different syntaxes and realize distinct ways of processing a source code, entailing some distinctions in practice of the corresponding programming techniques.

The system allows one to calculate or subject to other processing more than a hundred so-called *data objects* modelling the basic notions of field theory in curved space-time and differential geometry. However GRC_{EC} is *not* suited to the purpose of *abstract index manipulations*. All the work with data objects with indices is carried out using, essentially, explicit sets of their components specified with respect to a definite gauge.

3 Language of problem specifications

Let us outline now the GRC_{EC} input interface as an element which often crucially affects the general estimation of a system by a user.

The exhaustive description of the GRC_{EC} language occupies scores of pages of the Manual but we hope to exhibit its main features by means of the commenting on a number of examples of the typical GRC_{EC} programs (the term *problem specification* is probably more adequate here and will be usually used below instead). The first example is the following.

Example 1.

```
1 Problem First_test.
2 >>slang<<  %> <-This is the file inset record<%
3 Data:
4     declare COORDINATES x,y,zeta,zeta~;
5     ABBREVIATIONS are c=(1+I*SQRT(3))/2,
6                       c~=(1-I*SQRT(3))/2;
7     TETRAD elements are
8       T0=d x + I*E**(-x) d y,
9       T1=d x - I*E**(-x) d y,
10      T2= E**(c*x) d zeta + I*E**(c~*x) d zeta~,
11      T3= E**(c*x) d zeta - I*E**(c~*x) d zeta~;
12 end of data.
13 Instructions:
14    find SPINOR CURVATURE;
15    obtain and type VACUUM EINSTEIN EQUATIONS;
16    classify UNDOTTED WEYL SPINOR;
17    stop;
18 end of instructions.
19 Run!
```

(*Remark*: The enumeration on the left is not a part of the code and was introduced for the sake of the reference convenience alone).

The above example illustrates a typical property of the GRC_{EC} programs: a specialist in the gravity theory is usually able to comprehend its purport without additional elucidations — or, at worst, with the help of minimal ones. Of course, there are some designations such as 'SPINOR CURVATURE', 'VACUUM EINSTEIN EQUATIONS', etc., whose precise meaning and *concrete forms of representation* are not automatically manifest (although, in principle, these expressions are often referred to in the literature concerning the physical field in question). However this is in fact such a *special terminology* which is a compulsory element of any more or less narrow scientific field and which is to be explained in a Manual or is handled by means of a sort of the 'HELP' facility.

One can see that the above problem specification is given in a form simulating the sequence of imperative sentences in English incorporated with a number of records coding the mathematical expressions. It obviously comprises *the specification of the initial data* (lines 3-12) and *the indication what results have to be obtained* (lines 13-18), given in the form of *instructions*.

In turn, the coding of the mathematical expressions displayed in example 1 shows agreement with the majority of computer algebra systems (and originated mostly from *Algol*). In particular it is rather close to the corresponding *Reduce*notations. At the same time they are enhanced

to provide an efficient and straightforward exterior forms representation. (The features of *the Excalc* subpackage of *Reduce* dealing with exterior forms are not, strictly speaking, supported.) The practice amassed gives evidence that the approach realized is efficient and natural in the present framework.

In particular, the 'in line' record of the following tetrad of 1-forms

$$\theta^0 = dx + ie^{-x}dy, \quad \theta^1 = dx - ie^{-x}dy,$$
$$\theta^2 = e^{c\,x}d\zeta + ie^{\bar{c}\,x}d\bar{\zeta}, \quad \theta^3 = e^{c\,x}d\zeta - ie^{\bar{c}\,x}d\bar{\zeta}.$$

can be easily revealed in the lines 8-11 of example 1. Here $x, y, \zeta, \bar{\zeta}$ are the coordinates, $c = \frac{1}{2}(1 + i\sqrt{3}), \bar{c} = \frac{1}{2}(1 - i\sqrt{3})$ are the abbreviated notations playing here the role of symbolic constants. (These data correspond to the exact solution of the vacuum Einstein equations described by the metric $g = 2\,\theta^0 \cdot \theta^1 + 2\,\theta^2 \cdot \theta^3$, see Ref. [8], Eq. (10.14)).

The processing which has to be carried out with the data specified is described in lines 14-16 of the example text, their purport requiring probably no separate comments. It seems worth noting only that each of these three brief *instructions* invokes automatically a collection of sophisticated routines performing all the work and yielding an immediate result. It should be emphasized that a user must *not* specify any formulae, describe the methods of the treating the relevant equations, etc. He or she simply indicates *what* is to be obtained and then draws an answer[2].

Let us outline now a deeper level of the $\mathrm{GRC}_{\mathrm{EC}}$ language and discuss the main features of its *syntax*, adding in appropriate cases the comments on the *semantic* as well.

Although a $\mathrm{GRC}_{\mathrm{EC}}$' problem specification usually looks like a free description, it actually exhibits a strict syntax structure. In particular each problem specification comprises

- the *title* (line 1 in example 1), which specifies the *problem name* (the string 'First_test'),
- the *starting thrust* 'RUN!'[3] (the last line) and
- a collection of *sections*.

There are two sections (*apparent* ones, there may also exist a *hidden* section) in example 1: the section of the initial DATA (lines 3-12) and the section of INSTRUCTIONS to be executed (lines 13-18).

In its turn every section is identified by the *heading* (lines 3,13) and the *conclusion* (lines 12,18, respectively). A section body confined between them contains a number of *paragraphs*, each of them being finished by a semicolon[4]. In example 1 each paragraph in the sections of DATA and INSTRUCTIONS is disposed in a separate line. This is not necessary however and the admissible format of the problem specification is free.

The set of the types of sections supported is fixed. Two of them (DATA and INSTRUCTIONS) have been mentioned above, and the others will be briefly characterized below.

The section of REGIME SPECIFICATION controls some general global options. The following example of the section record

```
Regime specification:
 set CLASSICAL FORMALISM;
 DIMENSION is 5;
end of specification.
```

contains the *two paragraphs*. They entail the picking out the standard formalism of the Riemannian geometry and fix the space-time dimension to the value 5 (which corresponds to the Kaluza-Klein type theories).

Further, the problem specification may involve a section of SUBSTITUTIONS whose each paragraph describes certain *substitution rules*. The section represents an element of the *pattern matching* control facility of $\mathrm{GRC}_{\mathrm{EC}}$ and provides the corresponding data to the system. An example of the section of SUBSTITUTIONS with two substitution rules is the following:

```
Substitutions:
(1) SIN(theta)**2=1-COS(theta)**2;
(2) DF(rho,r)=-rho**2;
end of substitutions.
```

(It is worth noting that the second substitution rule represents in fact the differential equation $d\rho/dr = -\rho^2$, where ρ (rho) is so-called SCALAR, see section 5 below.)

These substitution rules *are activated* by issuing the instruction

```
Excite substitutions (1),(2);
```

which refers to the enumeration distributed over the section of 'SUBSTITUTIONS'. An arbitrary subset of the substitution rules stockpiled may be simultaneously activated. Subsequently, some of them may be cancelled out by means of an instruction with the action 'ABOLISH' supporting a similar syntax.

[Another, 'dynamic', way of implementing a substitution rule is also provided for. Specifically, one may issue, for example, the instruction

```
Let i*a*rho*C.C.(rho)=(C.C.(rho)-rho)/2;
```

(which means the replacing $i\,a\,\rho\,\bar{\rho} \to \frac{1}{2}(\bar{\rho} - \rho)$). Then the new paragraph

```
(3) i*a*rho*C.C.(rho)=(C.C.(rho)-rho)/2;
```

(automatically labelled by the next ordinal number) is added to the section of SUBSTITUTIONS (which is created in the case of its initial absence) and the instruction 'EXCITE SUBSTITUTION (3)' is automatically issued.]

The next section of SAMPLES FOR COMPARISON comprises the mathematical expressions written down 'by hand' in the form of assignments. They are intended for the comparisons with the results produced by $\mathrm{GRC}_{\mathrm{EC}}$ itself. Additionally, the replacing of the values generated by the system by equivalent but, generically, non-identical 'samples' may be carried out.

The syntax of the paragraphs of the 'SAMPLES FOR COMPARISON' and 'DATA' sections essentially coincide but only *working assignments* (lines 7-11 in example 1) rather than *declarations* (line 4, respectively) or *foremost assignments* (lines 5-6) are here allowed. In the case of the example 1, the record

```
Samples for comparison:
 VOLUME element is
   VOL = -4*I d x /\ d y /\ d zeta /\ d zeta~;
end of samples list.
```

might be added to the problem specification. Then the instruction

```
Compare VOLUME with sample;
```

makes the system determine whether *the volume element*, possessing *the export name* 'VOLUME' (and defined in our case as $i\,\theta^0 \wedge \theta^1 \wedge \theta^2 \wedge \theta^3$), actually coincides with the expression shown above (specifically, $-4i\,dx \wedge dy \wedge d\zeta \wedge d\bar{\zeta}$, in the standard notation).

[2] Of course, solving of a more complicated problem would require a more elaborated control over the system run.

[3] The usage of the upper and lower case letters in keywords will be discussed below.

[4] There is an exception: in the case of a shortened form of the instruction with the action 'TYPE' the question sign '?' is used instead; see example 2 below.

Practice gives evidence for the extreme usefulness of the above tool. It is especially convenient to use as samples the results of the preparatory system runs; this allows one to bring the data to be processed to the optimal form (perhaps with the help of some re-casting 'by hand') that is often difficult to achieve due to the autonomous, by default, fashion of the mathematical transformations brought about by the system simplifier.

The content of the 'NOTATIONS' section ensures the possibility to modify so-called *kernel identifiers* of the data objects. We have seen an instant of the kernel identifier in example 1: that is the one-symbol token 'T' which is used in lines 8-11 for the constructing the *component identifiers* T0, T1, T2, T3 constituting together the data object 'TETRAD'. Adding the record

```
Notation:
  Theta denotes TETRAD;
end of notations.
```

to the problem specification, one may then refer to the TETRAD elements by means of the new component identifiers Theta0, Theta1,... instead of T0, T1, etc.[5]

An important auxiliary tool of *the supplementary terminology adjusting* utilizes the content of so-called SLANG (or JARGON) section. It finds a lot of important applications in practice by virtue of its special role in simulating natural language in the system.

SLANG section contains a number of records similar to the following ones:

```
Notation & NOTATIONS, notations & NOTATIONS;
           are synonymous;
end & END, of & OF, denotes & DENOTE
           are synonymous;
```

They establish so called *restricted synonymy relations* between the tokens from each pair[6]. Moreover, one may associate a single term with an entire *words sequence*. For example, the following statements

```
OBTAIN AND TYPE & O.and.t,
VACUUM EINSTEIN EQUATIONS & V.E.EQ
                                  are synonymous;
```

allows one to shorten the lengthy instruction 'OBTAIN AND TYPE VACUUM EINSTEIN EQUATIONS' to 'O.and.t V.E.EQ'.

Next, the supplementary terminology adjusting allows one to use the lower case or mixed versions of the keywords (the system core assumes just the usage of the *upper case* alphabet in all the keywords). The corresponding collection of the synonymy assignments is contained in a file supplied with the system.

Specifically, the record '>>slang<<' in line 2 of example 1 represents so-called *file inset*. Its function is to insert the content of the file whose the name is pointed out (here 'slang') in the corresponding point of the text of problem specification (whereas the very record of the file inset is removed)[7]. The record '>>slang<<' just ensures the implementation of the additional lower case and mixed versions of the keywords eliciting them from the file 'slang'.

In example 1 all the 'common' words therein involving the lower case letters, except for the problem name 'First_test', the content of the *long comment* '%> <-This is ... <%', (discriminated by the separators '%>' and '<%'), and the mathematical formulae, are not the 'true' keywords but are connected to the latter by means of the restricted synonymy relations provided by the standard SLANG description[8].

We have listed all the top level syntax structures provided for the problem specifications in framework of the GRG_{EC} input language. Our examples give evidence for the sufficient clarity, flexibility and naturalness of the resulting records. In most cases their comprehension is immediate and does not requires an experience in the programming as such. On the other hand the GRG_{EC} language is sufficiently strict to prevent any ambiguity and pernicious confusions.

4 Run control

The next topic worth discussing concerns with the tools provided for the control over the GRG_{EC} run. *The instructions* immediately serve that purpose and example 1 involves four of them (in lines 14-17). These refer, in turn, to five names of the *primitive actions*: 'FIND', 'OBTAIN', 'TYPE', 'CLASSIFY' and 'STOP'.

For the sake of the better comprehending the role of the instructions facility let us consider the following more demonstrative example.

Example 2.

```
1 Problem Schwarzschild_vacuum.
2 >>slang<<
3  Data:
4    declare COORDINATES u,r,theta,phi;
5    declare CONSTANT m;
6    TETRAD is T0=r*(d theta + I*SIN(theta) d phi),
7              T1=C.C.(T0),
8              T2=d u, T3=2 d r -(1-2*m/r)d u;
9  end of data.
10 Instructions:
11  find and type S-FORMS,CONNECTION ↓
                                 ↑and SPINOR CURVATURE;
12  unload;
13 comment: the current state has been is saved↓
                                      ↑on a hard disk;
14  erase RICCI SPINOR;
15  calculate RICCI SPINOR from DOTTED CURVATURE↓
                                     ↑and type it;
16  erase CONNECTION and DIFFERENTIALS OF TETRAD;
17  find CONNECTION from DIFFERENTIALS OF S-FORMS↓
                                     ↑and type it;
18  save and erase DIFFERENTIALS OF S-FORMS↓
                                   ↑and CONNECTION;
19  find,type and erase CONNECTION;
20  find CONNECTION by standard way and type it;
21  erase CURVATURE,DIFFERENTIALS OF TETRAD and↓
                          ↑DIFFERENTIALS OF S-FORMS;
22  calculate CURVATURE from spinor components↓
                                    ↑ and type it;
23  find and type WEYL INVARIANTS;
24  restore DIFFERENTIALS OF TETRAD and ↓
                          ↑DIFFERENTIALS OF S-FORMS;
25  obtain and type VACUUM EINSTEIN EQUATIONS;
26  ALL KNOWN?
27 pause;
28  empty BOX;
29  save DIFFERENTIALS OF S-FORMS,VOLUME;
30  erase ALL KNOWN;
31  type ALL SAVED using data saved;
32  stop;
33 end of instructions.
34 Run!
```

Note that the numerous instructions in the above example (each of them occupies a separate line with numbers from 11 to 32) do not follow any sophisticated idea and serve

[5] A distinct, so-called *universal* syntax rule is additionally provided for the coding the indexed components. It assumes the separation of the kernel names and the indices by the vertical bar characters, for example, Theta|0, T|0, RICCI|1|1~, etc.

[6] More precisely, *the stars* of synonymy links are created (or, if necessary, destroyed).

[7] Not only a single file name but a sequence of them might be put between the *file inset separators* '>>' and '<<', the corresponding path (or paths) being specified separately.

[8] Our out-of-line instances involve the records utilizing the both upper case and lower case symbols (that is usually done in a practice), whereas all the keywords in the main text are given using the upper case characters alone.

mostly for the illustrating sake. They exhibit the main rules for the construction of the *instruction stream*.

First of all, it has to be noted that GRC_{EC} does not support such control facilities as *loops, if-then-else* statements, *goto* operators, subroutines, etc. Essentially, these are superfluous here. The instructions are merely performed one by one, successively.

The executing of the instruction stream may however be interrupted. The point is that the most preferable fashion of the GRC_{EC} run is the so-called *quasi-batch* mode when all the data and a majority of instructions are prepared preliminarily in the form of a file containing the text of the corresponding problem specification. Its processing starts in a batch fashion but the possibility to intervene the calculation is provided for. [The 'purely batch' and 'purely interactive' modes are also available.]

Return to example 2. The instruction 'PAUSE' (line 27) causes just a transfer of the control to the user's terminal. Another natural takeover of a control occurs after the instruction stream is exhausted (in particular, it is allowed to be empty from the very beginning if the 'INSTRUCTIONS' section is absent in the problem specification at all).

Having received a control, a user may issue the next instruction (or instructions) from the terminal, typing them on a keyboard in exactly the same form as they would be made out to the section of INSTRUCTIONS. Besides, using the file inset facility, a collection of the instructions, prepared previously in the form of a file, may be invoked. Further the control can be returned to the instruction stream, skipping preliminary a number of the current items, if necessary.

After these general remarks, let us comment on the instances of instructions displayed in example 2.

Each instruction record begins at an *action name*. These are 'FIND AND TYPE' in line 11, 'UNLOAD' in line 12, 'COMMENT' in line 13, etc. [There is an exception from the above rule: the record 'ALL KNOWN?' (line 26) is the *the shortened form* of the 'TYPE ALL KNOWN' instruction. Here the (implicit) action name is 'TYPE', 'ALL KNOWN' is the data object name — the action *target*.] The second and third cases refer to *primitive* (i.e. indivisible) actions (similarly for the actions displayed in lines 14, 21, 24, etc.), whereas the first one represents the name of a *compound action*. There are other compound actions in example 2: 'FIND, TYPE AND ERASE' (line 19), 'OBTAIN AND TYPE' (line 25) and others.

Further, an action — primitive or compound — can be either

directed one, i.e. applied to some data object — a target (an example: the action 'ERASE' in line 16 is applied to the data objects 'CONNECTION' and 'DIFFERENTIALS OF TETRAD'),

or be able to receive some *parameter*(s); then it is called a *parameterized action* (examples: the action 'COMMENT' in line 13, the parameter is the sequence ': the system state ...'; the action 'EMPTY' in line 28, the parameter is 'BOX'),

or admit no target and parameters (examples: the action 'UNLOAD' in line 12; the action 'PAUSE' in line 27; the action 'STOP' in line 32).

Besides a target, a directed action may admit the parameters as well (examples: the action 'CALCULATE' in line 15, here the target is 'RICCI SPINOR', the parameter is 'FROM UNDOTTED CURVATURE'; the action 'TYPE' in line 31, here the target is 'ALL SAVED', the parameter is 'USING DATA SAVED').

Executing instruction with compound action, each its primitive constituent is successively applied to the common target, if any, one after another.

Next possibility provided for the instruction constructing is the so-called *extra action* which can be seen in lines 15, 17, 20, 22. In all these cases the extra action name is 'TYPE'. Whereas the main action name is placed at the beginning of the instruction record, the extra action name stands at its end and is separated from the left by 'AND' and from the right by 'IT' or 'THEM'[9].

A reason of the introducing the extra action facility is the following. It makes sense to use it if the action in an instruction consists of, say, two primitive actions (e.g. 'FIND' and 'TYPE' for the instruction in line 15), and the first of them requires a parameter ('FROM DIFFERENTIALS OF S-FORMS') whereas the second action does not (or requires a different parameter). Using an extra action, it is easy to forward a proper parameter to a desirable action. [For the instruction example considered, the equivalent result is achieved by the following *two* subsequent instructions:

```
Calculate RICCI SPINOR from UNDOTTED CURVATURE;
type RICCI SPINOR;
```

Incidentally, the second line may also be shortened to 'RICCI SPINOR?'.]

Now let us discuss the notion of *data object*. GRC_{EC} supports the two their kinds: the *primitive* data objects and *the composites*. Primitive data objects constitute the basic class and each composite is simply a (predefined) collection of the primitive elements. Of course, the constituents of every composite were selected not spontaneously but fitting for the geometrical or physical meaning of the primitive elements involved in it.

From a viewpoint of the GRC_{EC} language, a primitive data object is regarded as an indivisible entity although it may actually consist of a number of *components* accessible in principle via *the component identifiers*. This models the relations taking place between the corresponding mathematical or physical origins of the notions in question.

For example, the data object 'S-FORMS, CONNECTION AND SPINOR CURVATURE', referred to in line 11 (regarded as a constituent of an instruction, it is called a *compound target*), comprises just the three composites. In their turn

```
   S-FORMS comprise {UNDOTTED S-FORMS,
                     DOTTED S-FORMS},
CONNECTION comprises {UNDOTTED CONNECTION,
                      DOTTED CONNECTION},
    SPINOR comprises {UNDOTTED WEYL SPINOR,
 CURVATURE            DOTTED WEYL SPINOR, RICCI
                      SPINOR, SCALAR CURVATURE}.
```

All the data objects on the right are already primitive (but each comprises several components except SCALAR CURVATURE).

Next, let us consider line 12. The action 'UNLOAD' realizes the complete *unloading the problem state* into a disk file. All the data are written in the intrinsic representation that allows one further (during another session) to quickly *load* that information, putting the system immediately to the state in which it was at the moment of the UNLOADing.

The mentioned facility provides a simple and efficient tool for the maintaining a *database* which accumulates the results of the past calculations (in particular, elaborated ones made with the help of an eminently powerful computer). For example, the archives of the various characteristics of the physical fields (i.e. the exact solutions of the field equations) can be realized. Besides the results, such

[9] If there exists a parameter affected the extra action then it (the parameter) is placed at the very end of the instruction record.

a database ensures the very 'programs' (problem specifications) which automatically generate them. Moreover, the further calculations and transformations of data can be carried out.

Let us proceed with the example consideration. Line 13 is simply a commenting instruction. The instruction in line 14 ERASEs the value of RICCI SPINOR while the instruction in line 15 calculates it again but *by another*, non-standard way FROM DOTTED CURVATURE (by default, calculation of RICCI SPINOR FROM UNDOTTED CURVATURE is assumed). Additionally, the result CALCULATEd is TYPEd on the terminal screen[10].

In line 18, the action 'SAVE' sends the values of the data objects 'DIFFERENTIALS OF S-FORMS' and 'CONNECTION' to the temporary depository — the BOX, a file on a hard disk — and then they are removed from RAM by means of the action 'ERASE'. (One of these data objects will be further RESTOREd by means of the instruction displayed in line 24).

The next noteworthy remark concerns the line number 20. By default, GRG_{EC} tries by to minimize the work attempting to FIND (here, essentially, to calculate) the requested data objects from the data which is *already known*. Correspondingly, it may choose, if possible, the ways of calculations, deriving, for example, CONNECTION either FROM DIFFERENTIALS OF TETRAD or FROM DIFFERENTIALS OF S - FORMS depending on what of these data is currently available. The option 'BY STANDARD WAY' forbids such a 'free' behaviour and forces the system to follow the primary calculation method.

A useful example concerning the collective data addressing is given in line 26. The data object named 'ALL KNOWN' is a representative of the class of so-called *unfixed* data objects whose content depends on the state of environment. The name 'ALL KNOWN' refers to all the ordinary ('fixed') data objects which are known at the moment when the instruction is executed. Another unfixed data object is referred to in line 31. Of course, 'ALL SAVED' comprises all the 'fixed' data objects whose values have been SAVEd in the BOX[11].

Line 26 also exhibits an example of the shortened form of the instruction with the action 'TYPE'.

The implications of the 'PAUSE' action (line 27) have been already mentioned: it transfer the control to the user's terminal allowing him or her to issue further instructions from the keyboard (or invoke them from a file using a file inset facility).

Finally, the parameter 'USING DATA SAVED' referred to in line 31 means the following:

1. if the requested data object has no value at the moment it is RESTOREd from the BOX;
2. the action (in our example, to TYPE) is applied to it;
3. the value previously RESTOREd is again ERASEd and the system returns to the state existed before the execution of the instruction.

One can see that the above abbreviated reference 'USING DATA SAVED' to the algorithm described fairly well characterizes its essence.

In a more general context, this is just the feature which the language of the system control have to exhibit; its designing was permanently pursuing such a goal at least.

[10] and, by default, is copied to the *hardcopy file*

[11] 'EMPTY BOX' instruction (line 28) just removes all the BOX content.

5 Elements of design solutions

Similarly to any sufficiently complex software, GRG_{EC} involves a lot of algorithms playing important role but only few of them are worth, indeed, a separate discussion. We outline here a selection of topics concerning GRG_{EC} design which possibly could be of a certain common interest.

It has been mentioned that GRG_{EC} borrows the *Reduce*'s algebraic processor capabilities to simplify the general mathematical expressions, it does not carry out any simplifications itself. However a number of routines has been developed which might be regarded as complements of *Reduce* enabling it to handle some specific data supported by GRG_{EC}.

For example, a routine which ensures the handling of the complex (as well as real or pure imaginary) variables and functions has been developed. Following the corresponding conventions, it may be implemented as a *Reduce*'s subpackage and is able in principle to function independently upon GRG_{EC}. The facility in question realizes, for example, the transformation

```
IM((u+I*v)/(k**7*k~**3*a)) ⇒
          4          4          7   7
 - I*(RE(k )*u + IM(k )*v)/(a*k *k~ )
```

and is able to properly comprehend the derivative

```
DF(RE(rho~),rho) = 1/2
```

(*Reduce* itself yields 0 here[12]), where u, v are DECLAREd REAL, a is DECLAREd pure IMAGINARY, and k & k~, rho & rho~ are DECLAREd COMPLEX CONJUGATED; 'RE' and 'IM' are of course the *real* and *imaginary part* operators.

An extremely useful and efficient original tool implemented in GRG_{EC} is the so-called 'SCALARS' facility. (The term 'SCALARS' should be mixed up neither with the own *Reduce*'s notion of scalars nor with its independent physical regarding originated from the field theory). It often noticeably speeds up calculations and in certain cases provides the possibility of some data handling which the standard *Reduce* is unable to carry out (without additional elaborate programming at least).

A SCALAR is the data structure which intrinsically incorporates the two distinct data types: a *variable* and an *unspecified function*. [To be more precise, either SCALAR-function or its derivatives (all or several of them) may possess a concrete analytical representations which are properly utilized (in particular, are automatically substituted in the relevant situations).]

The SCALARS were introduced for the implementation of the *chain derivative rule* which is hard to realize within the basic *Reduce*'s framework; further they gave rise to the other advanced applications.

In order to demonstrate the way of the SCALARS treatment let us resort to a manifest example and

```
DECLARE SCALARS U(x),V(y),W(x,y,V,U);
```

Here where x, y are the (previously DECLAREd) COORDINATES (i.e. independent variables).

Then the differential D F(W) (for DECLAREd but unspecified FUNCTION F) is *automatically* expanded to

```
(DF(F(W),W)*DF(W:(x,y,V,U),x)
 + DF(F(W),W)*DF(W:(x,y,V,U),U)*DF(U:(x),x)) d x+
(DF(F(W),W)*DF(W:(x,y,V,U),y)
 + DF(F(W),W)*DF(W:(x,y,V,U),V)*DF(V:(y),y)) d y
```

One can see that the relevant derivatives are constructed here just in accordance with the chain differentiation rule. "Janus-like faces" characteristic of SCALARS are explicitly manifested here: each SCALAR essentially possesses the two

[12] Certain problems concerning with a proper handling the derivatives of RE's, IM's, C.C.'s, etc. were a primary motive for the developing the routine in question.

representatives, a variable and a function; these are the pairs `U` and `U:(x)`, `V` and `V:(y)`, `W` and `W:(x,y,V,U)`, respectively. Specifically, under the action of the derivative operator the 'expanded' expression of the `SCALAR` with explicit arguments is used while in other cases a (distinct) separate identifier is involved.

A distinguishing feature of the `SCALAR` data type is the strictly *tree-like* structure of the variable dependencies implied, any loops being forbidden. Associating the *branchings* with `SCALARS`, all the *leaves* are claimed to be `COORDINATES`. Obeying these conditions, the tree of the mutual `SCALARS` dependencies may be arbitrarily complex.

It can be seen that the 'SCALAR' data type suffices for the most applications when it is necessary to handle a ramified dependencies between a collection of variables.

The `SCALARS` were not incorporated with *Reduce*' algebraic processor. This facility functions as a separate collection of routines. Its implementation is not straightforward, the main problem being as follows. For every mathematical expression to be differentiated, it is necessary to know what variables belonging to the classes of `COORDINATES` and `SCALARS` it actually depends on, both *explicitly* (for the dependence upon `COORDINATES` and `SCALARS`), and *in total* (for the dependence upon `COORDINATES`)[13]. Thus the *dependence list* of the mathematical expression to be differentiated has to be preliminarily found.

The determination of the dependence list is realized by means of a twofold trick. At first, every mathematical expression (a function) is being associated with the *overestimating collection* of the dependence lists. The latter is formed by means of joining the dependence lists of the original expression constituents at the moment of the expression construction. Obviously, the dependence of the resulting expression (i.e. the collection of `COORDINATES` and `SCALARS` which are actually involved in the expression) may not exceed the joint dependence of its constituents. On the other hand, after simplification, the dependence of the mathematical expression *may reduce*, so joining the dependencies of the constituents yields generically the *overestimating* rather than actual dependence of the whole.

At second, the routine providing determination of the *actual dependence* lists was developed. They are constructed by means of *exhausting* the mentioned overestimating dependencies.

In principle, the determination of the dependence of a mathematical expression upon the variables belonging to a certain class is realized by means of its recursive scanning and collecting the relevant variables. In addition, let a list of the variables *yet not found* be supported. So, whenever the dependence upon a new variable is detected, this variable is added to the relevant dependence list and is removed from the list of variables yet not found. Then, starting with the (supposedly) overestimating dependence list mentioned above, the procedure may stop as soon as the list of variables yet not found *is exhausted*. In a majority of generic situations, this trick allows one to avoid scanning of the whole mathematical expression and noticeably saves time spent on determining the dependence lists.

In any case, practice shows that for non-trivial calculations this time expenditure proves to be negligible comparatively with the other expenditures, especially time spent on simplifications.

Let us give here a simple example which demonstrates more clearly a potential power of `SCALARS`. Given the following collection of declarations and foremost assignments

```
Declare COORDINATES r,th,ph,t;
declare SCALARS x,y,z(r,th,ph),
                W,Wx,Wy,Wz,Wt,
                Wxx,Wyy,Wzz,Wtt(x,y,z,t);
SCALAR VALUES are z=r*COS(th),
     x=r*SIN(th)*COS(ph), y=r*SIN(th)*SIN(ph);
SCALAR DERIVATIVES are DF(W,x)=Wx, DF(W,y)=Wy,
                       DF(W,z)=Wz, DF(W,t)=Wt,
                  DF(Wx,x)=Wxx, DF(Wy,y)=Wyy,
                  DF(Wz,z)=Wzz, DF(Wt,t)=Wtt;
```

the instruction

```
Evaluate I d (# d W)/VOL;
```

(here '#' denotes the Hodge star operation, flat 4-dimensional Minkowski metric is assumed) automatically yields the answer

```
- Wxx - Wyy - Wzz + Wtt
```

i.e. the D'Alambertian $(\partial_{tt} - \partial_{xx} - \partial_{yy} - \partial_{zz})W(x,y,z,t)$. [The partial derivatives have been *condensed* here to the variables (which belong to the class of `SCALARS`) and this ability usually accelerates calculations and makes the results more manifest.] Since x, y, z are, essentially, the functions of the spherical coordinates, W is a 'function of functions' in fact. Hence in the above example the derivatives *with respect to functions* are calculated.

Above, we have used some records denoting differential (exterior) forms. Let us now outline the basic features of the implementation of this important mathematical notion in $\mathrm{GRG}_{\mathrm{EC}}$.

The exterior calculus subpackage is a collection of routines, more or less independent, which only exploits the algebraic processor of *Reduce* for the simplification of general mathematical expressions. One of its important underlying elements is the intrinsic representation of the relevant data which was specially designed to minimize the specific overheads connected with the basic operations with exterior forms: summation, exterior multiplication, exterior derivation, etc. Not going into the full details here, we would like to give some examples only.

In particular, the 1-form

```
z d th + I*z*SIN(th) d ph
```

(see declarations of `z`,`th`,`ph`, etc. in the above example), consisting of the two terms, is intrinsically represented by the following bracket structure:

```
( ((z ((r th) z)) 4 (NIL . T)(T . T))
  (((TIMES I z (SIN th))((r th) z))↓
                    ↑8 (NIL . T)(NIL . T)(T . T)) )
```

(standard LISP notations are used). Here `(z((r th)z))` and `((TIMES I z (SIN th))((r th) z))` are the so-called *algebraic lists*. Their `CAR` elements, `z` $\sim z$ and `(TIMES I z (SIN th))` $\sim i\,z\,\sin\theta$, respectively, are the very functions (coefficients in the form expansion with respect to the exterior products of the differentials of coordinates or the elements of the other basis). Next, `CDR`s `(((r th) z))` (here coinciding for the both terms) are the lists of *the short lists of the overestimating dependences* of the corresponding functions. In our case, they both consist of the only element `((r th) z)`. This record means that the corresponding expressions (coding the functions r and $i\,z\,\sin\theta$) depend at most on the `COORDINATES` `r` and `th` *in total*, i.e. both explicitly and via the possible dependence on `SCALARS` which ultimately depend on `COORDINATES`, possibly, via the chain of other `SCALARS`. Removing the first element, the list `(z)` means that the functions may *explicitly* depend on the `SCALAR` `z` alone.

These data, characterizing the dependencies of mathematical expressions, is substantial for a constructing the derivatives taking into account the mutual connections of

[13] Thus the *three* lists of variables are to be actually supported.

SCALARS and COORDINATES and a proper applying the chain derivative rule.

Next, CDRs of the elements describing the terms, namely, (4 (NIL . T)(T . T)) and (8 (NIL . T)(NIL . T) (T . T)), encode the so-called '*tails*' $d\theta$ and $d\phi$, respectively. In general, a 'tail' of an exterior M-form is a product $dx^{\alpha_1} \wedge \ldots dx^{\alpha_j} \wedge \ldots dx^{\alpha_M}$ in holonomic mode or $b^{a_1} \wedge \ldots b^{a_j} \wedge \ldots b^{a_M}$ in anholonomic mode (b^a is the covector basis).

The *exponentiated ordinal numbers* in the beginning of the 'tail' lists equal $\sum 2^{\alpha_j}$ or $\sum 2^{a_j}$, where α_j or a_j are, for holonomic mode, the ordinal numbers of the coordinates whose differentials are involved in the 'tail' (α_j) or, for anholonomic mode, the ordinal numbers of the cobasis elements (a_j), respectively. In our case the COORDINATE th possesses the ordinal number 2 and COORDINATE phi possesses the ordinal number 3. Hence one obtains 4 and 8, respectively (the 'tail' $d\theta \wedge d\phi$ corresponds to the exponentiated ordinal number $4 + 8 = 12$).

Finally, the lists ((NIL . T)(T . T)) and ((NIL . T) (NIL . T)(T . T)) represent the *bitmaps* of the 'tails' elements and their *left parities*. In order to make this more transparent, let us re-cast these lists of pairs into the two sequences of CARs and CDRs of the pairs:

$\{\{\circ\ \times\}\,\{even\ even\}\}$

$\{\{\circ\ \circ\ \times\}\,\{even\ even\ even\}\}$

Here 'o'($\sim$NIL) denotes the absence while '$\times$'($\sim$T) the presence of the differential dx^α (the cobasis element b^a) in the sequence $dx^1, dx^2, \ldots$ ($b^1, b^2, \ldots$, respectively) which are involved in the 'tail', the mark '*even*'$\sim$NIL denotes the parity of the numbers of elements of the 'tail' on the left to that point ('*odd*'$\sim$T). For example, the 'tail' $d\theta \wedge d\phi$ is coded as

((NIL . T)(T . T)(T . NIL)) $\sim$

$\{\{\circ\ \times\ \times\}\,\{even\ even\ odd\}\}$.

The outlined structure comprises the minimal information necessary for the immediate constructing the results of summation, exterior multiplication and exterior derivative on exterior forms. It allows one to realize the exterior calculus in an efficient and straightforward way. However only the explicit expansions with respect to some basis can be efficiently handled.

6 Performance estimate and concluding remarks

GRC_{EC} system makes it possible to formulate physical problems and generate their solutions with the help of a simple language succeeding even in comparatively complicated cases.

A majority of more or less complicated problems is too bulky to be satisfactorily described in a section of a journal article. There is however an example which can be discussed here in brief and which gives some evidence of the above property, at least within certain limits. This is the calculation of the curvature of the so-called Bondi' metric (see Refs. [9]).

To that end, let us consider the following fragments of the GRC_{EC} output

Example 3.

```
..... Some auxiliary messages are skipped.....
Problem Bondi_metric.
 DATA :
  declare COORDINATES u, r, theta, phi;
  declare SCALARS U, V, B, G(u, r, theta);
  TETRAD is T2 = -(E ** B) d u,
            T3 =(E ** B) *(d r + V d u / r),
            T0 = r *(-(U * E ** G) d u ↓
                                  ↑+(E ** G) d theta
                 +(I * SIN(theta)) d phi ↓
                             ↑/ E ** G)/ SQRT (2),
            T1 = r *(-(U * E ** G) d u ↓
                                  ↑+(E ** G) d theta
                 -(I * SIN(theta)) d phi ↓
                             ↑/ E ** G)/ SQRT (2);
  END OF DATA.
  INSTRUCTIONS :
    show time;
    find UNDOTTED SPINOR CURVATURE;
    type RICCI00~, UWEYL0;
    quit;
  END OF INSTRUCTIONS.
RUN!
.....Some output is skipped.....
 Time is 1.6 s.
==> find UNDOTTED SPINOR CURVATURE
--> find UNDOTTED SPINOR CURVATURE
SCALAR CURVATURE is not found in DATA section
::::>SCALAR CURVATURE has been calculated↓↔↓5.4 s.
RICCI SPINOR is not found in DATA section
::::>RICCI SPINOR has been calculated   ↓↔↓6.8 s.
UNDOTTED WEYL SPINOR is not found in DATA section
::::>UNDOTTED WEYL SPINOR ↓
                    ↑has been calculated   ↓↔↓7.8 s.
==> type RICCI00~, UWEYL0
RICCI SPINOR Component:
                ↓
RICCI  .    = ↓
   00       ↓
   ↑                                              2
   ↑2*DF(B:(u,r,theta),r) - DF(G:(u,r,theta),r) *r
   ↑-----------------------------------------------
   ↑                        2*B
   ↑                       E   *r
UNDOTTED WEYL SPINOR Component:
UWEYL  = ↓
     0   ↓
    ↑(2*DF(B:(u,r,theta),r)*DF(G:(u,r,theta),r)*r
↓↔↓   - DF(G:(u,r,theta),r,2)*r↓
                        ↑ - 2*DF(G:(u,r,theta),r))/
  2*B
(E   *r)
==> quit
```

Certainly, the above output does not require lengthy elucidations that is itself a good argument in favour of the approach realized.

The resulting simple equations described in example 3 can be rewritten in a manifest usual form as

$$e^{2B}\Phi_{0\dot{0}} = -G_r + \frac{2}{r}B_r,\ e^{2B}\Psi_0 = -G_{rr} - \frac{2}{r}G_r + B_rG_r.$$

They play an important role in the theory of gravitational radiation [9]. The intermediate calculations leading to them are however rather involved. As far as we know, the equivalent result required about a half a year long calculation 'by hand'.

It is worth noting that the equivalent problem (calculations in Bondi' metric) traditionally served as one of the tests of the performance rate for computer algebra systems dealing with the gravitation theory (cf. Ref [12]). In frames of GRC_{EC}, it corresponds to the calculation of the curvature 2-forms. The above test yields the estimate of the relevant time expenditure; it amounts to $5.4 - 1.6 = 3.8$ seconds (see the time labels in example 3). It should be mentioned that the calculation involves here the determination of the SCALAR CURVATURE besides the curvature 2-forms. The hardware characteristic are: IBM PC AT compatible computer with Am486DX4-S 120 MHz CPU (8 KB+256 KB cache), PCI BUS, 8 Mb RAM. A reference point useful for comparisons is provided by the time of the expanding (x+y+z)**100: *Reduce* consumes for this 18.6 seconds.

A comparison test with the straightforward *Excalc* code (written by V.V. Zhytnikov) which realizes the equivalent mathematical background may be of interest. The determination of the curvature 2-forms for the Bondi' metric requires about 19 seconds, although it should be borne in mind

that about half of the work is superfluous here: roughly speaking, the relevant result is calculated together with its complex conjugation. In any case GRG_{EC} is more than twice faster than *Excalc* on this problem.

GRG_{EC} has not been entirely completed yet and will be further developed and amended. Nevertheless, practice amassed gives the evidence in favour of its efficiency and excellent convenience for practical calculations. Hence, GRG_{EC} may be estimated as a promising base for the developing the advanced efficient tool for the doing computer analysis of a wide scope of problems in important fields of the theoretical physics.

Acknowledgments

S.T. was partially supported in frames of the Russia Ministry of Science project 640.01.77. I.G.O. is grateful to Stan Steinberg and *Ecodynamics Research Associates, Inc.* for partial support of this work. The authors are grateful to Dr. M.A.H. MacCallum for the useful comments on the draft version of the paper and to V.V. Zhytnikov for the important information concerning the CAS's performance rate.

References

[1] Wolfram, S. *Mathematica. A System for Doing Mathematics by Computer*, Second edition, *Addison-Wesley*, 1991; L. Parker and S.M. Christensen, *MathTensor. A system for doing tensor analysis by computer*, *Addison-Wesley*, 1994.

[2] *Macsyma. Reference Manual, version 13, Macsyma Inc.*, 1992.

[3] *Maple V. Library Reference Manual*, Springer-Verlag; *GRTensor. Component tensor calculations for General Relativity, Version 0.26*, 1993; Musgrave, P., Pollney, D. and Lake, K. *GRTensorII, Queen's University, Kingston,Ontario*, 1994.

[4] Hearn, A. *Reduce User's Manual. Version 3.5*, Rand publication *CP78*, 1993; Schruefer, E. *EXCALC. A System for Doing Calculations in the Calculus of Modern Differential Geometry. USER's MANUAL*, 1993; Kadlecsik, J. *Ricci calculus package in Reduce, User's Manual*, 1994; Hasper, J.F. and Dyer, C.C. *Tensor Algebra With REDTEN. A User Manual.* 1992; Staufer, D., Hehl F.W., Ito, N., Vinkelmann, V., and Zabolitzki, J.G. *Computer simulation and computer algebra*, Third edition, *Springer, Berlin*, 1993; Schruefer, E., Hehl F.W. and McCrea, J.D. *Gen. Rel. Grav.*, **19**, 197, 1987.

[5] Marti, J., Hearn, A., Griss, M., Griss, C. *Standard Lisp Report, SIGPLAN Notes, ACM, N.Y.*, **14**, N 10, p. 48, 1979.

[6] Fric I. *Sheep. User's guide. University of Stockholm*, 1977; MacCallum M.A.H. and Skea, J.E.F. *SHEEP: A computer algebra system for general relativity*, in *"Algebraic computing in general relativity", (Proceedings of the first Brazilian school on computer algebra, vol 2)*, eds. Reboucas M.J. and Roque, W.L., *Oxford Univ. Press, Cambridge*, pp. 1-172, 1994; Hörnfeldt, L. *Stensor. Reference Manual, University of Stockholm.*

[7] *The system ORTOCARTAN - user's manual, fourth edition, Warsaw*, 1992; Krasiński, B., Perkowski, A., *Gen. Rel. Grav.*, **25**, 165, 1993.

[8] Kramer, D., Stephani, H., Herlt, E., MacCallum M. *Exact solutions of Einstein's equations, Cambridge Univ. Press, Cambridge*, 1980.

[9] Bondi, H., van der Burg, M. G. J., Metzner, A.W.K., *Proc. Roy. Soc. London*, **A269**, 21, 1962; Campbell, S.J. and Wainwright, J. *Gen. Rel. Grav.*, **8**, 978, 1987.

[10] Tertychniy S.I., Zhytnikov V.V., Ponomariev V.N. *The specialized programming system for analytical calculations in General Relativity*, Contributed papers on GR-10, eds. Bertotti, B., DeFelice, F., Pascolini, A., *Roma*, 1983, vol. 1, p.449-450; Zhytnikov V.V., Obukhova I.G. and Tertychniy S.I. *Computer Algebra System for Calculations in Gravity, in: Proceedings of the VII Soviet conference on Gravity Theory, Yerevan University*, 1988, p. 67-68; Obukhova I.G., and Tertychniy S.I. and Zhytnikov V.V. *GRG - computer algebra program for gravity and classical field theory, Preprint Moscow State Pedagogical Institute*, 1991, 14p; Zhytnikov V.V., Obukhova I.G. and Tertychniy S.I. *Computer Algebra Program for Gravitation*, in: Abstracts of contributed papers of the GR-13, *Huerta Grante, Cordoba*, 1992) p. 309;

[11] Zhytnikov, V.V. *GRG: Computer Algebra System for Differential Geometry Gravitation and Field Theory Version 3.1, Moscow*, 1992, Chung-Li, 1993; GRG 3.1 is available via anonymous ftp on '`ftp.maths.qmw.ac.uk`' in the directory '`pub/grg3.1`'.

[12] Pollney, D., Musgrave, P., Santosuosso, K. and Lake, K. *Class. Quantum Grav.*, **13**, 2289, 1996.