TECHNICAL TRANSACTIONS | CZASOPISMO TECHNICZNE
FUNDAMENTAL SCIENCES | NAUKI PODSTAWOWE

1-NP/2015

ANDRZEJ WOSZCZYNA[*], PIOTR PLASZCZYK[**],
WOJCIECH CZAJA[***], ZDZISŁAW A. GOLDA[****]

SYMBOLIC TENSOR CALCULUS — FUNCTIONAL AND DYNAMIC APPROACH

ZASTOSOWANIA PROGRAMOWANIA FUNKCYJNEGO I DYNAMICZNEGO DO SYMBOLICZNEGO RACHUNKU TENSOROWEGO

Abstract

In this paper, we briefly discuss the dynamic and functional approach to computer symbolic tensor analysis. The *ccgrg* package for Wolfram Language/Mathematica is used to illustrate this approach. Some examples of applications are attached.

Keywords: computer algebra, symbolic tensor calculus, functional programming, dynamic programming, Mathematica, Wolfram Language

Streszczenie

Krótko omawiamy zastosowania programowania dynamicznego i funkcyjnego do symbolicznego rachunku tensorowego. Demonstrując ten rodzaj programowania, posługujemy się pakietem *ccgrg.m* napisanym dla języka Wolfram Language/Mathematica. Zamieszczamy kilka przykładów ilustrujących funkcjonalność pakietu.

Słowa kluczowe: algebra komputerowa, symboliczny rachunek tensorowy, programowanie funkcyjne, programowanie dynamiczne, Mathematica, Wolfram Language

DOI: 10.4467/2353737XCT.15.110.4147

This paper was prepared in LaTeX.

[*]Institute of Physics, Cracow University of Technology; Copernicus Center for Interdisciplinary Studies; uowoszcz@cyf-kr.edu.pl

[**]Astronomical Observatory, Jagiellonian University; piotr.plaszczyk@uj.edu.pl

[***]Copernicus Center for Interdisciplinary Studies; wojciech.czaja@gmail.com

[****]Astronomical Observatory, Jagiellonian University; Copernicus Center for Interdisciplinary Studies; zdzislaw.golda@uj.edu.pl

1. Introduction

Cartesian tensors are identified with matrices. In contrast, in curved coordinates, even more so in a curved space, viewing the tensor matrix *in extenso* is not instructive. The subject of interpretation are equations, invariants, symmetries, and observables, while the tensor components values — except for certain privileged frames of reference — are of minor importance. This follows directly from the general principle of covariance, which states that the physical sense is independent of the choice of coordinate system, whereby each coordinate choice is allowed.

The vast majority of the computer tensor tools — including those build in Wolfram Language/Mathematica, the symbolic language which we use here — identify tensors with matrices, providing them with procedures for tensor contraction, tensor product and the rules of symmetry. Objects of such a class require the evaluation of complete tensor matrices. A sequence of intermediate steps evaluates quantities, most of which are irrelevant to the actual goal of the calculation, and are an unnecessary burden. A few tensor packages can operate on abstract level, where the differential manifold is not specified. They cover the entire tensor algebra, and essentially, form the sophisticated programming languages [8, 12, 13]. On the other hand, all of them depend on the primary computer algebra languages they employ (Mathematica, Mapple) — the languages which constantly develop. The risk of losing compatibility between the primary language and the computer algebra package is greater the more complex the package.

The project we realized in 2013–2014 was aimed at constructing an open source package [1] for Mathematica, which defines some basic concepts and rules, and is open to development by users. The syntax is kept as close as possible to standard textbook notation [2] and typical scheme of manual calculations. On the other hand, we employ dynamic programming with modern computational techniques, a *call-by-need strategy* and *memoization*. The question is not how much can be computed, but how flexible the software can be? The results are discussed in this paper.

We have taken into account that physicists have in mind co- and contravariant tensor components, rather than individual tensors of a precised valence (covariant or contravariant tensors). Rising and lowering tensor indices is a basic manual technique which comes directly from the Einstein convention. What physicist actually do is they aggregate tensors of arbitrary valence (all combinations of upper and lower indices) into a single object. The choice of particular valence is left for the future, and decided upon later. In order to stick to that, while the matrix representation is employed, one would need to evaluate and store all the co- and contravariant index combinations. In the case of the Riemann tensor in four dimensions, this gives 8^4 expressions, while only a small proportion of them will probably be used further on. Although they are not independent as subjects to tensor symmetries, the benefits of this fact are not straightforward. Modern techniques of differentiation, and reducing terms are quick enough, while the evaluation of conditions *If* still cost time. As a result, computing the same component twice may cost only slightly more than retrieving it from another matrix element. But there still remains the problem of memory. In the matrix approach, thousands of array's components are build of long algebraic expressions.

The functional representation of tensors allows one to avoid some of these problems. In this paper, we focus on the *ccgrg*-package (the open source code written for *Mathematica*

9–10, and distributed from Wolfram Library [3]). The goal of the article is to argue for some particular computation technique. We do not give here the precise tutorial to the package. The same approach to tensors in Python is presented in [4].

2. Dynamic paradigm of programming

The idea is not new, as it dates back to the famous Richard Bellman book *Dynamic Programming (1957)*. Bellman introduces the concept of the goal function. This means that the starting point is identical to the final expression in the search. To evaluate this expression, the algorithm automatically splits tasks into sub tasks down to elementary level. The tree of calls is not explicitly known to programmers, and the computational process does not require any intelligent interference. For tensors, this means that the subroutines do not evaluate all the components but some of them, acting on demands of routines from a higher level. Most tensor components remains unknown. Only these are computed which contribute to the final expression. This process is known as *lazy evaluation* or *call-by-need strategy* [5, 6]. To evaluate Ricci scalar for Schwarzschild metrics (33), *ccgrg* calls 16 Riemann components. 40 Riemann components are needed for full Ricci tensor, and 808 Riemann components for Carminati–McLenaghan $\mathscr{W}_1$ invariant [7] $\mathscr{W}_1 = \frac{1}{8}(C_{abcd} + \mathtt{i}\,{}^*C_{abcd})C^{abcd}$ build of the Weyl C and dual Weyl ${}^*C$ contractions. In each case, the appropriate components are selected automatically and the rest of components remains unevaluated.

Tensors form a class of functions, where the tensor indices are the innermost group of arguments. For instance, the covariant derivative we denote as:

$$T_{ij;m} \to \nabla[\mathtt{T}][\mathtt{i},\mathtt{j},\mathtt{m}], \tag{1}$$

$$T_{ijk;mn} \to \nabla[\mathtt{T}][\mathtt{i},\mathtt{j},\mathtt{k},\mathtt{m},\mathtt{n}]. \tag{2}$$

The outer argument contains the tensor name, while the inner argument lists the tensor indices. In the case of Lie derivative

$$\pounds_U V_i \to \mathtt{LieD}[\mathtt{U}][\mathtt{V}][\mathtt{i}], \tag{3}$$

$$\pounds_U T_{ij} \to \mathtt{LieD}[\mathtt{U}][\mathtt{T}][\mathtt{i},\mathtt{j}], \tag{4}$$

the outermost argument U is the vector field that generates the dragging flow, the middle (`V` or `T`) is the tensor name, and the innermost, according to the general rule, specify indices. The grouping of arguments, and the appropriate ordering of groups allows one to work with headers $\nabla[\mathtt{T}]$, `LieD[U][T]`, etc., treating them as operators. Inserting new groups of outer arguments does nor affect the methods governing tensor index operations. Indexes run through $\{1,\ldots,\mathrm{dim}\}$ for covariant components and through $\{-\mathrm{dim},\ldots,-1\}$ for contravariant components (the inverse Parker–Christensen notation [8]). Thus we write:

$$T_{ij} \to \mathtt{T}[\mathtt{i},\mathtt{j}], \tag{5}$$

$$T_i^{\ j} \to \mathtt{T}[\mathtt{i},-\mathtt{j}], \tag{6}$$

$$T^{ij} \to \mathtt{T}[-\mathtt{i},-\mathtt{j}], \tag{7}$$

$$T^{ij}{}_k \to \mathtt{T}[-\mathtt{i},-\mathtt{j},\mathtt{k}], \tag{8}$$

and so on. Tensor valence operations (conversions from covariant to contravariamt components, and vice versa), and the covariant differentiation are the only methods of the object. Other tensor operations like contractions, products, etc. are realized by elementary algebra and the Einstein convention. Conversions are called just by placing or removing the sign "minus" in front of selected tensor indices.

3. The manifold

Tensors live on a manifold. This manifold is declared in the opening procedure `open[x,g]`, where the list of coordinate names x and metric tensor g are the arguments. When the manifold is 'opened', tensors are evaluated on this particular manifold. The metric tensor can be uniquely defined, or may contain arbitrary functions of coordinates. In the last case, `open` indicates the class of manifolds distinguished by the specified structure of the metric tensor.

Tensors are created by setting object data in the form of the pair: tensor matrix, and the associated tensor valence (all covariant components as a default). By calling `tensorExt[matrix, valence]` one creates a tensor object which can return components for arbitrary valence, on demand. Tensors which are in frequent use (Ricci, Riamann, Weyl, the first and the second fundamental forms on hypersurfaces), as well as covariant and Lie derivatives are predefined in the *ccgrg*-package. Tensor components are not evaluated in advance, only when called for the first time. This is guaranteed by the SetDelayed (:=) command. For instance, when calling the Riemann curvature tensor in the Schwarzschild spacetime one obtains:

$$\begin{aligned} R_{1212} \to \texttt{In[1]} &:= \texttt{tRiemann}[1,2,1,2] \\ \texttt{Out[1]} &:= -\frac{2M}{r^3} \end{aligned} \tag{9}$$

$$\begin{aligned} {R_{121}}^{2} \to \texttt{In[2]} &:= \texttt{tRiemann}[1,2,1,-2] \\ \texttt{Out[2]} &:= -\frac{2M(2M-r)}{r^4} \end{aligned} \tag{10}$$

$$\begin{aligned} R_{ijmn}R^{ijmn} \to \texttt{In[3]} &:= \texttt{Sum}[\texttt{tRiemannR}[\texttt{i},\texttt{j},\texttt{m},\texttt{n}]\texttt{tRiemannR}[-\texttt{i},-\texttt{j},-\texttt{m},-\texttt{n}] \\ &\quad \{\texttt{i},\texttt{dim}\},\{\texttt{j},\texttt{dim}\},\{\texttt{m},\texttt{dim}\},\{\texttt{n},\texttt{dim}\}] \\ \texttt{Out[3]} &:= 48\frac{M^2}{r^6} \end{aligned} \tag{11}$$

The tensor rank is fixed, except for the covariant derivative, where each differentiation rises the rank by one. Covariant differentiation is realized by header ∇ (or its full name, `covariantD`), and by extending the list of indices by one. Extension of this list by more than one returns derivatives of a higher order.

$$\begin{aligned} {R_{121}}^{2}{}_{;2} \to \texttt{In[1]} &:= \nabla[\texttt{tRiemannR}][1,2,1,-2,2] \\ \texttt{Out[1]} &:= -\frac{6M(2M-r)}{r^5} \end{aligned} \tag{12}$$

$$ {R_{121}}^{2}{}_{;22} \to \texttt{In[2]} := \nabla[\texttt{tRiemannR}][1,2,1,-2,2,2]$$

$$\texttt{Out[2]} := -\frac{6M(9M^2-4Mr)}{r^6} \tag{13}$$

Rising the tensor rank by differentiation does not affect the general ability to rise or lower indices by choosing appropriate signs, both for the original and the differentiation indices. The flexibility of tensor indexes stems directly from the functional representation of tensor and the *call-by-need strategy*.

4. Memoization and symmetries

Trees of calls for different tasks may have nonempty intersections (and typically this is the case). The `SetDelayed` function which enables *call-by-need strategy* at the same time forces the multiple evaluation of components whenever trees of calls overlap. To avoid unwilling effects, one needs to store computed values, and to have a mechanism that can recognize known and unknown components. This process is called *memoization*. The term *memoization* was introduced by Donald Michie in 1968 [9] and refers to the class of functions that can learn what they found in past calls. In *C* or Python, the *memoization* tools are usually constructed by programmers (see [4]). Wolfram Mathematica provides dedicated tool in the core language. The construction consists in the recursive definition of the form

$$f[x\_] := f[x] = expr[x,\ldots] \tag{14}$$

where the same name of the function appears twice, first followed by `SetDelayed` and next, by `Set`. The definition of f which is initially equivalent to $expr[x,\ldots]$, is successfully supplemented by the values found during each function call. The memoized (memorized) values form cache. The cache is searched first; therefore, no tensor component is evaluated twice. This means that each of the expressions in the examples (10)–(14) is evaluated only once. Whenever called again, the values are instantly retrieved from the cache. The same refers to expressions called by subroutines. In the Schwarzschild space time (33), the Carminati–McLenaghan invariants $\mathscr{W}_1$ and $\mathscr{W}_2$, when evaluated separately, call 808 and 552 Riemann components, respectively. Jointly, they call 1046 Riemann components which means that 296 components are taken from the cache.

Calls appeal to tensor symmetries. The symmetry rules are the argument conditions within the same memoizing function scheme

$$\texttt{f}[\texttt{x\_}]/;\ \langle \text{test}[\text{x}]\rangle := \texttt{f}[\text{x}] = expr[x,\ldots] \tag{15}$$

For instance, for the Ricci tensor R_{ij}, the conditional definition takes the form

$$\begin{aligned}&\texttt{tRicciR}[\texttt{i\_},\texttt{j\_}]/;\texttt{i}>\texttt{j} := \texttt{tRicciR}[\texttt{i},\texttt{j}] = \texttt{tRicciR}[\texttt{j},\texttt{i}] \\ &\texttt{tRicciR}[\texttt{i\_},\texttt{j\_}] := \texttt{tRicciR}[\texttt{i},\texttt{j}] = \texttt{Sum}[\texttt{tRiemannR}[-\texttt{s},\texttt{i},\texttt{s},\texttt{j}],\{\texttt{s},\texttt{dim}\}]\end{aligned} \tag{16}$$

with positive i, j (covariant components). The rules of symmetry are introduced directly in the definition of the tensor. In order to recognize Ricci symmetries, the kernel does not refer to the symmetries of the Riemann tensor. Subsequently, the Riemann tensor symmetries are not the result of the evaluation of Christoffel symbols, etc. Symmetries defined as the argument conditions in lazy-evaluated functions put constraints directly on computational processes.

5. The confidence of results in dynamic programming

Dynamic programming allows one to reach the goal at a minimal cost of time and memory. The *call-by-need strategy* avoids computing unnecessary components. Memoization protects against multiple evaluations. The cost we pay, however, is confidence in the results. Functions that can learn and remember are reluctant to forget. The user must not change the tensor definition during the same session. The cache has priority over evaluation. A new definition does not affect the values which are already found. If definitions change, one may readily collect a mixture of results belonging to different definitions.

The content of the cache together with the general tensor definition can be viewed by the Mathematica command `Definition[tensorname]`. Based on this command, `cacheview[tensorname]` returns the list of the *memorized* tensor components. For instance

```
 In[26] :=  cacheview[tRiemannR]
Out[26] :=  {{tRiemannR,{−4,1,4,1}},{tRiemannR,{−3,1,3,1}}}        (17)
```

means that ${R^4}_{141}$ and ${R^3}_{131}$ Riemann components have been already called and stored in the cache. To view all the memoized quantities which refer to `tRiemannR`, one writes

```
 In[27] :=  associated[tRiemannR]
Out[27] :=  {{tRiemannR,{−4,1,4,1}},{tRiemannR,{−3,1,3,1}}
            {covariantD[tRiemannR],{1,2,1,2,2}}}}                  (18)
```

Commands

```
In[31] :=  retreat[tRiemannR]
In[32] :=  retreat[tRiemannR,associated]                            (19)
```

clear the stored values displayed in (17) and (18), respectively. In contrast to the core-languge *Mathematica* commands `Clear` and `Unset`, thr command `retreat` does not affect the general definition of a tensor. Calling `cacheview` after `retreat` returns the empty list.

For sake of safety, the name of each memoizind tensor shall be appended to `memRegistry`. This is realized by `erasable[tensorname]`. `memRegistry` allows the automatic erasing of the whole cache content whenever `open[x,g]` is called. User may safely redefine the manifold with no risk of unwanted remnants from the past computations. *ccgrg* is equipped with extensive random tests of memory clearance. The above mentioned security tools do not prevent some incidental mistakes such as those shown below

```
In[41] :=  f[k_] := f[k] = k;f/@Range[8];
In[42] :=  f[k_] := f[k] = 1/k;f/@Range[8];
```

To effectively implement *memoization* to algorithms, minimal training and discipline are indispensable.

6. Tensor definition scheme

This is the place to show the typical construction of a tensor. We choose the induced metrics (the first fundamental form) $h_{ij} = g_{ij} - v_i v_j$ on the hypersurface orthogonal to the vector field v_i in the space with the metric g_{ij}.

```
erasable/@{hcov,h};                                                    (20)
hcov[v_][j_,i_]/;i < j := hcov[v][j,i] = hcov[v][i,j];
hcov[v_][i_,j_] := g[i,j] − v[i]v[j]/vectorsquared[v]//simp[];
h[v_][i_,j_]/;indeX[i,j] := h[v][i,j] = tensorExt[hcov[v]][i,j]//simp[];
```

In the first line we append `memRegistry`. The next line contains the symmetry settings. The third line defines the matrix of the covariant tensor (`hcov` is not a tensor in the meaning of the *ccgrg* package and will be invisible in the general context.) The last line creates the tensor `h` as an object containing data (`hcov`), methods to control the index values (condition `indeX`), and methods to rise or lower indices (provided by the `tensorExt` procedure).

Following the scheme above, users may define their own tensors and append to the package, or just use them in notebooks. Summation, multiplying tensors by numbers or functions (not containing indices as arguments!), and covariant differentiation return tensors which not need to be separately defined as in (20). Yet, even in these cases, the definition scheme (20) is strongly recommended to assure effectiveness.

7. Examples of use

7.1. Differential operators in an arbitrary coordinate system

In technical sciences there is a need to express differential operators in some particular curved coordinates or on curved surfaces. The task may appear as a part of the heat transport or diffusion problems for systems of more complex geometry. Many typical gradient or Laplacian expressions are catalogued in the literature, but still the range of geometries that can be encountered in nature is much richer. Below, we choose a catenoid — one of the best known minimal surfaces — for which, we hope, Laplacian is not published in print.

The family of catenoids numbered by r and parametrized by u and v define a curved coordinate system in the Euclidean space. The first step is to find the Euclidean metric form in coordinates r, u, v (the core Wolfram Language)

```
In[1] :=  Needs["ccgrg‘"];
In[2] :=  x[r_,u_,v_] = rCosh[v/r]Cos[u];
In[3] :=  y[r_,u_,v_] = rCosh[v/r]Sin[u];
In[4] :=  z[r_,u_,v_] = v;
In[5] :=  crd = {r,u,v};
In[6] :=  form = Dt[x[r,u,v]]^2 + Dt[y[r,u,v]]^2 + Dt[z[r,u,v]]^2//FullSimplify
```

$$\mathtt{Out[6]:=}\quad \mathtt{Cosh}[\frac{\mathtt{v}}{\mathtt{r}}]^2(\mathtt{r}^2\mathtt{Dt[u]}^2+\mathtt{Dt[v]}^2)+\frac{\mathtt{Dt[r]}^2(\mathtt{rCosh}[\frac{\mathtt{v}}{\mathtt{r}}]-\mathtt{vSinh}[\frac{\mathtt{v}}{\mathtt{r}}])^2}{\mathtt{r}^2}+ \frac{\mathtt{Dt[r]Dt[v]}(\mathtt{v}-\mathtt{vCosh}[\frac{2\mathtt{v}}{\mathtt{r}}]+\mathtt{rSinh}[\frac{2\mathtt{v}}{\mathtt{r}}])}{\mathtt{r}} \tag{21}$$

The second step is to evaluate the covariant differentiation $f_{,i}{}^{;i}$ of an arbitrary function f on the manifold with the metric form (21):

```
In[7] :=  open[crd, toMatrix[form, crd]];
In[8] :=  Sum[∇[f[r,u,v]][i,−i], {i,dim}]//FullSimplify
```

$$\mathtt{Out[8]:=}\quad \frac{\mathtt{r}^2\mathtt{f}^{(2,0,0)}[\mathtt{r,u,v}]}{\left(\mathtt{r}-\mathtt{vTanh}\left[\frac{\mathtt{v}}{\mathtt{r}}\right]\right)^2}+\frac{\mathtt{Sech}^2\left[\frac{\mathtt{v}}{\mathtt{r}}\right]\mathtt{f}^{(0,2,0)}[\mathtt{r,u,v}]}{\mathtt{r}^2}+\mathtt{f}^{(0,0,2)}[\mathtt{r,u,v}]+ \frac{2\mathtt{rf}^{(1,0,1)}[\mathtt{r,u,v}]}{\mathtt{v}-\mathtt{rCoth}\left[\frac{\mathtt{v}}{\mathtt{r}}\right]}-\frac{\left(\mathtt{rSinh}\left[\frac{2\mathtt{v}}{\mathtt{r}}\right]+2\mathtt{v}\right)^2\mathtt{Sech}^4\left[\frac{\mathtt{v}}{\mathtt{r}}\right]\mathtt{f}^{(1,0,0)}[\mathtt{r,u,v}]}{4\left(\mathtt{r}-\mathtt{vTanh}\left[\frac{\mathtt{v}}{\mathtt{r}}\right]\right)^3} \tag{22}$$

The only time-consuming operations are simplifications `FullSimplify`, which are inevitable in this code. Similar computations can be easily performed for an arbitrary transformation defined by inputs the `In[2]−In[4]`.

7.2. Riemann curvature invariants: the case of spherical symmetry

Carminati–McLenaghan invariants [7] form a complete set of Riemann invariants for the spacetime which obey the Einstein equations with the hydrodynamic energy-momentum tensor. These invariants read

$$\mathscr{R}_1 = \frac{1}{4}S^a{}_bS^b{}_a, \tag{23}$$

$$\mathscr{R}_2 = -\frac{1}{8}S^a{}_bS^b{}_cS^c{}_a, \tag{24}$$

$$\mathscr{R}_3 = \frac{1}{16}S^a{}_bS^b{}_cS^c{}_dS^d{}_a, \tag{25}$$

$$\mathscr{W}_1 = \frac{1}{8}(C_{abcd}+i^*C_{abcd})C^{abcd}, \tag{26}$$

$$\mathscr{W}_2 = -\frac{1}{16}(C_{ab}{}^{cd}+i^*C_{ef}{}^{ab})C_{cd}{}^{ef}C_{ef}{}^{ab}, \tag{27}$$

$$\mathscr{M}_1 = \frac{1}{8}S^{ad}S^{bc}(-i^*C_{abcd}+C_{abcd}), \tag{28}$$

$$\mathscr{M}_2 = \frac{1}{8}i^*C_{abcd}S^{bc}S_{ef}C^{aefd}+\frac{1}{16}S^{bc}S_{ef}(C_{abcd}C^{aefd}-{}^*C_{abcd}{}^*C^{aefd}), \tag{29}$$

$$\mathscr{M}_3 = \frac{1}{16}S^{bc}S_{ef}C_{abcd}C^{aefd}+\frac{1}{16}S^{bc}S_{ef}{}^*C_{abcd}{}^*C^{aefd}, \tag{30}$$

$$\mathscr{M}_4 = -\frac{1}{32}S^{ag}S^c{}_dS^{ef}C_{ac}{}^{db}C_{befg}-\frac{1}{32}S^{ag}S^c{}_dS^{ef*}C_{ac}{}^{db*}C_{befg}, \tag{31}$$

$$\mathscr{M}_5 = \frac{1}{32} S^{bc} S^{ef} (i^{*}C^{aghd} + C^{aghd})({}^{*}C_{abcd}{}^{*}C_{gefh} + C_{abcd}C_{gefh}). \tag{32}$$

S_{ab} stands for the Plebanski tensor (the traceless Ricci tensor $S_{ab} = R_{ab} - \frac{1}{4}Rg_{ab}$). Invariants $\mathscr{W}_1$ and $\mathscr{W}_2$ are complexes, which are defined by means of contractions of the Weyl C and the dual Weyl ${}^{*}C$ tensors.

In the case of vacuum and spherically symmetric spacetime (Black Hole)

$$ds^2 = -\left(1 - \frac{2M}{r}\right) dt^2 + \left(1 - \frac{2M}{r}\right)^{-1} dr^2 + r^2 \left(d\vartheta^2 + \sin^2\vartheta d\varphi^2\right), \tag{33}$$

the complete computation appears as follows:

```
 In[1] :=  Needs["ccgrg`"];
 In[2] :=  x = {t,r,ϑ,φ};
 In[3] :=  g := DiagonalMatrix[−(1 − 2M/r),(1 − 2M/r)^−1,r^2,r^2Sin[ϑ]]^2];
 In[4] :=  $Assumptions = And@@{0 < t,0 < r,0 < ϑ < π,0 < φ < 2π,0 < M};
 In[5] :=  open[x,g];
 In[6] :=  {CMinvR1,CMinvR2,CMinvR3}
Out[6] :=  {0,0,0}
 In[7] :=  {CMinvW1,CMinvW2}
Out[7] :=  {6M^2/r^6,−6M^2/r^6}
 In[8] :=  {CMinvM1,CMinvM2,CMinvM3,CMinvM4,CMinvM5}
Out[8] :=  {0,0,0,0,0}
 In[9] :=  {TimeUsed[],$MachineType,$Version,$ProcessorType,
           $ProcessorCount}
Out[9] :=  {8.80264,PC,10.0forLinuxx86(64-bit)(December4,2014),x86-64,2}
```
(34)

As see result of these operations, two nonvanishing invariants were found, while 1320 of 4096 components of the Riamann tensor object were evaluated and `memorized`. For the PC computer with AMD64 processor and Linux Mint x86 (64-bit), the whole operation took 8.80264 seconds CPU.

8. Final remarks

The degree of complexity in a symbolic programming is much less predictable than in numerical computation. This is due to the fact that the quantity of algebraic terms in an expression is not a uniquely determined number. This number significantly depends on the ability of the algorithm to recognize mathematical identities, and for the same function, may differ in orders of magnitude. In this circumstance, the economic style of the dynamic programming (*call-by-need strategy, memoization*) may take the first rank role.

The research supported by grant from the John Templeton Foundation.

References

[1] Woszczyna A., Kycia R. A., Golda Z. A., *Functional Programming in Symbolic Tensor Analysis*, Computer Algebra Systems in Teaching and Research, Vol. IV, No 1, pp. 100–106, 2013.

[2] Synge J. L., Schild A., *Tensor Calculus*, University of Toronto Press, 1959.

[3] Woszczyna A., et al.: *ccgrg — The symbolic tensor analysis package, with tools for general relativity*, 2014, http://library.wolfram.com/infocenter/MathSource/8848/.

[4] Czaja W., *GraviPy*, https://pypi.python.org/pypi/GraviPy/0.1.0.

[5] Hudak P., *Conception, Evolution, and Application of Functional Programming Languages*, ACM Computing Surveys **21**, pp. 383–385, 1989.

[6] Reynolds J. C., *Theories of programming languages*, Cambridge University Press, 1998.

[7] Carminati J., McLenaghan R. G., *Algebraic invariants of the Riemann tensor in a four-dimensional Lorentzian space*, J. Math. Phys. **32**, 3135, 1991.

[8] Parker L., Christensen S. M., *MathTensor: A System for Doing Tensor Analysis by Computer*, Reading, MA: Addison-Wesley, 1994.

[9] Michie D., *Memo Functions and Machine Learning*, Nature, No. 218, 1968, pp. 19–22.

[10] Wald R. M., *General Relativity*, The University of Chicago Press, 1984.

[11] [Tensorial homepage] http://home.comcast.net/˜djmpark/TensorialPage.html.

[12] [GRTensor homepage] http://grtensor.phy.queensu.ca/.

[13] [xAct homepage] http://www.xact.es/.

[14] Korol'kova A. V., Kulyabov D. S., Sevastyanov L. A., *Tensor computations in computer algebra systems*, Programming and Computer Software, 39 (3), 2013, pp. 135–142.