GRTensorII

GRTensorII Release 1.50
For MapleV Releases 3 and 4

C. Calculating tensor components

Peter Musgrave
Denis Pollney
Kayll Lake

July 1996

Contents

1 Tensor notation **C2**

2 Calculating tensor components **C6**

3 Displaying tensor components **C9**

4 Modifying tensor components **C12**

5 Simplification strategies **C15**

6 Accessing tensor components **C16**

7 Standard object library **C17**

Queen's University at Kingston, Ontario

The goal of the GRTensorII program is the calculation of components of indexed objects, in particular tensors. This booklet describes the GRTensorII notation by which tensors are referenced in all calculation commands, as well as the functioning of commands related to determining components of tensors and expressing the output in a convenient form.

The main method by which calculation is accomplished in GRTensorII is through the `grcalc()` command. Simple calculation is often not the last step, however, in producing interpretable output. Often large terms result in individual tensor components which need to be simplified before they can be read. Of greater concern, the presence of large terms in intermediate objects can seriously affect the ability to carry out later calculations based on these objects. In such cases simplification at crucial points becomes necessary if any progress is to be made at all. GRTensorII possesses a number of commands allowing the manipulation of components of tensors, most notably `gralter()` and `grmap()`.

The commands described in this booklet assume that a spacetime (metric or basis) has been loaded (or created) for the current GRTensorII session. For a description of how to do this, see Booklet *B: Specifying spacetimes.*

1 Tensor notation

The commands provided with GRTensorII are designed to calculate and operate on tensors with any given index configuration. An example of such a command is the `grcalc()` command which requests the calculation of the tensors named in its arguments. For instance,

```
> grcalc ( R(dn,dn), R(up,dn,dn,dn) ):
```

requests the calculation of the covariant Ricci tensor (R_{ab}) and Riemann tensor in standard form ($R^a{}_{bcd}$). This section provides a description of the notation by which tensors are referred to in GRTensorII commands.[1]

1.1 Tensor names and indices

In general, GRTensorII commands which act on tensors take the form:

commandName (*tensorSeq*, [*other_Arguments*])

Here the *tensorSeq* is a sequence of GRTensorII objects each of which has the following form:

tensorName (*indexSeq*)

The *tensorName* is a string specifying a name from the GRTensorII library of predefined objects.[2] Examples are:

`R`	– the Riemann tensor (and its index contractions),
`G`	– the Einstein tensor,
`g`	– the metric,
`Chr`	– the Christoffel symbols,
⋮	
etc.	

[1]For more concrete examples of how these commands operate, the reader may wish to skip past this section or examine some of the online examples at the GRTensor home page [1].

[2]Or a user-defined object, as can be created via the methods outlined in Booklet *D: Defining new tensors.*

Section 7 lists objects available in the standard library.

The argument *indexSeq* is a sequence of names giving the index configuration of the tensor. Available index types are:

`up`, `dn`	– covariant/contravariant indices,
`pup`, `pdn`	– partial derivatives,
`cup`, `cdn`	– covariant derivatives,
`bup`, `bdn`	– covariant/contravariant basis indices,
`pbup`, `pbdn`	– partial derivatives in a basis,
`cbup`, `cbdn`	– covariant derivatives in a basis.

The different indices can be divided into two groups: those that refer to components arising from a metric, g_{ab}, and those arising from a spacetime described by a set of basis vectors, ${e_{(a)}}^b$. Either of these types of spacetimes can be created using `makeg()` or loaded into GRTensorII using `qload()`, as described in Booklet *B: Specifying spacetimes.*

The indices take the following meanings:

`up, dn`: These refer to the standard covariant and contravariant indices of tensors calculated from a metric, g_{ab}. The covariant components of the metric itself are referred to as `g(dn,dn)`.

`pup, pdn`: These are partial derivatives of the named tensor. Thus `R(dn,dn,pdn)` refers to the object:

$$R_{ab,c} := \frac{\partial R_{ab}}{\partial x^c},$$

where x^a are the coordinates of the metric.

`cup, cdn`: These specify covariant derivatives of the named tensor. The definition of `R(up,dn,cdn)` is given by:

$$R^a{}_{b;c} := R^a{}_{b,c} - \Gamma^d_{bc} R^a{}_d + \Gamma^a_{bd} R^d{}_c,$$

where Γ^a_{bc} are the Christoffel symbols (see Section 7 for their definition).[3]

`bup, bdn`: These indices refer to the basis components of a tensor. They can only be used if the spacetime has been specified in the form of a set of basis vectors. In this case, the object `R(bdn,bdn)` refers to the object

$$R_{(a)(b)} := R_{cd}{e_{(a)}}^c{e_{(b)}}^d,$$

where ${e_{(a)}}^d$ are the n basis vectors.[4]

`pbup, pbdn`: These are the corresponding partial derivatives in the basis. Thus `R(bdn,bdn,pbdn)` is equivalent to

$$R_{(a)(b),(c)} := \frac{\partial R_{(a)(b)}}{\partial x^d}{e_{(c)}}^d,$$

with x^a the coordinates of the basis.

[3]Here and throughout these booklets we make use of the usual summation convention such that if an index is repeated in both upper and lower positions in a term, then a summation is implied over all values of that index.

[4]Although the definition given above writes the basis form of the Ricci tensor as a transformation of the metric component form, in fact the objects in the standard GRTensorII library have alternate definitions which are optimized for calculation in a basis so that it is not necessary to revert to coordinate components at any point during a calculation.

GRTensorII name	Common representation
`R(dn,dn)`	R_{ab} (Ricci tensor)
`R(dn,dn,dn,dn)`	R_{abcd} (Riemann tensor)
`R(up,dn,dn,dn)`	$R^{a}{}_{bcd}$
`R(dn,dn,pdn)`	$R_{ab,c}$ ($= \partial_c R_{ab}$)
`R(dn,dn,cdn)`	$R_{ab;c}$ ($= \nabla_c R_{ab}$)
`R(bdn,bdn)`	$R_{(a)(b)}$ (basis components of the Ricci tensor)
`R(bdn,bdn,pbdn)`	$R_{(a)(b),(c)}$
`R(bdn,bdn,cbdn)`	$R_{(a)(b);(c)}$

Table 1: Some examples of index specifications in GRTensorII object names.

`cbup`, `cbdn`: These are the covariant derivatives in the basis. The object `R(bup,bdn,cbdn)` is defined by

$$R^{(a)}{}_{(b);(c)} := R^{(a)}{}_{(b),(c)} + \gamma^{(a)}{}_{(b)(d)} R^{(d)}{}_{(c)} - \gamma^{(d)}{}_{(b)(c)} R^{(a)}{}_{(d)},$$

where $\gamma^{(a)}{}_{(b)(c)}$ are the Ricci rotation coefficients. See Booklet *E: Bases and tetrads* for their definition.

If the spacetime has been specified in the form of a basis, then any of the above types of indices can be used, since the metric is easily calculated via

$$g^{ab} := \eta^{(c)(d)} e_{(c)}{}^{a} e_{(d)}{}^{b}.$$

In such cases it is easy to transform between basis and metric components of a tensor simply by requesting a different set of indices.[5] However, if the spacetime is specified in the form of a metric, g_{ab}, then the basis components of tensors can not be referenced unless a basis is first explicitly defined (as can be done, for example, using the `nptetrad()` command described in Booklet *B: Specifying spacetimes*).

Finally, we note that the `grdef()` command (used to define new tensors outside of the standard GRTensorII library) necessarily uses a somewhat different notation from that described in this section to express tensor indices. This command described in Booklet *D: Defining new tensors*.

1.2 Operators

Certain commonly used tensors require extra information (in addition to the background geometry) in order to be calculated. For instance, consider the definitions of the 'electric' and 'magnetic' parts of the Weyl tensor,

$$E_{ab} := C_{acbd} v^c v^d, \qquad H_{ab} := C^{*}_{acbd} v^c v^d.$$

In addition to the Weyl tensor (which can be calculated directly from the metric), they require the specification of a vector field v^a. Naturally, these tensors will have different values if different vector fields are specified.

[5]More exotic index permutations than those specified above are also possible. For instance `R(up,dn,bdn,bdn)` refers to the object

$$R^{a}{}_{b(c)(d)} := R^{a}{}_{bef} e_{(c)}{}^{e} e_{(d)}{}^{d},$$

which has mixed basis and metric indices.

In GRTensorII, certain tensor definitions require the specification of such parameters as arguments to the tensor name. The arguments are placed in square braces '`[]`' before the index list. For instance, assuming a single-index tensor (a vector field) `v(up)` had been defined, the electric and magnetic Weyl tensors would be referenced in GRTensorII using

`E[v](dn,dn)`, and `H[v](dn,dn)`.

Note that in the definitions of the electric and magnetic Weyl tensors assume that the argument is a single index object, and so the index does not have to be specified, only the vector name (in this case '`v`').

A more general example is the d'Alembertian derivative operator, $\Box := \nabla^a\nabla_a$. The d'Alembertian defined by GRTensorII takes a tensor with an arbitrary number of indices as an argument. We could reference $\Box R_{abcd}$ using

`Box[R(dn,dn,dn,dn)]`

Objects involving specifiable arguments were originally defined to allow the use of differential operators in GRTensorII, and so they have come to be called 'operators'. This terminology will be used throughout these booklets. A list of operators in the standard GRTensorII object library is given in Section 7.

1.3 Alternate spacetimes

Any calculations carried out by GRTensorII are done so for a particular background geometry, specified either by a metric or basis, which has been loaded and assigned a name by `qload()` (or created by `makeg()`). Generally, the last spacetime to be loaded is the default spacetime for which all calculations subsequent calculations are carried out.[6] However, it is possible to have loaded a number of different spacetimes into a single session and still calculate components for a particular spacetime even if it is not the default. This is done by specifying the name of the desired spacetime as an argument to the tensor in square braces. For instance, consider the set of commands:[7]

```
> qload ( schw ):
> qload ( kerr ):
> grcalc ( R(dn,dn) ):
> grcalc ( R[schw](dn,dn) ):
```

The first two commands load the two spacetimes `schw` and `kerr`. After the second command, the default spacetime for which subsequent calculations will refer is `kerr`. The first `grcalc()` command requests the calculation of the covariant Ricci tensor. This command automatically assumes that the calculation is requested for the default spacetime, ie. `kerr`. The second `grcalc()` command also calculates the Ricci tensor, however in this case the desired spacetime is given explicitly as an argument to the tensor name. Thus the second command calculates the Ricci tensor for the `schw` spacetime.[8]

[6]The default spacetime can also be switched through use of the `grmetric()` command. See Booklet *B: Specifying spacetimes*.

[7]The `qload()` command is described in Booklet *B: Specifying spacetimes*, and the `grcalc()` command is the subject of Section 2 of this booklet.

[8]In passing, note that the two `grcalc()` commands could also have been executed with the single command:

```
> grcalc ( R(dn,dn), R[schw](dn,dn) ):
```

It is currently not possible to specify alternate spacetimes in this way for GRTensorII operators. For such objects the `grmetric()` command must be used to switch the default background geometry for calculation.

1.4 Composite objects

Certain objects in the standard GRTensorII library have been defined as 'composite objects'. These objects are actually simply aliases which refer to some group of standard objects. Every object in the group will be acted upon when the alias is used. An example is the set of Newman-Penrose spin coefficients. These objects can be reference individually using the object names `NPsigma`, `NPepsilon`, etc. They can also be referenced as a group using the name `NPSpin`. Thus the single command

```
> grcalc ( NPSpin ):
```

is equivalent to the set of commands

```
> grcalc ( NPsigma ):
> grcalc ( NPepsilon ):
⋮
```
etc.

which would have to be issued for each of the twelve spin coefficients.

Other notable examples of composite objects include the Newman-Penrose curvature components (`WeylSc` and `RicciSc`, described in Booklet *E: Bases and tetrads*) and the scalar curvature invariants (see `?grt_invars` and Section 7 of this booklet).

2 Calculating tensor components

GRTensorII's *raison d'etre*, is embodied in a single command, `grcalc()`. This command is responsible for the calculation of the components of any tensor, scalar, or operator defined for GRTensorII. Its usage is as follows:

`grcalc (` *objectSeq* `)`

objectSeq – A sequence of GRTensorII indexed object names.

Example: `> grcalc ( R(up,dn,dn,dn), R(dn,dn), Ricciscalar ):`

In order to perform any calculation with `grcalc()`, a background geometry must have been previously loaded into the current session (see Booklet *B: Specifying spacetimes*), either as a metric or set of basis vectors. Calculations of either basis or metric tensor components can be carried out by specifying the type of index, as outlined in the previous Section 1.3.

The results of a calculation are not displayed automatically because individual terms can often be inconveniently large before they are simplified using `gralter()` (see Section 4). A request to display the results of a `grcalc()` command must be given explicitly using the command `grdisplay()` (Section 3).

Multiple objects can be calculated at once by separating them with commas in the argument to `grcalc()`. The underscore character, '_' can be used as a short-cut if the object arguments to a GRTensorII command are to be re-used in succession. For instance, in the pair of commands

```
> grcalc ( R(up,dn), Ricciscalar ):
> grdisplay ( _ ):
```

the underscore character instructs `grdisplay()` to substitute the arguments of the previous `grcalc()` command (namely `R(up,dn)` and `Ricciscalar`) into its own argument list.[9]

Calculation of intermediate objects: The `grcalc()` command calculates the requested objects for the default metric or for any metric specified in square brackets after the tensor name (see Section 1.3). The internal definition of each tensor also contains a list of tensors which are required by that definition. Any tensors in this list (for example, the Christoffel symbols in the case of any of the curvature tensors) will also be calculated automatically. If any of these objects have already been calculated for the spacetime in question, the results of the previous calculation are used rather than re-calculating the objects.

Derivatives: The derivatives (partial or covariant) of any tensor or scalar can be calculated by appending the indices `pdn` or `cdn` to the regular index list of the object. Thus the covariant derivative of the Ricci tensor, $R_{ab;c}$, could be calculated using the command

```
> grcalc ( R(dn,dn,cdn) ):
```

In the case of curvature tensors defined on a basis, the indices `pbdn` and `cbdn` should be used instead. See the previous section for definitions of the derivative indices.

Operators: Operators defined in the GRTensorII standard library can also be applied to tensors. The arguments of the operator are specified in square braces before the index list. For instance, the d'Alembertian of the Ricci tensor could be calculated using the command

```
> grcalc ( Box[ R(dn,dn) ] ):
```

See Section 1.2 for a more complete description of how operators are referenced in GRTensorII commands. Section 7 provides a list of operators defined by the standard library.

2.1 Calculating individual components

An occasional difficulty when working with complicated spacetimes is that individual components of curvature tensors can expand to a size which MapleV is unable to handle, resulting in an

[9]Another short-hand alternative has been included, because the `grcalc()` and `grdisplay()` commands are so often used together. The command `grcalcd()`, is an alias for the combined `grcalc()` and `grdisplay()` applied in succession. Thus the given command pair could be replaced by the single command

```
> grcalcd ( R(up,dn), Ricciscalar ):
```

'`Object too large`' error. It may happen that this error occurs for only a few components of a tensor, while many others are well within the limitations of the computer algebra system, and sometimes in such cases useful information about the tensor can be inferred even if all of the components are not explicitly calculated.[10]

The command `grcalc1()` exists to perform the calculation of a single component of a tensor:

`grcalc1 (` *object*`,` *indexList* `)`

object – The name of the object to be calculated.

indexList – A MapleV list whose entries specify the component to be calculated, either by indicating the coordinate names or numbers.

Example: `> grcalc1 ( R(dn,dn,dn,dn), [r,r,theta,phi] ):`

Note that since only one component has been calculated, GRTensorII will not recognize the entire object as being calculated. Thus, the result of the calculation cannot be displayed using the `grdisplay()` command. Instead, the component can be accessed using `grcomponent()` (described in Section 6, below), as in the command sequence:

```
> grcalc1 ( R(dn,dn), [1,1] ):
> f(x,y,z,t) := grcomponent ( R(dn,dn), [1,1] ):
```

(where we've also demonstrated the use of coordinate numbers as an alternative to coordinate names.)

2.2 Clearing calculations

Once the components of a tensor are calculated for a given metric the results are saved and used in any subsequent calculations involving the tensor. Further calls to calculate the same object will return the already determined components. In order to clear the results of a tensor calculation, the command `grclear()` can be used.

[10]See the Tomimatsu-Sato worksheet, `tosa.ms`, available from the GRTensorII world-wide-web page [1].

`grclear` (*objectSeq*)

objectSeq – This argument can be either a sequence of GRTensorII objects or one of the special parameters:

results – Clears all of the GRTensorII objects which have been calculated for the current default spacetime. The default spacetime (ie. metric and/or basis vectors) remains initialized.

metric – Clears all of the objects calculated for the current spacetime, including the metric components `g(dn,dn)`.

spacetime – Clears all of the objects calculated for the current default spacetime as well as the components of the metric and basis vectors, if assigned.

If either of the three special parameters are used as the *objectSeq* argument, then a prompt asks the user to confirm the use of `grclear()`.

Example: `> grclear ( results ):`

3 Displaying tensor components

The command `grdisplay()` can be used to display the components of GRTensorII objects which have previously been calculated for a particular spacetime.

`grdisplay` (*objectSeq*)

objectSeq – A sequence of names of previously calculated objects.

Example: `> grdisplay ( R(bdn,bdn) ):`

The `grdisplay()` command displays the components of all of the tensors named in its argument.[11] Only the non-zero components are displayed, and only the independent components up to index symmetries (thus only the components in the upper diagonal of a symmetric 2-tensor would be displayed).

Components whose size exceeds the value of the `grOptionDisplaySize` variable will have only their length displayed rather than their value. This limit can be increased by assigning a larger value to this variable. It is preferable to attempt to reduce the size of the components by applying simplifications via `gralter()` or `grmap()` as described in Section 4, below.

For 1- and 2-index objects, the output will be in the form of a matrix unless the size of one of the individual tensor components (as measured in MapleV words) exceeds the value of the `grOptionTermSize` variable. If so, then the regular display of individual components is used.

[11] Recall that if `grcalc()` or `gralter()` have been used immediately prior to this command, then the short-cut underscore character can be used to indicate the arguments of the previous command are to be carried over (see the discussion in Section 2).

The display of coordinate names over coordinate numbers can be toggled using the global variable `grOptionCoordNames`.

For descriptions of the `grOption` variables, see Booklet *F: Installation and setup* or the `?groptions` online help page.

3.1 Derivative aliases

Large numbers of derivatives in an expression can confuse the appearance of the output. It is often desirable to express partial derivatives in a variable by a special symbol or abbreviation. A set of commands are included in GRTensorII to perform such substitutions to each component of a result returned by `grcalc()` with the aim of producing more readable output.

The most useful of these commands is `autoAlias()`, which specifies shorthand notation for partial derivatives to be used throughout the session.

`autoAlias (` *expressionName* `)`

expressionName – The name of an expression which contains functions whose derivatives are to be aliased.

Example: `> autoAlias ( diff(g(r,t),t)/r + diff(f(r,t),r)/t ):`

The `autoAlias()` command automatically creates aliases for partial derivatives functions named in its argument. The form of the shorthand notation is to represent derivatives in a coordinate by placing the coordinate in a subscript. Thus, for example

$$\frac{\partial f(r,t)}{\partial r} \qquad \text{becomes} \qquad f_r.$$

The `autoAlias()` command is given a single expression as an argument. It automatically creates derivative aliases (up to third order) for each function contained in the expression. Thus, if $f(r,t)$ is defined by

$$f(r,t) := h(r,t) + \frac{\partial h(r,t)}{\partial r} + \frac{\partial g(r,t)}{\partial t}$$

and is given as an argument to `autoAlias()`, then derivative aliases will be automatically created for the functions $h(r,t)$ and $g(r,t)$. Derivative aliases created with `autoAlias()` will be active for the entire MapleV session.

Note that in order to apply `autoAlias()` to all functions contained in the components of a particular tensor, the command `gralter()` (or `grmap()`) must be used, as in:

```
> gralter ( R(dn,dn), autoAlias ):
```

This command ensures that derivatives of all functions in all component of the Ricci tensor are assigned an alias of the form specified above. (See Section 4 for a description of the `gralter()` command.)

The command `diffAlias()` can be used to perform a similar task. The main differences are that `diffAlias()` allows the user to specify exactly which functions are to be aliased.

`diffAlias (` *varList*, *functionList* `)`[12]

varList – A MapleV list of variables (coordinates) for which the shorthand alias is to be applied (eg. `[r,theta]`).

functionList – A set of names of functions in the above specified variables for which the shorthand is to be applied (eg. `[f, g]`).

Example: `> diffAlias ( [r,t], [f,g] ):`

As with `autoAlias()`, the aliases take the form of subscripts of the relevant variable. Returning to the above example, the command

```
> diffAlias ( [r,t], [h,g] ):
```

would be equivalent to the command

```
> autoAlias ( f(r,t) ):
```

`diffAalias()` commands can also be placed in metric files (see Booklet *B: Specifying spacetimes*) to assign the aliases automatically when the metric is loaded.

The final form of derivative aliasing available with GRTensorII allows more general representations of the derivatives than either the `diffAlias()` or `autoAlias()` commands. The `grDalias()` command allows users to specify a character representation of the derivative in a particular coordinate, such as an apostrophe (`'`). The syntax of the command is:

`grDalias (` *function*, *var1*, *str1*, *var2*, *str2*, ... `)`

function – A function for which derivative aliases are to be created.

var – The variables on which the *function* is dependent.

str – Strings which should represent derivatives of the *function* in terms of the preceding named variable.

Example: `> grDalias ( f(r,t), r, `'`, t, `*` ):`

The `grDalias()` command replaces derivatives in the listed variables with a corresponding string which is specified by the user. Thus, after issuing the command

```
> grDalias ( f(r,t), r, `'`, t, `*` ):
```

[12]The code for `diffAlias()` was contributed by Roberto Sussman.

the functions

$$\frac{\partial f(r,t)}{\partial r}, \quad \frac{\partial^2 f(r,t)}{\partial r^2}, \quad \frac{\partial f(r,t)}{\partial t}, \quad \frac{\partial^2 f(r,t)}{\partial t^2}, \quad \frac{\partial^2 f(r,t)}{\partial r \partial t},$$

would appear, respectively, as

$$f', \qquad f'', \qquad f^{*}, \qquad f^{**}, \qquad f'^{*}.$$

`grDalias()` commands can also be placed in metric files (see Booklet *B: Specifying spacetimes*) to assign the aliases automatically when the metric is loaded.

4 Modifying tensor components

The large number of summations involved in the calculation of curvature tensors often results in individual components consisting of enormous numbers of terms. Only with the aid of symbolic simplification functions can one hope to extract useful information from the expressions.

Two functions are provided with GRTensorII which allow the application of MapleV functions to individual tensor components: `gralter()` and the more general `grmap()`.

`gralter (` *objectSeq*`, [`*function1*`], [`*function2*`], ... )`

objectSeq – A sequence of GRTensorII indexed object names.

function – Single-argument functions to be applied in turn to each component of the objects named in the *objectSeq* argument.

Example: `> gralter ( R(dn,dn), expand, factor ):`

The `gralter()` command is the standard means of applying simplification routines to GRTensorII object components. The command recognizes any of the following strings in its *function* arguments:

1. `simplify` – Applies the MapleV `simplify()` command to each component of the named objects. See `?simplify`.
2. `trig` – Applies trigonometric simplification to each component. See `?simplify[trig]`.
3. `power` – Simplifies terms containing exponents, exponentials, and logarithms. See `?simplify[power]`.
4. `hypergeom` – Simplifies terms containing hypergeometric functions. See `?simplify[hypergeom]`.
5. `radical` – Converts radicals, logarithms, and exponentials to their MapleV canonical form. See `?simplify[radical]`.
6. `expand` – Applies the routine `expand()` to expand individual terms in each component. See `?expand`.
7. `factor` – Applies the routine `factor()` to factorize each component. See `?factor`.

8. `normal` – Applies the routine `normal()` to convert each term to its MapleV 'factored normal form'. See `?normal`.

9. `sort` – Applies the routine `sort()` to sort the terms of polynomials into descending order. See `?sort`.

10. `sqrt` – Simplifies expressions involving square roots of symbolic quantities. This has the effect of forcing MapleV to recognize that, for instance, $\sqrt{r^2} = r$. See `?simplify[sqrt]`.

11. `trigsin` – Applies trigonometric simplification (as Option 2, above), however in this case biased towards introducing sine functions.

12. `cons` – This option will apply any constraint equations that have been specified with the metric. See Booklet *B: Specifying spacetimes*.

13. `consr` – Like the previous option, this option applies any constraints on the spacetime. However, in this case the constraints are replied repeatedly until there is no change in the expression to which they are being applied.

14. *other* – Any other single-argument MapleV function can be specified (eg. `radsimp()`). The function will be applied to each component of the listed GRTensorII objects.

Note that each of the above options are numbered, and in fact any of the numbers (except 14) can be used as a substitute for the corresponding names. Thus,

```
> gralter ( R(dn,dn), 6, 7 ):
```

is equivalent to the command

```
> gralter ( R(dn,dn), expand, factor):
```

Alternatively, the user may choose not to specify any form of simplification in the command arguments. In this case, a menu of the above options is presented and the user asked to choose from the above numbers. In this case Option 14 is available, and the user will be prompted for the MapleV command which is to be applied.

The `gralter()` command can apply any single-argument MapleV function to the components of a tensor. In order to make use of MapleV functions which possess more than one argument, the `grmap()` command can be used. This command can apply any MapleV function to each component of a GRTensorII object.

As an example, consider the case in which one wishes to give a particular value to some constant, say $Q = 0$, in each component of R_{ab}. In order to perform this substitution for a single function, we would use the standard MapleV command `subs()`, as in

```
> subs ( Q=0, F(x,y,z) ):
```

where $F(x, y, z)$ is some complicated function containing Q as a parameter. Note that in this case `subs()` takes two arguments, and thus could not be applied to `R(dn,dn)` using `gralter()`. We could, however, use `grmap()` to perform the substitution for each component of `R(dn,dn)` as follows:

```
> grmap ( R(dn,dn), subs, Q=0, 'x' ):
```

Here the first argument specifies the object to which the command is to be applied, the second is the command itself (‘`subs`’), and the third and following arguments specify the arguments that should be passed to the given function. A place-holder, `'x'`, is used among these arguments to indicated where the tensor component is to be inserted in the command. As the above command is executed the components R_{11}, R_{12}, ..., are substituted into the location of x in the command:

```
> subs ( Q=0, x ):
```

The general format of the `grmap()` command is as follows:

`grmap (` *objectSeq*`,` *function*`,` *fnArg1*`, ... , 'x', ... ,` *fnArgN* `)`

objectSeq – A GRTensorII indexed object name, or sequence of object names.

function – The name of a function to be applied to each component of a tensor.

fnArg1 ... *fnArgN* – The arguments of the *function* named above.

`'x'` – A place-holder indicating where the components of the tensors listed in *objectSeq* are to be slotted into the arguments of the *function*.

Example: `> grmap ( R(dn,dn), Ricciscalar, subs, b(r)=B, c(r)=C, 'x' ):`

Finally, we note that simplifications carried out by the `gralter()` and `grmap()` command are carried out only after the use of `grcalc()` to calculate the components of a particular tensor. In general, however, the calculation of a tensor requires the calculation of a number of intermediate objects, which might also benefit from some simplification in order to allow the later calculations to proceed more smoothly.

For example, if one requests a calculation of the Riemann tensor immediately after loading a metric then the program must first calculate the Christoffel symbols and then perform summations and take derivatives of their components. For very complicated metrics, if the Christoffel symbols are left in ‘unsimplified’ form, this can cause problems for the subsequent calculation of the Riemann tensor.

In this trivial example the most obvious solution is first to calculate and simplify the Christoffel symbols, then proceed to calculate the Riemann tensor. However, occasionally one comes across calculations with large numbers of such intermediate calculations in which it might be desirable to apply some form of simplification universally to each. The `grcalcalter()` command can perform this operation automatically, effectively performing a `grcalc()` followed by a `gralter()` command for each intermediate object in a calculation.

`grcalcalter ( `*objectSeq*`, [`*function1*`], [`*function2*`], ... )`

objectSeq – A sequence of GRTensorII indexed object names.

function – Single-argument functions to be applied in turn to each component of the objects named in the *objectSeq* argument.

Example: `> grcalcalter ( R(dn,dn), expand, factor ):`

The arguments to `grcalcalter()` are identical to those of `gralter()` given above. Simplification routines can be specified either by name or number, and if no simplification routine is specified a menu will be presented and the user prompted to supply one. The simplification routine which is chosen will then be applied to the calculation of each intermediate object required of the calculation of objects listed in *objectSeq*.

The `grcalcalter()` command should, however, be used with caution. Only rarely is it helpful to apply powerful simplifications universally. More often this will result in a large number of redundant attempts at simplification which increase the calculation time without noticeable benefit. These matters are the topic of the next section.

5 Simplification strategies

A brief experience with tensor calculation using computers makes clear the importance of selecting a correct simplification strategy for calculations. By choosing not to perform a simplification at a crucial step, one risks allowing components to become too large to be tractable by the computer algebra package. Alternatively, simplifying at each stage of a calculation often results in needless or redundant expansions and contractions of terms which can use up a great deal of time with little or no benefit.

Unfortunately, the choice of which simplification routines should be applied, and at what stages, is a fairly non-algorithmic problem, and thus difficult to automate in a software package. A number of widely held generalizations, such as the idea that simpler components will result from the proper use of tetrads, have proven to be inaccurate in practical situations.[13]

Though the problem of finding the best simplification strategy can be considered to be non-algorithmic, it certainly is often solvable. If a number of general rules are kept in mind while performing a calculation, it is possible within a reasonable amount of time to determine a solution to a given problem, whenever such a solution exists at all given the memory limitations of the computer algebra system/platform. Summarized here are some general rules of simplification appropriate to GRTensorII running under MapleV.

- There should be some default simplification procedure applied to every step of a calculation. Failure to do this can make a simple calculation intractable. The MapleV routine `normal()` is the best starting point for the default. If the calculation involves exponentials (e.g. the Bondi metric) the routine `expand()` may be more appropriate. Only if very general functions are involved is it appropriate to consider no default simplification. In general cases this may be the optimal choice. The choice of default simplification can be altered through the use of

[13]For an analysis of the efficiency of different formalisms in calculating tensor components, see [2].

the global variable `grOptionDefaultSimp` (see `?groptions`, or Booklet *F: Installation and setup*) or by using the `grcalcalter()` command.

- Simplification of the metric tensor (or basis components) before further calculation will improve performance.
- In the null tetrad formalism, pre-calculation and further simplification of the spin coefficients (and their complex conjugates) will improve performance only in more complicated cases. The same holds for the Christoffel symbols in the coordinate approach. (Note that these objects represent 'first derivatives' of the metric. Generally, a good point to apply simplifications is after any derivatives or covariant derivatives of intermediate objects are taken.)
- For further simplification after an object has been calculated, the MapleV routine `simplify()` is seldom a good first choice. The routine `expand()` followed by `factor()` is often more appropriate. If the spacetime is sufficiently general there will be no further reduction in component size if `normal()` has been used as default.
- If complicated functions are involved, it can be advantageous to substitute the explicit forms of the functions *after* a more general calculation is completed. For both coordinate and tetrad calculations the removal of trigonometric and like functions via elementary transformations (such as $u(\theta) := \cos\theta$) will often improve performance.
- Experience shows that component calculations most often tend to finish quickly (within a few minutes at most) or not at all. When a calculation is proceeding slowly, it should be halted, the simplification strategy altered, and the worksheet re-executed. (This is more easily done in the 'windows' based interfaces to MapleV than in the text based interface.)

For most situations these general rules will give adequate performance, and reduce the calculation of curvature for even complex spacetimes to an essentially trivial exercise. That is to say, if after applying these methods an answer is still not forthcoming (most likely because of the size of intermediate expressions), then it is quite possible that no answer is attainable given the limitations of the computer algebra platform, and that the problem is not solvable without some more sophisticated analysis.[14]

6 Accessing tensor components

Results obtained by use of the `grcalc()` are stored in a data structure which contains the components of the particular tensors as well as a great deal of other information regarding the tensors including index configuration, index symmetries and calculation algorithms. Generally, the user is not interested in the additional information, and thus the commands `grdisplay()`, `gralter()`, `grmap()`, etc., have been provided to allow users to display and operate on the components of tensors without dealing with the more complicated data structure in which they are stored.

Often, however, it is useful to be able to perform more complicated operations with the components of tensors, in which case the `grmap()` command can become unwieldy. For instance, a user might like to solve the individual Einstein equations calculated for a given metric. It would be most convenient, then, to extract the tensor component values from the GRTensorII data structure and place them in some user defined variables which can be more easily manipulated.

[14] See the Tomimatsu-Sato example, `tosa.ms`, available from the GRTensorII world-wide-web site [1].

The main means of extracting tensor components is through the command `grcomponent()`:

`grcomponent (` *object*`,` *component* `)`

object – The indexed object name of the tensor whose components are to be extracted.

component – A MapleV list specifying the particular component, either by giving the index names (eg. `[t,r,theta,phi]`) or their corresponding numbers (eg. `[1, 2, 3, 4]`). In the case of GRTensorII scalars, this parameter can be omitted, or an empty list, '`[]`', used.

Example: `> f(x) := grcomponent ( R(up,dn,dn,dn), [t,r,r,r] ):`

Unlike most other GRTensorII commands, `grcomponent()` returns a value (a function, usually) which can be assigned to some other MapleV name.

While `grcomponent()` extracts individual components of tensors, the command `grarray()` can be used to place all of the components of an n-index tensor into an MapleV array. For example, using the command

```
> Riem := grarray ( R(dn,dn,dn,dn) ):
```

the components of the covariant Riemann tensor are stored in the array `Riem` and can be accessed as `Riem[1,2,1,2]`, etc. The specification of the `grarray()` command is as follows:

`grarray (` *object* `)`

object – The indexed object from which the array is to be created.

Example: `> Riem := grarray ( R(dn,dn,dn,dn) ):`

For both the `grcomponent()` and `grarray()` commands, it is important that the names to which the extracted components are assigned are not the same as the names of any standard GRTensorII object. Thus, the assignment

```
> R := grarray ( R(dn,dn,dn,dn) ):
```

will result in error messages in later attempts to use `R(dn,dn,dn,dn)` since the name '`R`' has been assigned.

7 Standard object library

A number of commonly used curvature tensors are defined automatically within GRTensorII along with optimized algorithms for their calculation. This section lists the objects (tensors and operators[15]) which can be calculated once a metric g_{ab} is specified (see Booklet *B: Specifying spacetimes*). Objects which are calculated from a set of basis vectors are listed in Booklet *E:*

[15]For a definition of the GRTensorII use of the word 'operator', see Section 1.2.

Bases and tetrads.

In addition to the objects listed in this section, new tensors can be defined using the command `grdef()`. For a description of this command, see Booklet *D: Defining new tensors.*

7.1 Curvature tensors

While only a single index configuration is given in each of the following definitions, alternate configurations as well as their derivatives can be specified via the means described in Section 1.

Note that the definitions given in this section do not necessarily specify the algorithm used for calculation. For instance, generally the Ricci tensor will be calculated directly from the Christoffel symbols, even though the definition below is given in terms of a contraction of the Riemann tensor. Within GRTensorII, algorithms often make a choice of calculation method based on criteria such as which intermediate objects have already been calculated.

GRTensorII name	Definition
`g(dn,dn)`	covariant metric, g_{ab}
`ds`	line element form of the metric
`detg`	metric determinant, $g := \det(g_{ab})$
`dimension`	spacetime dimension,
`sig`	spacetime signature,
`x(up)`	coordinates, x^a
`kdelta(up,dn)`	Kronecker delta, $\delta^a{}_b := \begin{cases} 1, \text{ if } a = b, \\ 0, \text{ otherwise.} \end{cases}$
`LevCS(dn,dn,dn,dn)`[16]	Levi-Civita alternating symbol, $[a,b,c,d] := \begin{cases} 1 \text{ for } (a,b,c,d) \text{ in even order,} \\ -1 \text{ for } (a,b,c,d) \text{ in odd order,} \\ 0 \text{ otherwise.} \end{cases}$
`LevC(dn,dn,dn,dn)`	Levi-Civita tensor, $\epsilon_{abcd} := (-g)^{1/2}[a,b,c,d]$
`Chr(dn,dn,dn)`	Christoffel symbol of the first kind,[17] $\Gamma_{bca} := \frac{1}{2}(g_{ab,c} + g_{ac,b} - g_{cb,a})$

[16]Three dimensional versions of the Levi-Civita symbols also exist as `LevCS(dn,dn,dn)`, etc.

[17]Note that this index ordering differs from that of Misner-Thorne-Wheeler [3]. We retain this ordering for consistency with earlier versions of GRTensor.

`Chr(dn,dn,up)`	Christoffel symbol of the second kind, $\Gamma^a_{bc} := g^{ad}\Gamma_{bcd}$
`R(dn,dn,dn,dn)`	Riemann tensor,
	$R^a{}_{bcd} := \frac{\partial \Gamma^a_{bd}}{\partial x^c} - \frac{\partial \Gamma^a_{bc}}{\partial x^d} + \Gamma^a_{ec}\Gamma^e_{bd} - \Gamma^a_{ed}\Gamma^e_{bc}$
`R(dn,dn)`	Ricci tensor, $R_{ab} := R^c{}_{acb}$
`Ricciscalar`	Ricci scalar, $R := R^a{}_a$
`S(dn,dn)`	trace-free Ricci tensor, $S_{ab} := R_{ab} - \frac{1}{n} R g_{ab}$
`C(dn,dn,dn,dn)`	Weyl tensor,
	$C_{abcd} := R_{abcd} - \frac{2}{n-2}(g_{a[c}S_{d]b} - g_{b[c}S_{d]a}) - \frac{2}{n(n-1)} R g_{a[c}g_{d]b}$
`Cstar(dn,dn,dn,dn)`	$C^*_{abcd} := \frac{1}{2}\epsilon_{abef}C^{ef}{}_{cd}$
`G(dn,dn)`	Einstein tensor, $G_{ab} := R_{ab} - \frac{1}{2} R g_{ab}$
`C2(up,up,dn,dn)`	$C2^{ab}{}_{cd} := C^{abef}C_{efcd}$
`S2(up,dn)`	$S2^a{}_b := S^a{}_c S^c{}_b$
`S3(up,dn)`	$S3^a{}_b := S2^a{}_c S^c{}_b$
`S4(up,dn)`	$S4^a{}_b := S3^a{}_c S^c{}_b$
`CS(dn,dn)`	$CS_{ab} := C_{acdb}S^{cd}$
`CSstar(dn,dn)`	$CS^*_{ab} := C^*_{acbd}S^{cd}$

7.2 Scalar polynomial invariants

GRTensorII includes the definitions of eighteen scalar polynomial invariants of the Riemann tensor. This includes the set of Carminati and McLenaghan [4], which constitutes sixteen invariants and has been proven to be complete for perfect fluid as well as Maxwell-type Ricci tensors. It consists of the invariants

$$CM := \{R, r_1, r_2, r_3, w_1, w_2, m_1, m_2, m_3, m_4, m_5\}$$

defined in the table below.

As indicated by Zakhary and McIntosh [5], more general Ricci types may require at least an extra invariant (here labeled m_6) to ensure completeness. They suggest that if the CM set is augmented by this additional invariant, the resulting set is complete for all Ricci and Petrov types. It is certainly true that none of the invariants in the resulting set can be constructed from invariant polynomials of equal or lower order. It is also true that any invariant of less than sixth order which is not within this set can be constructed identically from members of this set [6].

GRTensor name	Range		Definition[18]
`Ricciscalar`	Real	R	$:= R^a{}_a$
`R1`	Real	r_1	$:= \Phi_{AB\dot{A}\dot{B}}\Phi^{AB\dot{A}\dot{B}}$
			$= \frac{1}{4}S^a{}_b S^b{}_a$
`R2`	Real	r_2	$:= \Phi^A{}_B{}^{\dot{A}}{}_{\dot{B}}\Phi^B{}_C{}^{\dot{B}}{}_{\dot{C}}\Phi^C{}_A{}^{\dot{C}}{}_{\dot{A}}$
			$= -\frac{1}{8}S^a{}_b S^b{}_c S^c{}_a$
`R3`	Real	r_3	$:= \Phi^A{}_B{}^{\dot{A}}{}_{\dot{B}}\Phi^B{}_C{}^{\dot{B}}{}_{\dot{C}}\Phi^C{}_D{}^{\dot{C}}{}_{\dot{D}}\Phi^D{}_A{}^{\dot{D}}{}_{\dot{A}}$
			$= \frac{1}{16}S^a{}_b S^b{}_c S^c{}_d S^d{}_a$
`W1`	Complex	w_1	$:= \Psi_{ABCD}\Psi^{ABCD}$
			$= \frac{1}{8}(C_{abcd} + iC^*_{abcd})C^{abcd}$
`W2`	Complex	w_2	$:= \Psi^{AB}{}_{CD}\Psi^{CD}{}_{EF}\Psi^{EF}{}_{AB}$
			$= -\frac{1}{16}(C_{ab}{}^{cd} + iC^*_{ab}{}^{cd})C_{cd}{}^{ef}C_{ef}{}^{ab}$
`M1`	Complex	m_1	$:= \Psi_{ABCD}\Phi^{AB\dot{A}\dot{B}}\Phi^{CD}{}_{\dot{A}\dot{B}}$
			$= \frac{1}{8}S^{ab}S^{cd}(C_{acdb} + iC^*_{acdb})$
`M2a`	Real	m_{2a}	$:= \frac{1}{16}S^{bc}S_{ef}C_{abcd}C^{aefd}$
`M2b`	Real	m_{2b}	$:= \frac{1}{16}S^{bc}S_{ef}C^*_{abcd}C^{*aefd}$
`M2`	Complex	m_2	$:= \Psi_{ABCD}\Psi^{AB}{}_{EF}\Phi^{CD\dot{A}\dot{B}}\Phi^{EF}{}_{\dot{A}\dot{B}}$
			$= (m_{2a} - m_{2b}) + \frac{1}{8}iS^{bc}S_{ef}C^*_{abcd}C^{aefd}$
`M3`	Real	m_3	$:= \Psi_{ABCD}\bar{\Psi}_{\dot{A}\dot{B}\dot{C}\dot{D}}\Phi^{AB\dot{A}\dot{B}}\Phi^{CD\dot{C}\dot{D}}$
			$= m_{2a} + m_{2b}$
`M4a`	Real	m_{4a}	$:= -\frac{1}{32}S^{ag}S^{ef}S^c{}_d C_{ac}{}^{db}C_{befg}$
`M4b`	Real	m_{4b}	$:= -\frac{1}{32}S^{ag}S^{ef}S^c{}_d C^*_{ac}{}^{db}C^*_{befg}$
`M4`	Real	m_4	$:= \Psi_{ABCD}\bar{\Psi}_{\dot{A}\dot{B}\dot{C}\dot{D}}\Phi^{AB\dot{C}\dot{E}}\Phi^{CE\dot{A}\dot{B}}\Phi^D{}_E{}^{\dot{D}}{}_{\dot{E}}$
			$= m_{4a} + m_{4b}$
`M5a`	Real	m_{5a}	$:= \frac{1}{32}S^{cd}S^{ef}C^{aghb}C_{acdb}C_{gefh}$
`M5b`	Real	m_{5b}	$:= \frac{1}{32}S^{cd}S^{ef}C^{aghb}C^*_{acdb}C^*_{gefh}$
`M5c`	Real	m_{5c}	$:= \frac{1}{32}S^{cd}S^{ef}C^{*aghb}C_{acdb}C_{gefh}$
`M5d`	Real	m_{5d}	$:= \frac{1}{32}S^{cd}S^{ef}C^{*aghb}C^*_{acdb}C^*_{gefh}$
`M5`	Complex	m_5	$:= \Psi_{ABCD}\Psi^{CDEF}\bar{\Psi}^{\dot{A}\dot{B}\dot{E}\dot{F}}\Phi^{AB}{}_{\dot{A}\dot{B}}\Phi_{EF\dot{E}\dot{F}}$

[18]Definitions are given in terms of Ricci and Weyl spinors, $\Phi_{AB\dot{A}\dot{B}}, \Psi_{ABCD}$ and the corresponding expression in terms of the trace free tensors, S_{ab}, C_{abcd}.

			$= (m_{5a} + m_{5b}) + i(m_{5c} + m_{5d})$
M6	Complex	m_6	$:= \Psi_{ABCD}\Phi^{AB\dot{A}\dot{B}}\Phi^{CD\dot{C}\dot{D}}\Phi^{AB}{}_{\dot{A}\dot{B}}\Phi_{EF\dot{E}\dot{F}}$
			$= \frac{1}{32}S_a{}^e S_e{}^c S_b{}^f S_f{}^d (C^{ab}{}_{cd} + iC^{*ab}{}_{cd})$
W1R, W2R, etc.	Real		Re(w_1), Re(w_2), Re(m_1), etc.
W1I, W2I, etc.	Real		Im(w_1), Im(w_1), Im(m_2), etc.
invars	—		{Ricciscalar, R1, R2, R3, W1, W2, M1, M2, M3, M4, M5, M6}
Rinvars	—		{ Ricciscalar, R1, R2, R3 }
Winvars	—		{ W1, W2 }
Minvars	—		{ M1, M2, M3, M4, M5, M6 }
CMinvars	—		{Ricciscalar, R1, R2, R3, W1, W2, M1, M2, M3, M4, M5}
SSinvars[19]	—		{Ricciscalar, R1, R2, R3, W1R, M1R, M2a, M5a }

7.3 Operators

As discussed in Section 1.2, operators in GRTensorII are objects which require the specification of some additional information in order to be calculated. For instance, they can be tensors whose definition involves other user-specifiable tensors, or derivative operators which act on a specifiable tensor.

The standard library of GRTensorII defines a number of operators which can be divided into two classes: vector operators, which calculate tensors associated with vector fields, and derivative operators, which differentiate their argument. These are listed in the following tables.

For concreteness, the tables use the names

`f(x)` to represent scalars,

`v(up)` to represent single-index objects, and

`T(up,dn)` to represent tensors with an arbitrary number of indices.

These names do not have to be entered literally into the arguments of the operator, but rather any tensor of a similar type may be substituted.

[19]In spherical symmetry (and its specializations to plane symmetry, etc.), this subset of the invariants constitutes a complete set [7].

GRTensorII name	Definition
`LieD[v,T(up,dn)]`	Lie derivative, $L_v T^a{}_b := v^c \nabla_c T^a{}_b - T^c{}_b \nabla_c v^a + T^a{}_c \nabla_b v^a$
`Box[T(up,dn)]`	d'Alembertian, $\Box T^a{}_b := \nabla^c \nabla_c T^a{}_b$
`Dsq[f(x)]`	$\partial_a f(x) \partial^a f(x)$
`CDsq[f(x)]`	$\nabla_a f(x) \nabla^a f(x)$

GRTensorII name	Definition [8]
`vnorm[v]`	vector norm, $v_a v^a$
`h[v](dn,dn)`	projection tensor, $h_{ab} := g_{ab} - (v_a v^a) v_a v_b$
`acc[v](up)`	acceleration, $a^a := v^a \nabla_a v^b$
`expsc[v]`	expansion scalar, $\Theta := \nabla_a v^a$
`shear[v](dn,dn)`	shear tensor, $\sigma_{ab} := \nabla_{(a} v_{b)} + a_{(a} v_{b)} - \frac{1}{3}\Theta h_{ab}$
`shear[v]`	shear scalar, $\sigma := (\sigma^a{}_b \sigma^b{}_a)^{1/2}$
`vor[v](dn,dn)`	vorticity tensor, $\omega_{ab} := a_{[a} v_{b]} - \nabla_{[a} v_{b]}$
`vor[v](up)`	vorticity vector, $\omega^a := \frac{1}{2}\epsilon^{abcd} v_b \omega_{cd}$
`vor[v]`	vorticity scalar, $\omega := (\omega^a{}_b \omega^b{}_a)^{1/2}$
`RayEqn[v]`	Raychaudhuri's equation, $-R_{ab} v^a v^b = v^a \nabla_a \Theta - \nabla_a a^a + \Theta^2 + (\sigma^2 - \omega^2)$
`E[v](dn,dn)`	electric part of the Weyl tensor, $E_{ab} := C_{acbd} v^c v^d$
`H[v](dn,dn)`	magnetic part of the Weyl tensor, $H_{ab} := C^*_{acbd} v^c v^d$
`KillingTest[v]`	determines if v^a is a Killing vector (see also `?killing`).

References

[1] GRTensorII software and documentation can be obtained free of charge from the ftp site at `astro.queensu.ca` in the `/pub/grtensor` directory, or from the world-wide-web page `http://astro.queensu.ca/~grtensor/`.

[2] Denis Pollney, Peter Musgrave, Kevin Santosuosso, and Kayll Lake. Algorithms for computer algebra calculations in spacetime: I The calculation of curvature. *Class. Quantum Grav.*, 13:2289–2309, 1996. This paper is also archived at the GRTensorII world-wide-web pages [1].

[3] Charles W. Misner, Kip S. Thorne, and John Archibald Wheeler. *Gravitation*. W. H. Freeman and Co., New York, 1973.

[4] J. Carminati and R. G. McLenaghan. Algebraic invariants of the Riemann tensor in a four-dimensional Lorentzian space. *J. Math. Phys.*, 32:3135–3140, 1991.

[5] E. Zakhary and C. B. G McIntosh. A complete set of Riemann invariants. preprint, 1996.

[6] Denis Pollney. Notes on completeness of sets scalar polynomial invariants of the Riemann tensor to order five. Unpublished (available upon request), 1996.

[7] Peter Musgrave. Scalar polynomial invariants of the Riemann tensor for spherical symmetry. Unpublished (available upon request), 1996.

[8] Jürgen Ehlers. Contributions to the relativistic mechanics of continuous media. *Gen. Rel. Grav.*, 25:1–43, 1993.

Commands described in this booklet:

`grcalc` (*objectSeq*) C6

`grcalcd` (*objectSeq*) C7

`grcalc1` (*object*, *indexList*) C8

`grclear` (*objectSeq*) C9

`grdisplay` (*objectSeq*) C9

`autoAlias` (*function*)) C10

`diffAlias` (*varList*, *functionList*) C11

`grDalias` (*function*, *var1*, *str1*, *var2*, *str2*, ...) C11

`gralter` (*objectSeq*, [*function1*], [*function2*], ...) C12

`grmap` (*objectSeq*, *function*, *fnArg1*, ..., 'x', ..., *fnArgN*) C14

`grcomponent` (*object*, *component*) C17

`grarray` (*object*) C17

The information contained in this booklet is also available from the following online help pages:

`?grt_objects`, `?grt_operators`, `grt_invars`, `?groptions`, `?grcalc`, `?grcalc1`, `?grclear`, `?grdisplay`, `?diffAlias`, `?autoAlias`, `?grDalias`, `?gralter` `?grmap`, `?grcomponent`, `?grarray`, `?killing`, `?grt_basis`.