An Application of Symbolic Computation in the Physical Sciences

Charles C. Dyer
Departments of Astronomy and Computer Science
and Scarborough College, University of Toronto
1265 Military Trail, Scarborough, Ontario, Canada M1C 1A4

Abstract

An example of a problem in the physical sciences is discussed where application of various symbolic computation facilities available in many algebraic computing systems leads to a significant expansion of the range problems that can be solved. Since most interesting problems in the physical sciences eventually require the numerical solution of systems of equations, of various types, we will introduce an example and describe an apporach to a solution, beginning at the development of relevant differential equations, using, for example REDUCE, and leading eventually to the generation of highly efficient and stable numerical code for the solution, using, in our case, the C language. The use of SCOPE and GENTRAN,as well as series packages in REDUCE will be discussed.

In many areas of interest, a considerable amount of work has to be performed to arrive at the symbolic equations to solve, and this is particularly true in General Relativity and related gravitation theories. Some packages, such as REDTEN [1], for calculation in this field will be discussed.

1 Introduction

With the development of good symbolic mathematical systems, the physical scientist has a set of powerful new tools to expand the realm of tractable problems related to physical systems. Initially, this expansion impacts the possibility of performing complex symbolic calculations that would have been much more difficult when done by hand. There are many useful calculations that are essentially mechanical in nature, but are discouragingly long to carry out, and consequently prone to errors. One of the basic methods by which science proceeds is to cast forward well beyond the present wisdom, and then to derive the consequences of the new idea, imagining a new and exciting possibility, say for a geometric structure in General Relativity. If the time required to derive the field equations for the proposed new geometry is very long, this imposes a significant limitation on the rate of progress and the daring of such leaps. The case of General Relativity provides a good example of this issue, where simply the derivation of the field equations for a particular situation can take considerable time, and can be prone to errors which completely alter the physical situation. A number of systems exist for use in tensor analysis, differential geometry, etc., with particular on General Relativity, including EXCALC, MapleTensor, MathTensor, REDTEN, SHEEP, and others.

We have developed and used REDTEN [1], which runs in REDUCE. It provides quite a broad suite of facilities for dealing with the mathematics of theories like General Relativity, with the availability of all the usual curvatures, geodesic equations, Lie derivatives, Killing equations, etc. We have found REDUCE to be a particularly good platform on which to build such a system as REDTEN, in part because of its firm basis in LISP, which provides for excellent garbage collection, easy input/output handling to allow easy addition of notation specific to particular application areas, such as tensor derivative notation. In addition, REDUCE has a very mature body of application packages, frequently with a strong basis in the physical and engineering sciences. While the situation has improved in some competitive systems, our earlier efforts in some sytems were frustrated by the limitations imposed in some of the above mentioned issues.

The purely formal problems discussed above frequently lead eventually to systems of equations which do not yield to analytic methods resulting in closed-form solutions. Even in the apparently simple case of Newtonian gravitation, with its rigid geometry, only the very simplest of imaginable physical situations can be solved in closed form. Thus, for much of the physical sciences, the problem does not end with the formal mathematical system, but requires the use of complex numerical programs to arrive at physically meaningful solutions. It is frequently the case that the output from the symbolic mathematics system is far from the form that is usable for numerical solution. It is here that the proper union between algebraic systems and good numerically oriented compilers requires careful consideration, both for efficiency and for the integrity of the solutions produced. To illustrate the relationship between symbolic mathematical systems and useful results in the physical sciences, we will consider a particular problem whose solution benefits significantly from the union.

While most symbolic computation systems are highly capable of solving many problems, they require considerable re-casting of mathematical notation, simplification rules, etc., to be useful as extensions of the normal working modes of physical scientists. Significant work remains to be done to address the difficulties in mapping from the "working envi-

ISAAC 94 - 7/94 Oxford England UK
© 1994 ACM 0-89791-638-7/94/0007..$3.50

ronment assumptions" of a typical physical scientist to the very precise specification of dependenceies, etc., required to obtain correct results from algebraic systems. The impact of such difficulties in the acceptance of the use of algebraic systems is considerable, with frequent expressions of frustration by users when a particular system seems unable to simplify a particular result when the user can see very clearly that a simpler forms exists. More subtle, but much more damaging are the cases where a user neglects to specify appropriate dependencies, frequently because the human being is so capable of keeping track of side relations without deliberate effort. There are many stories of truly beautiful results, which are later (hopefully not too much later, and by the user, not a referee!) discovered to derive their beauty from an unfortunately forgotten dependency specification. With the rapid increase in desltop computing power, going to the other extreme with regard to dependencies might well be a workable alteration, where everything depends on everything, unless otherwise specified. Of course at the practical level, the best compromise would probably be to place a mark in the order of introduction of objects, after which all mutual deppendencies are assumed until explicity specified.

2 Solutions for Self-Gravitating Cosmic Strings

As a case study in the range of uses for symbolic computation, we consider the solution of the coupled Einstein field equations and the Euler-Lagrange equations for a scalar field in the case of cylindrical symmetry. The details of the physics involved can be found in [2].

The space-time geometry is described by the metric for the line element:

$$ds^2 = -e^{2(K-U)}(dt^2 - dr^2) + e^{-2U}W^2 d\phi^2 + e^{2U}dz^2$$

where U, K and W are unknown functions of r only and will be solved for in our calculations. Using this metric, the Euler-Lagrange equations for the scalar field with potential V become:

$$e^{2(U-K)}\left(R'' + R'\frac{W'}{W}\right) - RP^2\frac{e^{2U}}{W^2} = \frac{dV}{dR} \quad (1)$$

$$e^{2(U-K)}\left(P'' - P'\left(\frac{W'}{W} - 2U'\right)\right) = e^2R^2P \quad (2)$$

where $' \equiv \partial/\partial r$. The energy-momentum tensor, $T_{\mu\nu}$, can be computed from the Lagrangian density, and the Einstein tensor, $G_{\mu\nu}$, can be computed from the line element. These calculations can be performed easily in most tensor manipulation packages, such as REDTEN, MathTensor, or MapleTensor. The resulting Einstein field equations, $G^\mu_\nu = 8\pi T^\mu_\nu$, which couple the string stress-energy to the space-time geometry, are then:

$$8\pi\left[R'^2 + \frac{e^{2K}}{W^2}R^2P^2 + \frac{e^{2U}}{W^2e^2}P'^2 + 2Ve^{2K-2U}\right] = -2\frac{W''}{W} + 2K'\frac{W'}{W} - 2U'^2 \quad (3)$$

$$8\pi\left[R'^2 - \frac{e^{2K}}{W^2}R^2P^2 + \frac{e^{2U}}{W^2e^2}P'^2 - 2Ve^{2K-2U}\right] = 2K'\frac{W'}{W} - 2U'^2 \quad (4)$$

$$8\pi\left[-R'^2 + \frac{e^{2K}}{W^2}R^2P^2 + \frac{e^{2U}}{W^2e^2}P'^2 - 2Ve^{2K-2U}\right] = 2K'' + 2U'^2 \quad (5)$$

$$8\pi\left[R'^2 + \frac{e^{2K}}{W^2}R^2P^2 + \frac{e^{2U}}{W^2e^2}P'^2 + 2Ve^{2K-2U}\right] = -2\frac{W''}{W} + 4U'\frac{W'}{W} + 4U'' - 2U'^2 - 2K'' \quad (6)$$

The integrability of the Einstein field equations requires that the energy-momentum tensor be divergence-free, ie. $T^{\mu\nu}_{\ ,\nu} = 0$. These conservation laws often give, analogous to the first integrals of classical mechanics, an important indication of how to solve the field equations. Using the energy-momentum tensor given above, the only non-trivial conservation equation requires that

$$R'\left(e^{2(U-K)}\left(R'' + R'\frac{W'}{W}\right) - RP^2\frac{e^{2U}}{W^2} - \frac{dV}{dR}\right) + P'\frac{e^{2U}}{e^2W^2}\left(e^{2(U-K)}\left(P'' - P'\left(\frac{W'}{W} - 2U'\right)\right) - e^2R^2P\right)$$

be zero. Note that this is derivable as a linear combination of equation (1) and (2) and therefore indicates that one of equation (1) and (2) can be taken as redundant. Since the conservation of energy equation was derived from the Einstein equations, one of these equations can be taken as redundant. We will take equation (4) as the redundant equation, which we will nevertheless continue using as a consistency check for our numerical integration, and keep both equations (1) and (2). Equation (4) can be rewritten as:

$$R'^2 = \frac{1}{4\pi}\left(K'\frac{W'}{W} - U'^2\right) + \frac{e^{2K}}{W^2}R^2P^2 - \frac{P'^2e^{2U}}{e^2W^2} + 2Ve^{2(K-U)} \quad (7)$$

Choosing the usually assumed form of the potential, $V = \alpha\frac{e^2}{8}(R^2 - \eta^2)^2$, and $K = 2U$ (derivable from a regularity condition on the axis), and using the the transformations $W \to \sqrt{8}W/\eta e$, $R \to \eta R$, $r \to \sqrt{8}r/\eta e$, and $e^U \to Z$, the set of equations to solve becomes (see [2] for details):

$$Z'' = \frac{1}{2}\pi\eta^2Z^3\left(\frac{P'^2}{W^2} - 16\alpha(R^2-1)^2\right) + Z'\left(\frac{Z'}{Z} - \frac{W'}{W}\right) \quad (8)$$

$$W'' = -8\pi\eta^2Z^2 \times \left(Z^2\frac{R^2P^2}{W} + 2\alpha W(R^2-1)^2\right) \quad (9)$$

$$R'' = Z^4\frac{RP^2}{W^2} + 4\alpha Z^2R(R^2-1) - R'\frac{W'}{W} \quad (10)$$

$$P'' = 8Z^2P^2P + P'\left(\frac{W'}{W} - 2\frac{Z'}{Z}\right) \quad (11)$$

$$R'^2 = \frac{1}{4\pi\eta^2}\frac{Z'}{Z}\left(2\frac{W'}{W} - \frac{Z'}{Z}\right) + Z^2\left(Z^2\frac{R^2P^2}{W^2} - \frac{P'^2}{8W^2} + 2\alpha(R^2-1)^2\right) \quad (12)$$

This system of equations is typical of those arising in General Relativity, being strongly coupled and nonlinear. In our case, the additional complications of the self-consistent solution with the Euler-Lagrange equations will lead to the requirement of a series of numerical solutions.

3 Conditions for finding solutions

The boundary conditions for the set of equations can be reduced to the following conditions, on the axis:

$$Z(0) = 1, \quad P(0) = W'(0)$$
$$Z''(0) = \frac{\pi\eta^2}{4}\left(\frac{P''(0)^2}{W'(0)^2} - 16\alpha\right) \tag{13}$$

including the free parameters $W'(0)$, $R'(0)$, $P''(0)$. Initial values are required for higher order derivatives up to the appropriate order, but these are derived by differentiating and taking limits onto the axis therefore no new parameter freedom is introduced. We thus have a 5-parameter (η, α, $W'(0)$, $R'(0)$, and $P''(0)$) set of equations. We now want to find solutions to the Einstein-scalar-gauge field equations by varying those parameters.

The action of a Lagrangian describing a vortex must remain finite in order that the gauge and scalar field adequately describe a vortex solution. Finite action requires that $\lim_{r\to\infty} R = 1$ and $\lim_{r\to\infty} P = 0$. A solution of our set of coupled equations will be one that can satisfy these asymptotic conditions.

4 Numerical Solution

The system of differential equations cannot be solved in closed form, but require numerical solution of some form. A number of methods were considered, and we concluded that a Taylor series method was most appropriate in this situation. Among the reasons for this choice, was the ease with which a scheme could be implemented, particularly with regard to imposing the variety of boundary conditions on the axis of the string, and the ease with which we could investigate the behaviour at large distance from the string. For both of these considerations, the availability of Taylor expansions for all the relevant functions directly from the numerical integration, as opposed to, for example, fitting to the functions at a later stage, was particularly convenient. In addition, since we were particularly in studying the gravitational effects of cosmic strings on light propagation and particle motions, the availability of actual metric functions, through their Taylor series, was very important. Thus the scheme which we wished to implement involved obtaining a reasonably high order Taylor series for the metric functions on each small integration interval, which would then serve as the metric on small regions of spacetime.

Even with this set of equations, the task of obtaining all the relevant Taylor expansions of the differential equations is a reasonable challenge, particularly to avoid errors that can creep into the calculation. In addition, since there are a number of parameters to search and tune for acceptable solutions, satisfying the asymptotic conditions, the amount of numerical computation is quite large, and any improvement in the quality of the code is very beneficial. Thus we considered the possibilities of implementing our scheme using an algebraic computation, such as Maple or REDUCE, as much as possible. Both Maple and REDUCE were used for the calculations, but REDUCE had a number of advantages in our particular case. The availability of high quality packages well suited to our purposes made the choice clear to us. We used the TAYLOR [3] and TPS [4] packages distributed with REDUCE 3.4 to develop the Taylor series and then manipulate them in order to produce the right hand sides of the differential equations. These packages are very easy to use, and both give the user a significant amount of control over their actions.

From the Taylor series, the integration scheme is quite simple to implement in principle, but the expressions quickly become very complicated, especially at the higher orders, say order 5 or 6, which we wish to maintain for later purposes, as well as accuracy. At this point, two other packages distributed with REDUCE come into play. The GENTRAN [5] package for transforming REDUCE expressions into numerical code, in languages such as C or FORTRAN, is available in other systems as well. To add especially good optimization in the production of this numerical code, we used the SCOPE [6] package in conjunction with GENTRAN. The use of these packages gave the ability to go from the formal mathematical statement of the coupled differential equations to the generation of globally optimized expressions for the expensive to compute terms for the Taylor series solutions. The optimization was very good, with excellent recognition of common subexpressions across many assignments, and excellent elimination of exponentations in the expressions, particularly appropriate for the C language. The documentation for SCOPE gives numerous examples of the level of optimization to obtainable, and the results of our application were consistent with those examples.

We did attempt a similar approach in MAPLE, using the code generation facilities available, including the optimization option. The results were not comparable with those produced by the SCOPE/GENTRAN combination. When combined with the quality of the series manipulation packages, our confidence in the REDUCE route was confirmed.

5 Code Production

The TAYLOR or TPS packages were used to generate, from the actual differential equations, the Taylor expansions for the dependent variables completely expanded in terms of the functions and their first derivatives, so that the Taylor integration scheme can be directly generated. The following extract from a source file for REDUCE and SCOPE reflects the result of using the Taylor packages. Since the actual expressions are very large, we only present here the first few Taylor coeficient terms, and then simply give the number of similar source lines for the higher order coefficients. The particular choice of control flags given for REDUCE is typical, but usually a variety of choices is useful, since their choice can influence the resulting code size, sometimes quite dramatically.

```
load_package scope;
off priall;
optlang!*:='C;
off pri;
on exp;

UPP:=(- 16*e**(2*u)*a**2*pet*r**4*w**2 + 32
*e**(2*u)*a**2*pet*r**2*w**2 - 16*e**(
2*u)*a**2*pet*w**2 + e**(2*u)*pet*
pp**2 - 8*up*w*wp)/(8*w**2)$

WPP:=(2*e**(2*u)*a**2*pet*( - e**(2*u)*p**2*
r**2 - 2*r**4*w**2 + 4*r**2*w**2 -
2*w**2))/w$

RPP:=(e**(4*u)*a**2*p**2*r + 4*e**(2*u)*a**2
*r**3*w**2 - 4*e**(2*u)*a**2*r*w**2 -
rp*w*wp)/w**2$

PPP:=(8*e**(2*u)*a**2*p*r**2*w - 2*pp*up*w +
```

```
 pp*wp)/w$

U3:=(16*e**(4*u)*a**2*p**2*pet*r**2*up*w +
16*e**(4*u)*a**2*p*pet*pp*r**2*w + 16*
e**(2*u)*a**2*pet*r**4*w**2*wp - 64*e
**(2*u)*a**2*pet*r**3*rp*w**3 - 32*e**
(2*u)*a**2*pet*r**2*w**2*wp + 64*e**(2
*u)*a**2*pet*r*rp*w**3 + 16*e**(2*u
)*a**2*pet*w**2*wp - 2*e**(2*u)*pet
*pp**2*up*w - e**(2*u)*pet*pp**2*wp +
16*up*w*wp**2)/(8*w**3)$

W3:=(2*e**(2*u)*a**2*pet*( - 4*e**(2*u)*p**
2*r**2*up*w + e**(2*u)*p**2*r**2*wp
 - 2*e**(2*u)*p**2*r*rp*w - 2*e**(2
*u)*p*pp*r**2*w - 4*r**4*up*w**3
 - 2*r**4*w**2*wp - 8*r**3*rp*w**3
+ 8*r**2*up*w**3 + 4*r**2*w**2*wp +
 8*r*rp*w**3 - 4*up*w**3 - 2*w**2*
wp))/w**2$

R3:=(2*e**(4*u)*a**2*p**2*pet*r**2*rp*w + 4
*e**(4*u)*a**2*p**2*r*up*w - 3*e**(4*u
)*a**2*p**2*r*wp + e**(4*u)*a**2*p
**2*rp*w + 2*e**(4*u)*a**2*p*pp*r*w +
4*e**(2*u)*a**2*pet*r**4*rp*w**3 - 8*e
**(2*u)*a**2*pet*r**2*rp*w**3 + 4*e**(
2*u)*a**2*pet*rp*w**3 + 8*e**(2*u)*
a**2*r**3*up*w**3 - 4*e**(2*u)*a**2*r
**3*w**2*wp + 12*e**(2*u)*a**2*r**2*rp
*w**3 - 8*e**(2*u)*a**2*r*up*w**3 + 4*
e**(2*u)*a**2*r*w**2*wp - 4*e**(2*u)*a
**2*rp*w**3 + 2*rp*w*wp**2)/w**3$

P3:=(- 8*e**(4*u)*a**2*p**2*pet*pp*r**2 +
32*e**(2*u)*a**2*p*r**2*w*wp + 64*e**(
2*u)*a**2*p*r*rp*w**2 + 32*e**(2*u)
*a**2*pp*r**2*w**2 - e**(2*u)*pet*pp**
3 + 16*pp*up**2*w**2 - 8*pp*up*w*wp)/(
4*w**2)$

U4:= << 45 lines >>

W4:= << 67 lines >>

R4:= << 45 lines >>

P4:= << 28 lines >>

U5:= << 119 lines >>

W5:= << 172 lines >>

R5:= << 131 lines >>

P5:= << 98 lines >>

% Total of about 750 lines of code in above
%    format of about 50 characters per line

% on gcd, factor;
on factor;
on gcd;
gentranout "theworks.trn";
optimize {
upp :=: upp, u3 :=: u3, u4 :=: u4, u5 :=: u5,
wpp :=: wpp, w3 :=: w3, w4 :=: w4, w5 :=: w5,
ppp :=: ppp, p3 :=: p3, p4 :=: p4, p5 :=: p5,
rpp :=: rpp, r3 :=: r3, r4 :=: r4, r5 :=: r5
} ;
gentranshut "theworks.trn";
end;
```

After running the above through REDUCE, and thus SCOPE, the resulting C source code consists of nearly 600 lines of code, with 16 assignment statements for the variables of interest, and 414 assignments to intermediate variables, all of the form "G" followed by an integer. It is impractical to show the many lines of code here, so we show below excerpts from the code produced, with the special notation that an assignment to an intermediate variable is preceded by the count of the number of times that value is actually used in the subsequent code. Some of the assignments to the variables of interest are shown, while assignments which produce many lines of code are simply replaced by the count of the number of lines (of roughly the width shown here) in the actual assignment.

```
{
21:   G2=exp(2*u);
10:   G12=exp(6*u);
18:   G24=w*w;
18:   G30=G46*w;
4:    G53=G286*G132;
6:    G54=G285*G236;
4:    G57=G245*G245*G106;
5:    G64=G306*G107;
7:    G80=G285*G162*wp;
5:    G181=G393*rp;
5:    G202=wp*w;
4:    G285=rp*up;
14:   G486=a*a;
2:    G543=G552+64.0*G140*wp-(192.0*G251*wp)+96.0
       *G123-(4.0*G377)-(448.0*G118*pet)+704.0*G146
       -(320.0*G149);
2:    G544=2.0*(G119+6.0*G466*G119+G316+4.0*(-(G496
       *G118)-G312)-(G498*G58))-(G368*G210);
2:    G545=2.0*G493*G57-(G466*G77)+4.0*G385*G202
       +16.0*(G250*G181-(G374*G373))-G77;
2:    G548=128.0*(G334*G226-(G266*G111)+G24*G90);
2:    G549=16.0*(G498*G71-(2.0*G466*G71)
       +G462*G71)-(G498*G76);
2:    G550=2.0*G264*G127-(G220*wp)
       -(G385*w)-(G370*G233);

upp=(4.0*G555+G215-G592)/(8.0*G24);
u3=(4.0*G553+16.0*(G293+G346-G550)
   -(G215*wp)-G591)/(8.0*G46);
u4=(4.0*G548+24.0*G547-(32.0*G545)-G214+160.0
   *G103-(448.0*G437)-(512.0*G205)+16.0*(-G549
   -G546)+128.0*(G402+G428-G544)+8.0*(G543+G591
   *wp)+G513*(192.0*(2.0*G127-G298-G207))+12.0
   *G495*G215)/(32.0*G30);
u5= << 20 lines >>
wpp=(G555-G590)/w;
w3=(G553+4.0*(G550+4.0*G208-G289+2.0
   *(-G183-G187-G346))+G590*wp)/G24;
w4=(G548+G547+G543+2.0*(G549+G545+4.0*G109
   +12.0*G519+2.0*G103-(32.0*G428)-(120.0*G205)
   +16.0*(G544-G394)+24.0*(G437-G402))+G594
   *G123+G597*G219-(16.0*G229*wp)-(4.0*G590
   *G495)+G252*G201-(8.0*G496*G94)-G563)/G596;
w5= << 18 lines >>
```

```
ppp=(G395-(2.0*G217*pp)+8.0*G258*G128)/w;
p3=(pp*(-G592-(8.0*G247))+G209*(G570+G569)
   -(G474*G2)+64.0*G268*G144+G569*G202*p)/(4.0*G24);
p4=(4.0*(8.0*(G188-G145)+4.0*G585*G369+16.0*G299
   *G185+10.0*G571*G237+G557*wp+G293*G213)+r*(64.0
   *G260-(8.0*G387))+G12*(64.0*(G138-G100))-(8.0*
   G560*G68)+G173*(4.0*G217-wp)+G206*(G599*G24
   -G576)+G496*(128.0*G237-(64.0*G120*G7))+G598
   *pp-(G589*G466)-(16.0*G256*G127)+G595*G326
   -(4.0*G386*G201)-(20.0*G325*G84)-(256.0*G301
   *G120)-(64.0*G197*G122)-G589)/(G596*pet);
p5= << 36 lines >>
rpp=(G586+G391-(G202*rp)-(4.0*G302*G167))/G24;
r3=(G585+12.0*G86+G55+wp*(-G586-(3.0*G391))
   +rp*(2.0*G262-(4.0*G153))+G167*(4.0*(G147
   +G264*G2)+2.0*G366*w)-(8.0*G140*r)+G600*G87
   -(G571*G86)+4.0*G286*G207+2.0*G239*G55)/G46;
r4= << 15 lines >>
r5= << 44 lines >>
}
```

This C code is significantly more compact and efficient than the input expressions, as evidenced by the following statistics produced by SCOPE in the process of optimization.

```
Number of operations in the input is:
    Number of (+,-)-operations : 640
    Number of (*)-operations : 4786
    Number of integer exponentiations : 2226
    Number of other operations : 654

Number of operations after optimization is:
    Number of (+,-)-operations : 550
    Number of (*)-operations : 1156
    Number of integer exponentiations : 0
    Number of other operations : 20
```

The improvement is dramatic, particularly since all integer exponentiations have been removed (none is available in C!), and the number of multiplications has been reduced to about 24 % of the original number. The total number of operations has been reducd to about 21 %, yielding a five fold speed increase, ignoring the fact that the relative slowness of floating multiplies, compared to additions, makes the increase even more dramatic.

It is instructive to examine the code closely to see the source of much of the optimization. Each of the local variables, here of the form "G...", is used at least twice once given a value. In a number of cases common subexpressions are used more than twice, with a resulting significant optimization. While a good human programmer can optimize small expressions, the scale of vision required even in this example is beyond the capabilities of most programmers.

6 Code Production Correctness

While the results of using SCOPE, or a similar package, to produce optimal code are quite impressive, especially for larger problems, it is important to sound a note of caution. As in any complicated piece of software, there is a significant possibility of programming error, either by faulty design, or by accidentally entering incorrect source. This became an important issue in the work described here, when owing to the complexity of the resulting code, we decided that it would reassuring to do a careful check of the correctness of the final production C code.

Our procedure was very simple and involved a direct transformation of the C expression sequence directly into REDUCE source code. This was then read into REDUCE, along with the original expressions that were supplied to the SCOPE process. Much to our surprise, on computing differences between the formal input and the resulting C code, which should have simplified to zero, we found a few small, but irreducible differences remaining.

The impact of the small differences could be transformed away, by simply adding back the differences to the final C expressions, which would then produce agreement with the original input to SCOPE. Since the differences always involved only a small number of terms when compared to the total number of terms in each expression, the loss of optimization was very small indeed. If the error had not been discovered, as described, the net impact could have been quite disastrous, since the essence of the problem we were studying could be seriously altered, especially in asymptotic properties by a missing term, if it happened to contribute to that aspect of the structure.

Thus, we would urge any user of such packages to always take the extra step of simply converting the resulting C, FORTRAN, or similar code back into a form readable by the original algebraic software package, and compute differences between the intended expressions and the resulting expressions that will be passed onto the final compiler.

We hasten to add that the SCOPE group found the error in the distributed version of SCOPE (in REDUCE 3.4) after we pointed out the problem to them.

To show the care that must be used in using such packages, we include below a simple case where the error occurred, where in this case the error manifested itself by a simple change in the FACTOR flag within REDUCE.

```
===== Input file for REDUCE 3.4
load_package scope;
on priall;
optlang!*:='C;
on factor;
on gcd;
u2 := (pet/16)*(p2^2*e^(2*u0) - 16*e^(-2*u0))$
w3 := -2*pet*(r1^2 + 2*e^(-2*u0))$
r3 := r1*(3*pet*p2^2*e^(2*u0) + 32*pet*r1^2
        + 24*p2 +16*(pet - 6)*e^(-2*u0))/32$
p4 := (16*pet*p2*e^(-2*u0) + 192*r1^2*e^(-2*u0)
        - 3*pet*p2^3*e^(2*u0) - 16*pet*p2*r1^2)/8$
% the OFF FACTOR output file ---
gentranout "badscope.nof";
off factor;
optimize { n_p4 :=: p4, n_u2 :=: u2,
        n_w3 :=: w3, n_r3 :=: r3 } iname N;
gentranshut "badscope.nof";
% the ON FACTOR output file ---
gentranout "badscope.fac";
on factor;
optimize { f_p4 :=: p4, f_u2 :=: u2,
        f_w3 :=: w3, f_r3 :=: r3 } iname F;
gentranshut "badscope.fac";
end;
===== Output File "badscope.nof" ---
{
    N4=exp(2*U0);
    N21=R1*R1;
    N16=N21*N4*PET;
    N15=PET*exp(4*U0)*P2*P2;
    N22=3.0*N15;
    N23=16.0*PET;
```

```
    N_P4=(192.0*N21+P2*(N23-N22-(16.0*N16)))
            /(8.0*N4);
    N_U2=(N15-N23)/(16.0*N4);
    N_W3=(-(2.0*N16)-(4.0*PET))/N4;
    N_R3=(R1*(N23+N22+32.0*N16+24.0*N4*P2)
            -(96.0*R1))/(32.0*N4);
}
===== Output File "badscope.fac" ---
{
    F4=exp(2*U0);
    F15=F4*P2;
    F18=R1*R1;
    F16=F18*PET;
    F19=P2*PET;
    F20=3.0*F19*P2;
    F_P4=(192.0*F18+16.0*F19-(16.0*F16*F15)
            +F20*P2*exp(4*U0))/(8.0*F4);
    F_U2=(PET*(F15-4.0)*(F15+4.0))/(16.0*F4);
    F_W3=-(PET*(4.0+2.0*F18*F4))/F4;
    F_R3=(R1*(16.0*PET+F4*(32.0*F16+24.0*P2
            +F20*F4)-96.0))/(32.0*F4);
}
===== Reduce test file generated
        directly from above two files ---
t_u := f_u2 - n_u2;
t_w := f_w3 - n_w3;
t_r := f_r3 - n_r3;
t_p := f_p4 - n_p4;
end;
===== These should all be zero -- one is not !
T_U := 0
T_W := 0
T_R := 0
          2*U0       3
        3*E    *PET*P2
T_P := ----------------
                4
```

Hence, the output generated by scope differs in a real way, depending on the FACTOR switch setting. The result with OFF FACTOR is correct, while the ON FACTOR result is quite incorrect! The difference arises due to a single sign error on the term with numeric coefficient 3. The error is indicated below:

```
Wrong expression:
F_P4=(192.0*F18+16.0*F19-(16.0*F16*F15)
    +F20*P2*exp(4*U0))/(8.0*F4);

Correct expression:
F_P4=(192.0*F18+16.0*F19-(16.0*F16*F15)
    -F20*P2*exp(4*U0))/(8.0*F4);
```

7 Conclusions

We have considered a typical problem involving a considerable mix of a formal theoretical nature at the outset, where packages such as REDTEN running in REDUCE can be used to develop the differential equations describing a physical system. The difficulty of solving these equations in closed form leads to the requirement to produce good numerical code for their solution, and in this case, we were led to the Taylor series method since it yields not only the direct solution, but in addition significant information about the metric functions and their derivatives at each point in spacetime. The actual numerical code is sufficiently complicated to make automatic code generation desirable, and once checked with the algebraic system, we can have strong confidence in the correctness of the production numerical code.

With the development of good code generation and optimization packages, such as SCOPE, with the support of other good packages, such as are available in REDUCE and similar systems, it is possible to raise the level of the programming language from the usual FORTRAN or C level, to the formal algebraic level of REDUCE, MAPLE, and other similar packages. From the point of view of the physical scientist or engineer, this represents a significant advance. Unfortunately it will take a reasonably long time for this to become the standard mode of operation in these areas. The rapid spread of the use of algebraic computing will assist in this evolution, although a significant number of users of algebraic systems restrict their use to something similar to the use of a desk calculator.

In an epoch where the effort put into the development of excellent compilers for RISC machines is so important, it is appropriate to view packages such as SCOPE and its competitors as high level front ends for compilers. This is particularly important when considered from the point of view of global optimization, since these packages are capable of using knowledge of the algebraic structure across a whole family of related expressions. This is clearly far beyond many of the present so-called highly optimizing compilers designed for modern RISC machines. Many of these are so ignorant of possible algebraic optimizations, that the direct product of two square roots remains unchanged, with two calls to the square root routine, even at the most extreme level of optimization. The capabilities of a good algebraic system in generating code can go much farther than this, as shown even in the fairly modest example presented here.

8 Acknowledgements

It is a pleasure to thank J. Harper, F. Marleau, E. Shaver, and G. Starkman for stimulating discussion in this work, and to acknowledge the support of the Canadian Natural Sciences and Engineering Research Council for their financial support. In addition, the quick response of J. van Hulzen and his SCOPE group in discovering the problem with the SCOPE distributed with REDUCE 3.4 was of great help to us.

References

[1] J. Harper and C. Dyer, *The REDTEN User's Manual, 1986, 1994.*

[2] C. Dyer, F. Marleau, E. Shaver, submitted to *Phys. Rev. D.*

[3] R. Schopf, in *REDUCE 3.4 Miscillaneous Documentation.*

[4] A. Barnes, J. Padget, in *REDUCE 3.4 Miscillaneous Documentation.*

[5] B. Gates, M. Dewar, in *REDUCE 3.4 Miscillaneous Documentation.*

[6] J. van Hulzen, in *REDUCE 3.4 Miscillaneous Documentation.*