GRTENSORII

GRTensorII Release 1.50
For MapleV Releases 3 and 4

D. Defining new tensors

Peter Musgrave
Denis Pollney
Kayll Lake

July 1996

Contents

Queen's University at Kingston, Ontario

The GRTensorII standard library contains the definitions of a number of tensors, scalars, and other indexed objects whose components can be calculated. The `grdef()` command is included to facilitate the specification of new tensors in a simple and natural manner. It allows tensors to be defined either as an equation in terms of previously defined tensors, or by manual entry of the component values. Inner and outer products of tensors, symmetrizations, and derivatives can all be specified as part of the tensor definitions. Furthermore, index symmetries of the newly defined tensor can be included. A calculation algorithm for the new tensor is created automatically based on the definition and will automatically take index symmetries into account in order to provide an efficient calculation.

This booklet outlines the usage and various options available with the `grdef()` command.[1]

1 Command syntax

There are two ways to define a new tensor using `grdef()`. The first method is to simply state the name of the tensor, including its 'index structure' (ie. a list of covariant and contravariant indices). The second method provides a complete definition in terms of previously defined tensors.

Compare the following two invocations of `grdef()`:[2]

```
> grdef ( `A{a b}` ):

> grdef ( `G2{a b} := R{a b} - (1/2)*Ricciscalar*g{a b} + lambda*g{a b}` ):
```

The first form of the command defines a tensor whose component values are arbitrary and can be manually specified for the background spacetime. Specifically, it creates a definition of a covariant two-index object, `A`. The indices are listed in curly braces, {}, and assigned the labels `a` and `b`. This tensor would be accessed as `A(dn,dn)` in commands such as `grcalc()` and `grdisplay()`. Note, however, that although `grdef()` creates a definition of the tensor, it does not calculate its components automatically. The command

```
> grcalc ( A(dn,dn) ):
```

must be given in order to explicitly request the calculation of the components for in current background spacetime.[3] In this case, since no prescription for calculating the components of this tensor has been given, when the `grcalc()` command is issued the user is prompted to input the components of `A(dn,dn)` manually.

The second version of `grdef()` listed above is somewhat more complicated. In this case the command once again defines a contravariant two-index tensor, `G2`, but this time it is explicitly assigned to an expression involving a number of previously defined tensors. The `grdef()` command uses this information to automatically create a calculation function for the object `G2(dn,dn)`. Thus, the command

[1] The `grdef()` command replaces the `grdefine()` command which was the GRTensorII standard for definitions of new tensors from Versions 1.00 to 1.21. The `grdefine()` command is still included for purposes of backwards compatibility. See `?grdefine` for details of its use.

[2] The specifics of the notation used by these two commands are described in the sections to follow.

[3] See Booklet *C: Calculating tensor components.*

```
> grcalc ( G2(dn,dn) ):
```

can be used to calculate the newly defined tensor based on its given definition involving `R(dn,dn)`, `Ricciscalar` and `g(dn,dn)`.

The argument to `grdef()` which describes the definition of a new tensor is a MapleV string (that is, enclosed in back-quotes, '‘'). The string specifies a tensor equation which includes information about the index structure, symmetries, and summations, which are to be carried out in the calculation of the tensor. This information is parsed by the `grdef()` command in order to create a calculation function corresponding to the definition. The `grcalc()` command uses this calculation function in order to arrive at the components of the newly defined object in a given background geometry. The next section of this booklet details the format of the tensor definition string.

The syntax for the `grdef()` command is as follows:

`grdef (` *defString* `,` [*symSet*] `,` [*asymSet*] `,` [*rsumSet*] `)`

defString – The definition string (described in Section 2).

symSet – (*optional*) A set of lists of indices which are symmetric under interchange (described in Section 3).

asymSet – (*optional*) A set of lists of indices which are anti-symmetric under interchange (described in Section 3).

rsumSet – (*optional*) A set of ranges over which dummy indices are to by summed (described in Section 4).

Example: `> grdef ( ‘G2{(a b)} := G{a b} + lambda*g{a b}‘ ):`

The next sections give details of the use of each of the arguments listed above. However, the first-time reader may find it instructive to skip ahead to the Examples section (Section 5 of this booklet) to get some familiarity with the syntax of the `grdef()` command before reading the more in-depth descriptions that now follow.

2 Tensors and tensor expressions

The main job of the `grdef()` command is to parse a string containing a tensor definition. As an example, consider the argument of the second example given above,

```
‘G2{a b} := R{a b} - (1/2)*Ricciscalar*g{a b} + lambda*g{a b}‘
```

This is a special case of the general form used for tensor assignments,

‘*newObject* `:=` *objectDef*‘.

The string is interpreted by `grdef()` as an assignment of an expression involving indexed objects, *objectDef*, to a new indexed object, *newObject*. The format of this defining expression is outlined in the following paragraphs.[4]

Indexed objects: Indexed objects (tensors and the like) are specified by their name and index structure:

Name { *indices* }

The *Name* is simply an unassigned MapleV name by which the object is referenced, such as `R` in the case of the Riemann tensor and its contractions, and `Chr` in the case of the Christoffel symbols.[5]

The index list, *indices*, is always enclosed in curly braces, {}. Individual indices take the form of alphabetic characters separated by spaces. Different classes of index are specifiable:

`a`, `b`, `c`, ...	– covariant indices,
`^a`, `^b`, `^c`, ...	– contravariant indices,
`(a)`, `(b)`, `(c)`, ...	– covariant basis indices,
`^(a)`, `^(b)`, `^(c)`, ...	– contravariant basis indices.

Thus, within the `grdef()` command, the Riemann tensor $R^a{}_{bcd}$ is referenced using:

```
R{^a b c d}
```

while the basis components of the same tensor are

```
R{^(a) (b) (c) (d)}
```

The characters used in the index labels (`a`, `b`, ...) should be unassigned MapleV names. For instance, in the above example, none of `a`, `b`, `c`, `d`, can have been assigned a value. This notation differs from that used by such commands as `grcalc()` which uses the generic labels '`up`', '`dn`', etc., to indicate index positions. The `grdef()` notation attaches a unique label to each index. This additional information is required to be able to specify index summations and correspondences, as described below. (Scalars are referenced using simply their name without an index list or curly braces.)

Derivatives of tensors: The special characters '`,`' (comma) and '`;`' (semi-colon) can be used to specify partial derivatives and covariant derivatives in the index lists of tensors. Thus the tensor $R_{ab;c}$ $(= \nabla_c R_{ab})$ would be represented as:

```
R{a b ;c}
```

The specific action of the derivative operation is controlled by the form of the index (tensor or basis) which immediately follows the derivative indicator, as in the following examples:

`R{^a b ,c}`:	$R^a{}_{b,c} := \frac{\partial R_{ab}}{\partial x^c}$
`R{^a b ;c}`:	$R^a{}_{b;c} := R^a{}_{b,c} - \Gamma^d_{bc} R^a{}_d + \Gamma^a_{bd} R^d{}_c$
`R{^(a) (b) ,(c)}`:	$R^{(a)}{}_{(b),(c)} := \frac{\partial R^{(a)}{}_{(b)}}{\partial x^d} e_{(c)}{}^d$
`R{^(a) (b) ;(c)}`:	$R^{(a)}{}_{(b);(c)} := R^{(a)}{}_{(b),(c)} - \gamma^{(d)}{}_{(b)(c)} R^{(a)}{}_{(d)} + \gamma^{(a)}{}_{(b)(d)} R^{(d)}{}_{(c)}$

[4] As noted in the previous section, the *objectDef* expression does not need to be specified. In this case, the tensor is created without assigning it a definition, and the components must be manually entered using the command `grcalc()`.

[5] See Booklet *C: Calculating tensor components* for a list of objects in the standard GRTensorII library.

These correspond to the actions of the index labels `pdn`, `cdn`, `pbdn`, and `cbdn` in commands such as `grcalc()` (see Booklet *C: Calculating tensor components*).

Tensor sums and products: The exterior product of a pair of tensors is indicated by the '`*`' symbol. It is also possible to specify the product of a scalar function and a tensor using the same operator. Addition and subtraction of tensors is carried out as usual by the '`+`' and '`-`' operators. Round braces, `()`, can be used to group terms. Thus, the tensor expression

$$\frac{1}{2}\left(R_{abcd} + f(x) g_{ab} g_{cd}\right)$$

would be represented as:

```
(1/2) * ( R{a b c d} + f(x)*g{a b}*g{c d} )
```

The index names and configurations of each term must be consistent with each other and with the left-hand side of a tensor assignment.

Inner products (summations over indices): A summation over the range of an index can be specified by repeating the index name, once in the covariant and once in the contravariant position. Thus, the expressions

$$\sum_b R^{ab}{}_{bc} \qquad \text{and} \qquad \sum_b R_a{}^b R_{bc}$$

would be represented, respectively, as

```
R{^a ^b b c},        and        R{a ^b}*R{b c}.
```

By default the summation is carried out over all index values, however restricted summations (say over three dimensions out of four) can also be specified, as described in Section 4, below.

Operators: Operators can be used in `grdef()` just as they are used in calculations using `grcalc()`. The argument of the operator is placed in square braces. Thus, the command

```
> grdef ( `X := R{^a ^b}*Box[ R{a b} ]` ):
```

serves to define the scalar

$$X := R^{ab} \Box R_{ab},$$

A limitation on the use of operators, however, is that they can only be used on individual GRTensorII objects, not functions of objects. To define, for instance, the object

$$T_{ab} := \Box(R_{acdb} R^{cd}),$$

a two stage definition is needed. First an intermediate 2-tensor corresponding to the argument of the operator, in this case $R_{acdb}R^{cd}$, is defined:

```
> grdef ( `T1{a b} := R{a c d b}*R{^c ^d}` ):
```

The final object, T_{ab}, is then defined as an operator acting on $T1_{ab}$:

```
> grdef ( `T{a b} := Box[T1{a b}]` ):
```

Nested operators, to define objects such as

$$\Box\Box R_{abcd},$$

are handled in a similar fashion, defining intermediate objects which correspond to the action of an operator on a single object and building upon these.

New operators can not be defined using `grdef()`.

Symmetrizations: A short-hand notation can be used to indicate symmetrizations over a set of indices. The notation corresponds to the common usage of round and square braces to denote symmetrization and anti-symmetrization, respectively. Specifically, define

$$T_{abc\ldots(s_1\ldots s_m)\ldots de} := \frac{1}{m!}\sum_{\pi} T_{abc\ldots s_{\pi(1)}\ldots s_{\pi(m)}\ldots de},$$
$$T_{abc\ldots[s_1\ldots s_m]\ldots de} := \frac{1}{m!}\sum_{\pi} (-1)^{\mathrm{sign}(\pi)} T_{abc\ldots s_{\pi(1)}\ldots s_{\pi(m)}\ldots de},$$

where the summations are taken over all permutations, π, of the numbers $1\ldots m$, and $\mathrm{sign}(\pi)$ represents the sign (odd or even) of the permutation.

The tensor

$$T_{a(bcd)} := \frac{1}{6}\left(T_{abcd} + T_{acdb} + T_{adbc} + T_{abdc} + T_{adcb} + T_{bdc}\right),$$

could be represented in `grdef()` by the string[6,7]

```
T{a (b c d)},
```

while the tensor

$$T_{a[bcd]} := \frac{1}{6}\left(T_{abcd} + T_{acdb} + T_{adbc} - T_{abdc} - T_{adcb} - T_{bdc}\right),$$

could be represented by

```
T{ a [b c d]}.
```

Symmetrizations can also be carried over tensor products. Thus

$$R^{a}{}_{bc(d}R_{e)f}$$

could be written

```
R{^a b c (d}*R{e) f}
```

[6]A technical point arises from the fact that round braces are used to indicate both symmetrizations as well as basis indices. However, it is always possible for the parser to determine from the context which is intended. If the braces surround a single index, then the index is regarded as a basis index, otherwise a symmetrization is assumed.

[7]Spaces are not needed between indices where braces occur. The example given here could also be entered as `T{a(b c d)}`, omitting the space between the `a` and `b`.

Braces are determined to be 'covariant' or 'contravariant' based on the index which immediately follows an open-brace, `(`, `[`, and the index which precedes a close-brace, `)`, `]`. Braces enclose only indices of the same character. Thus the name

```
T{(a ^b c)}
```

indicates that the symmetrization is to be performed only over the covariant indices within the round braces, ie. `a` and `c`.

Finally, indices can be excluded from symmetrizations by placing them between vertical bars, '`|`'. Thus if a symmetrization were to be carried out over the second and fourth index of a four index tensor, this could be written as

$$T_{a(b|c|d)} := \frac{1}{2}\left(T_{abcd} + T_{adcb}\right),$$

which in `grdef()` notation is

```
T{a (b | c | d)}.
```

3 Index symmetries

Symmetries among tensor indices can be used in calculation programs to significantly reduce the time taken for a calculation by recognizing redundant components. The `grdef()` command can not determine on its own when such index symmetries exist. However, symmetries can be listed explicitly in the tensor definition so that the resulting calculation function is as efficient as possible.

The notation used to indicate symmetric and anti-symmetric index groups is identical to that used for symmetrizations, described in the previous section. Thus if braces are placed within the index list of the *new* tensor (on the left-hand side of the tensor assignment) these are interpreted as index symmetries rather than symmetrizations.

To give some explicit examples, round braces, `()`, indicate that a tensor is symmetric in the enclosed indices. The assignment

```
> grdef ( `A{(a b)} := R{a b}` ):
```

assigns a calculation function to the tensor A_{ab} which assumes that it is symmetric in its indices, ie.,

$$A_{ab} = A_{ba}.$$

The calculation algorithm called by the command

```
> grcalc ( A(dn,dn) ):
```

then calculates only the components in the upper diagonal and cross-assigns the components in the lower diagonal. By the same token, the command

```
> grdef ( `A{(a b)(c d)}` ):
```

assigns to the object `A(dn,dn,dn,dn)` a calculation algorithm which assumed the tensor is symmetric in its first two and last two indices,[8]

$$A_{abcd} = A_{bacd} = A_{abdc} = A_{badc}.$$

Since in this case A_{abcd} was not given a defining expression, a call to `grcalc()` would prompt the user to enter its components explicitly. However, it would only be necessary to input the independent components (up to the specified index symmetries).

Square braces act similarly, indicating indices in which the newly defined object is anti-symmetric. Thus

```
> grdef ( `A{[a b c] d}` ):
```

defines a tensor for which it is assumed that

$$A_{abcd} = A_{bcad} = A_{cabd} = -A_{acbd} = -A_{bacd} = -A_{cbad}.$$

The 'covariant' or 'contravariant' character of the braces is determined as for symmetrizations, described in the previous section. Also as above, indices which are to be excluded from a symmetrization can be enclosed in vertical bars, '`|`'.

Note that the symmetries need to be stated explicitly on the left-hand side of the assignment. Although in the definition of A_{ab} above the index symmetry is obvious from the properties of the Ricci tensor, the `grdef()` command does not search for such properties. It must be instructed as to which index symmetries to include in the calculation function for the tensor.

Another important consequence of this is that the symmetries which are stated explicitly will be incorporated into the calculation function regardless of whether the listed symmetry is an actual property of the tensor which is being defined. Thus the definition

```
> grdef ( `A{[a b]} := R{^c a c b}` ):
```

will assign a calculation function to A_{ab} which assumes that the tensor is anti-symmetric, which it clearly is not. If the `grcalc()` command is subsequently used to calculate A_{ab} for a particular metric, it will always arrive at the result

$$A_{ab} = 0,$$

which is incorrect in general.

Finally, the different role of braces on the left-hand side and right-hand side of a tensor assignment are emphasized: On the left-hand side, braces are informational, describing the symmetries of the newly defined object. On the right-hand side, the braces specify that the operation of symmetrization is to be carried out over the enclosed indices. Thus the command

```
> grdef ( `A{(a b)} := B{(a b)}` ):
```

[8] A calculation function satisfying the required symmetries is created automatically if such a function does not already exist among the existing GRTensorII object definitions.

defines the tensor

$$A_{ab} := \frac{1}{2}\left(B_{ab} + B_{ba}\right)$$

(as indicated by the ()-braces on the right-hand side), which is assumed to be symmetric in its indices (as indicated by the ()-braces on the left-hand side). If the braces on the left of the `grdef()` command are omitted, the calculation function assigned to the new object would not take into account the symmetry inherent in A_{ab}, and, for example, the components A_{12} and A_{21} would be calculated independently, even though by examining the right-hand side of the definition it is clear that they are equal.

Alternate specification of index symmetries

The notation used to specify tensor indices in `grdef()` has been chosen to mirror as closely as possible the standard notation of textbooks of classical tensor analysis. However, because the text typed onto an input line is linear (lacking visual super-scripts and sub-scripts) the presence of a large number of braces among covariant and contravariant indices can serve to obscure the contents of an index list. Not only can this make the worksheet difficult to read later on, it also increases the chance of an error being introduced into the tensor definition.

To alleviate this problem somewhat, an alternate method exists for specifying symmetries among the tensor indices. The `grdef()` command possesses optional extra arguments which can be used to enter symmetry and anti-symmetry lists. This can be used instead of the placing of braces among the tensor indices in the left-hand side of the definition string.

In these optional arguments, symmetries are expressed as lists of index numbers: The list `[2,3,5]` indicates that the second, third and fifth indices are involved in a symmetry (or anti-symmetry). To specify symmetries, include the argument

`sym={`*symLists*`}`

to the `grdef()` command. Thus,

```
> grdef ( `A{a b c d}`, sym={[1,2],[3,4]} ):
```

indicates that the new tensor A_{abcd} should be considered to be symmetric in its first two and last two indices:

$$A_{abcd} = A_{bacd} = A_{abdc} = A_{badc}.$$

(Note that this invocation of `grdef()` has an identical effect as one of the examples of the previous section.) Alternatively,

```
> grdef ( `A{a b c d}`, sym={[2,3,4]} ):
```

defines a tensor which is symmetric under interchange of its second third and fourth indices:

$$A_{abcd} = A_{acdb} = A_{adbc} = A_{abdc} = A_{acbd} = A_{adcb}.$$

Anti-symmetries are specified the same way using the argument

`asym={`*asymLists*`}`

The tensor defined by

```
> grdef ( `A{a b c}`, asym={[1,2]} ):
```

is assigned a calculation function which assumes it is anti-symmetric in its first two indices,

$$A_{abc} = -A_{bac},$$

while the command

```
> grdef ( `A{a b c d}`, sym={[1,3]}, asym={[2,4]} ):
```

assigns a calculation function which is symmetric in indices 1 and 3 and anti-symmetric in indices 2 and 4.

Finally, note that a limitation of this notation (and the use of braces as described in the previous section) is that some more complicated index symmetries such as those involving groups of indices, can not be specified. An important example is the index pair symmetry of the Riemann tensor,

$$R_{abcd} = R_{cdab}.$$

Although `grdef()` can not currently automatically create such symmetry functions, the Riemann symmetry function has been pre-programmed, and can be assigned to newly defined objects using the optional argument `symfn`. This argument has the form

`symfn =` *objectName*

where *objectName* is the name of a pre-defined object with the same symmetries as the object to be newly defined. Thus a new object with Riemann index symmetries can be created using the definition

```
> grdef ( `T{a b c d}`, symfn=R(dn,dn,dn,dn) ):
```

Symmetry functions assigned in this way overide any other index symmetry specifications (using braces or the `sym=` or `asym=` parameters) for the newly defined object.

4 Special cases

Defining vectors

Single index objects form a special case within `grdef()`. They can be defined as usual by the means given in the previous section. Alternatively, however, they can also be assigned a value directly by specifying the components as a list. For example, the command

```
> grdef ( `v^a := [0,0,0,1]` ):
```

defines a single index object, `v`, whose four components are assigned the values `[0,0,0,1]`. This form of the `grdef()` command alleviates the need to run `grcalc()` after the definition in order to assign values to the components. However, if the default spacetime is changed (either by creating or loading a new background[9]), then `grcalc()` would still be required in order to assign the components of the vector in the new geometry.

The Kronecker delta

A special object, `kdelta{^a b}`, allows single components of tensors to be isolated in tensor definitions. For example, if the coordinates of the background spacetime are $(r, theta, phi, t)$, the t-component can be selected using the object

`kdelta{^a $t}`.

The '$' character preceding the coordinate name t indicates that that the `kdelta` object is to take the form

$$\delta^a{}_t := \begin{cases} 1, & \text{if} \quad a = t, \\ 0, & \text{otherwise.} \end{cases}$$

Thus a vector in the t-direction could be defined using the command

```
> grdef ( `v{^a} := f(t)*kdelta{^a $t}` ):
```

(which is equivalent to the command

```
> grdef ( `v{^a} := [0,0,0,f(t)]` ):
```

in the given coordinates). Another example is a 2-index tensor,

$$T_{ab} := P(r,t)g_{ab} + (P(r,t) + \rho(r,t))\delta^t{}_a\delta^t{}_b,$$

which could be defined using the command

```
> grdef ( `T{(a b)} :=
         P(r,t)*g{a b} + ( P(r,t) + rho(r,t) )*kdelta{a $t}*kdelta{b $t}` ):
```

Multiple metrics

The `grdef()` command allows the definition of objects which depend on multiple background geometries. Such objects arise, for instance, when one considers the junction between two spacetimes described by different metrics. In a tensor definition, objects which are to use alternate background geometries are indexed using angle braces, `<>`. For example, the following definition can be used to define a tensor, `Dg`, which is the difference of two metrics:

```
> grdef ( `Dg{a b} := g<1>{a b} - g<2>{a b}` ):
```

In this definition, the indices `<1>` and `<2>` in the objects `g{a b}` indicate that the components of `g(dn,dn)` should be taken from metrics specified by the user at calculation time. To calculate the object `Dg`, the command

[9] See Booklet *B: Specifying spacetimes.*

```
> grcalc ( 1=ssym, 2=schw, Dg(dn,dn) ):
```

would be used, where in this example `ssym` and `schw` are the names of spacetimes that have been previously loaded in the session. In the calculation of `Dg`, the objects in its definition that are labeled with the index `<1>` would be taken from the `ssym` spacetime, while objects labeled by `<2>` would use components from the `schw` spacetime.

Restricted summations

A further optional argument to the `grdef()` command allows summations over indices to be restricted to a certain range of index values. Consider the definition of the scalar

$$\phi := T_{ab}u^a u^b,$$

where, for some reason, we wish to restrict the implied summation over the dummy indices a and b to the range $2 \ldots 4$. The `grdef()` command creating such a definition is:

```
> grdef ( `phi := T{a b}*u{^a}*u{^b}`, restrict={a=2..4, b=2..4} ):
```

The `restrict={}` argument is a set specifying the ranges over which summations over dummy indices are to be performed. Any indices not listed in the set are assumed to be summed over their entire range.

5 Examples

In this section, a number of `grdef()` commands are listed, summarizing the use of the command. A number of the following examples are mirrored from the previous sections where they are described in more detail. The `grdef()` command is also used in a number of the demonstration worksheets available from the GRTensorII world-wide-web site, `http://astro.queensu.ca/~grtensor/`.

1. A generic contravariant single indexed object (vector):

   ```
   > grdef ( `u{^a}` ):
   ```

2. A generic covariant single indexed object (1-form):

   ```
   > grdef ( `v{a}` ):
   ```

3. A generic 2-index tensor, anti-symmetric in its indices:

   ```
   > grdef ( `A{[a b]}` ):
   ```

4. The covariant derivative of the Weyl tensor, $C_{abcd;e}$:

   ```
   > grdef ( `dC{[a b] [c d] e} := C{a b c d ;e}` ):
   ```

5. The component of the Ricci tensor measured along a particular vector, $\phi := R_{ab}u^a u^b$:

```
> grdef ( `phi := R{a b}*u{^a}*u{^b}` ):
```

6. The Einstein tensor with cosmological constant:

```
> grdef ( `G2{(a b)} := G{a b} + lambda*g{a b}` ):
```

(Note the use of braces on the left-hand side to indicate the index symmetry. Although the tensors g_{ab} and G_{ab} have a clear index symmetry, the `grdef()` command will only apply symmetries explicitly stated on the left-hand side of the definition.)

7. Basis components of the Einstein tensor with cosmological constant:

```
> grdef ( `G2{((a) (b))} := G{(a) (b)} + lambda*eta{(a) (b)}` ):
```

(This command is redundant if the previous definition of `G2(dn,dn)` is already created, since the basis components could be calculated automatically using `grcalc ( G2(bdn,bdn) )`. See also the note at the end of this section.)

8. The Bel-Robinson tensor, $T_{abcd} := C_{aecf}C_b{}^e{}_d{}^f - \frac{3}{2}g_{a[b}C_{jk]cf}C^{jk}{}_d{}^f$.

```
> grdef ( `T{(abcd)} := C{a e c f}*C{b ^e d ^f} - (3/2)*g{a [b}*C{j k] c
f}*C{^j ^k d ^f}` ):
```

(An alternate definition of the Bel-Robinson tensor is an example in Booklet *A: Introduction and overview*.)

9. A timelike vector field, $v^a := [0, 0, 0, f(r, t)]$:

```
> grdef ( `v{^a} := [0,0,0,f(r,t)]` ):
```

... and a corresponding perfect fluid energy-momentum tensor:

```
> grdef ( `T{(a b)} := P(r,t)*g{a b} + (rho(r,t) + P(r,t))*v{a}*v{b}` ):
```

10. An object with Riemann symmetries:

```
> grdef ( `T{a b c d} := (1/6)*g{a[c}*g{d]b}*Ricciscalar`,
symfn=R(dn,dn,dn,dn) ):
```

Object names can not be used twice. If an object with the same name already exists (either in the standard library or as a result of a previous use of `grdef()`), its definition must be removed before it can be re-used. The `grundefine()` command performs this task:

`grundefine ( objectName )`

objectSeq – A sequence objects whose definitions are to be removed.

Example: `> grundefine ( G(dn,dn) ):`

6 Saving and loading definitions

Definitions created with `grdef()` can be saved to a file using the `grsavedef()` command, allowing them to be used in other sessions and applications. In this way, collections of objects not in the standard GRTensorII library can be maintained.

`grsavedef ( objectSeq, fileName )`

objectSeq – A sequence of objects whose definitions are to be saved to a file.

fileName – The name of a file where the object definitions are to be saved.

Example: `> grsavedef ( G2(dn,dn), T(dn,dn,dn,dn), ‘newdefs.mpl‘ ):`

For instance, the Einstein and Bel-Robinson tensors defined in the previous section could be saved to the file ‘`newdefs.mpl`’ using the command:

```
> grsavedef ( G2(dn,dn), T(dn,dn,dn,dn), ‘newdefs.mpl‘ ):
```

If the newly created file is to reside in a directory other than the one in which the MapleV session was started, then the full path name must be used. Note also that `grsavedef()` makes use of the MapleV `write()` command, which does not check for the existence of the file which is being written. Thus if a file of the same name already exists in the specified directory *it will be overwritten*.

Files created by the `grsavedef()` command can be read into a new GRTensorII session using the command `grloaddef()`:

`grloaddef ( fileName )`

fileName – The name of the file (containing definitions created by `grsavedef()`) to be loaded.

Example: `> grloaddef ( ‘newdefs.mpl‘ ):`

Commands described in this booklet:

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

`?grdef`, `?grundefine`, `?grsavedef`, `?grloaddef`, `?kdelta`, `?grt_objects`, `?grt_basis`, `?grt_operators`, `?grcalc`, `?grdefine`.