GRTensorII

GRTensorII Release 1.50
For MapleV Releases 3 and 4

B. Specifying spacetimes

Peter Musgrave
Denis Pollney
Kayll Lake

July 1996

Contents

Queen's University at Kingston, Ontario

The simplest way to specify a spacetime geometry in GRTensorII is to use the `makeg()` facility. This function aids input by prompting the user for the information required to specify a coordinate metric (a $n \times n$ dimensional 2-tensor) or basis (a set of n linearly independent vectors related by a user-defined inner product).[1]

In addition to using `makeg()`, new spacetimes can be constructed from previously defined spacetimes through the use of the commands:

`grtransform()`	– performs a coordinate transformation of a metric,
`nptetrad()`	– constructs a null tetrad corresponding to a metric,
`nprotate()`	– performs rotations of a null tetrad, and
`grnewmetric()`	– creates a metric from a 2×2 MapleV array.

These commands are described in Sections 4–7 of this booklet.

The metrics created for use in GRTensorII can be saved to an ASCII file directly using `makeg()` or the `grsaveg()` command. These files can be loaded into GRTensorII using either the `qload()` or `grload()` commands. A directory of commonly used metric/basis files is available from the GRTensor world-wide-web pages [1].

1 Metrics and bases

The `makeg()` can be used to enter all of the information needed to specify a spacetime, either as a metric or set of basis vectors. This section establishes some notation that will be used throughout these booklets.

Metrics are n dimensional 2-tensors which are assumed to be symmetric:

$$g_{ab} := \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{12} & g_{22} & & \vdots \\ \vdots & & \ddots & \\ g_{1n} & \cdots & & g_{nn} \end{bmatrix}$$

The components g_{ij} are functions of the n coordinates $\{x_1, \ldots, x_n\}$.

Bases are sets of n independent vectors,

$$\left\{ e_{(1)}{}^a = [e_{11}, \ldots, e_{1n}], \quad \ldots \quad e_{(n)}{}^a = [e_{n1}, \ldots, e_{nn}], \right\}$$

whose inner product is defined by the symmetric matrix

$$\eta^{(a)(b)} = \begin{bmatrix} \eta_{11} & \eta_{12} & \cdots & \eta_{1n} \\ \eta_{12} & \eta_{22} & & \vdots \\ \vdots & & \ddots & \\ \eta_{1n} & \cdots & & \eta_{nn} \end{bmatrix},$$

via

$$\langle e_{(i)}{}^a, e_{(j)}{}^b \rangle = \eta^{(i)(j)} e_{(i)}{}^a e_{(j)}{}^b = g^{ab}.$$

(Note that here and throughout GRTensorII we use the convention that basis vectors are labeled with bracketed indices $\{(a), (b), \ldots\}$.)

[1] Although GRTensorII allows specification of spacetimes as both metrics and bases, in these booklets we will often refer to both formats as simply *metrics* with the understanding that we are referring also to bases created via `makeg()`.

 GRTensorII development has been supported by the Natural Science and Engineering Research Council of Canada and the Advisory Research Committee of Queen's University.

2 Using `makeg()`

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

`makeg (` *metricName*`,` `[`*metricPath*`] )`

metricName – the name to be given to the newly created metric or basis. This name will also be used to create the filename (with a `.mpl` extension) in which the metric information will be saved.

metricPath – *(optional)* this argument can be used to specify a directory name in which the metric file is to be placed. The argument is a string with directory levels separated by forward slashes, '`/`'. If this argument is not specified the directory stored in the global option variable `grOptionMetricPath` is used instead.

Example: `> makeg ( bondi ):`

Upon issuing the `makeg()` command, the user is prompted to enter all of the information necessary to specify the spacetime. The first of these prompts asks for the format in which the data is to be entered:[2]

```
Do you wish to enter a 1) metric [g(dn,dn)],
                       2) line element [ds],
                       3) non-holonomic basis [e1...e4], or
                       4) NP tetrad [l,n,m,mbar]
```

The reply should be an integer from 1 to 4. Each option is described in turn below.

2.1 The metric as a covariant 2-tensor

The user is prompted to enter the following information:

Coordinates These are entered as a MapleV list, eg. `[r,theta,phi,t]`. The names which are used as coordinates must be previously unassigned. Any number of coordinates may be entered, and their number will determine the dimension of the spacetime.

Signature This is an integer, $s = n_+ - n_-$, where n_+ is the number of positive components on the diagonal of the metric, and n_- is the number of negative components on the diagonal of the metric in a locally orthonormal basis. If the global variable `grOptionLLSC` is set to `true`, then this prompt does not appear.[3]

Form of the metric (diagonal or symmetric) This will reduce the number of coordinates which the user is required to input. If the 'diagonal' option is chosen then off-diagonal components are automatically set to zero. Components on the lower diagonal are automatically set equivalent to those on the upper diagonal. GRTensorII does not currently handle non-symmetric metrics.

[2] Note that the first option refers to '`g(dn,dn)`'. This is the standard GRTensorII representation of the covariant metric. See Booklet *C: Calculating tensor components* for more details regarding this notation.

[3] See the discription of the `grOptionLLSC` in Booklet *F: Installation and setup* and the `?groptions` online help page.

Metric components A prompt appears for each unknown component of the covariant metric. Keep in mind that if a component involves functions of the coordinates then the coordinate dependence must be stated explicitly, as in `M(r,t)` for example.

2.2 The metric as a line-element

It is often most convenient to enter a metric in the form that it is most commonly presented in journals and texts, ie. as a line element. This helps to minimize the risk of transcription errors between paper and the computer.

Choosing the second option from the `makeg()` menu, the user is first prompted to enter the coordinate names. As above, this should take the form of a MapleV list of unassigned names, eg. `[r,theta,phi,t]`.

The line element is then entered using the notation `d[x]` to represent the coordinate differential dx. For example, the line element

$$dx^2 + (dy + dz)^2$$

would be entered as

d[x]^2 + (d[y] + d[z])^2:

Naturally, the line element must be a quadratic form in the coordinate differentials.

Once the line element is entered, it is converted to a $n \times n$ covariant 2-tensor (`g(dn,dn)` in GRTensorII's notation) which is displayed so that they can be checked for errors in the input.

2.3 Non-holonomic bases

Choosing Options 3 or 4 from the `makeg()` menu allows one to enter the components of a non-holonomic basis. The first of these options (option `3: non-holonomic basis [e1...e4]`) allows the user to also specify the inner product, $\eta^{(a)(b)}$, between individual basis vectors. The second (option `4: NP tetrad [l,n,m,mbar]`) assumes the inner product of a Newman-Penrose null tetrad,

$$\eta^{(a)(b)} := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}. \tag{1}$$

In both cases, the user is first prompted to input the coordinate names as a MapleV list, eg. `[r,theta,phi,t]`.

The next prompt asks if covariant or contravariant components of the tetrad are to be entered.[4] The user is then asked to enter the basis vectors. These should be specified in the form of a MapleV list, eg. `[1,0,0,0]`. The number of vectors that are entered is determined by the number of coordinates making up the spacetime. For general bases, the vectors are labeled by

[4]The option to enter both forms of tetrad also exists, since in certain cases the inversion of the tetrad introduces complicated terms (especially radicals) which MapleV has difficulty simplifying. In such cases it is sometimes preferable to enter both forms of the vector if they are known in a simple form. The user must be careful that the forms are mutually consistent. (For NP-tetrads, the object `testNP(bdn,bdn)` can be calculated to ensure that at least the basis vectors satisfy an NP inner product.

the numbers $1, \ldots, n$. For null tetrads, the basis vectors are labeled `l`, `n`, `m`, `mbar` respectively.

If the user has chosen to enter a general basis (Option 3 from the `makeg()` starting menu), the next prompts ask for the components of the inner product between the basis vectors. This is the contravariant two-tensor $\eta^{(a)(b)}$ whose $(a)(b)$ component is the value of the inner product of basis vectors a and b. In common applications, this tensor will have constant coefficients. The curvature tensors defined in the GRTensorII standard object libraries permit the use of inner products with non-constant coefficients, however it is required that the inner product be symmetric across the diagonal.

If a null tetrad has been selected (Option 4 from the starting menu), then the inner product between the basis vectors is assumed to be of the form specified in Eq. (1).

Once the inner product is selected (or chosen by default in the case of null tetrads) the basis vectors and inner product are both displayed so that they can be checked for errors in the input.

2.4 Other options

Once the data needed to specify the spacetime has been entered, a menu presents a number of additional options for correcting, saving, and supplementing the data. Typically the menu for metric entry looks like:

```
You may choose to 0) Use the metric without saving it,
                  1) Save the metric as it is,
                  2) Correct an element of the metric,
                  3) Re-enter the metric,
                  4) Add/change constraint equations,
                  5) Add a text description, or
                  6) Abandon this metric and return to Maple.
```

The options in this menu have obvious analogues for the cases of basis or null-tetrad input. Each option is described briefly below:

Use the metric without saving it: The metric is initialized in the current session as `g(dn,dn)`, and can thus be displayed via the command `grdisplay(g(dn,dn))`. The components are not saved to a file, and so will be lost when the MapleV session is ended. The components of the metric can be saved at a later time using the `grsaveg()` command.

Save the metric as it is: Any information that has been entered in this invocation of `makeg()` is saved to the file '*metricName*`.mpl`' (where *metricName* is the name specified in the argument to `makeg()`) and in the directory given by the global variable `grOptionMetricPath` or by the optional *metricPath* argument to `makeg()`.[5] The metric is initialized for the current session as in the previous option.

Correct an element of the metric: The user is prompted to enter the index values of the metric component to be corrected. This should take the form of a two-component MapleV list which contains coordinate names, eg. `[r,theta]`.

[5]For a description of the `grOptionMetricPath` variable, see Booklet *F: Installation and setup* or the online help page `?groptions`.

Re-enter the metric: The user is prompted to re-enter each component of the metric in succession as described in Section 2.1 above.

Add/change constraint equations: The user has the opportunity to add information taking the form of constraint equations to the metric. For example, consider a metric containing the function $m(r,t)$ which is required to satisfy the partial differential equations

$$\frac{\partial}{\partial r}m(r,t) = r^2 m(r,t), \quad \text{and} \quad \frac{\partial}{\partial t}m(r,t) = t^2 m(r,t)^2.$$

Choosing this option, the user is prompted to enter these constraint equations as a list:

```
[ diff ( m(r,t), r ) = r^2*m(r,t), diff ( m(r,t), t ) = t^2*m(r,t)^2 ]
```

These constraints could then be applied to objects calculated from this metric via `grcalc()` by using the appropriate options from `gralter()`. (See Section 5 and Booklet *C: Calculating tensor components*.)

Add a text description: A line of text describing the metric can be saved along with its components so that it may be more easily identified later. Such a note might include its full name, a journal reference, or some descriptive adjectives.[6]

The new metric created by `makeg()` automatically becomes the default metric to be used for subsequent calculations (see Section 3.1).

3 Loading spacetimes from a file

Metrics and bases created by `makeg()` are saved automatically to metric files in the default directory specified by the `grOptionMetricPath` variable. These metric files can be loaded in future sessions using `qload()` or `grload()`. Each of these is described below.

A directory of commonly used metric files is distributed with GRTensorII. A collection is also kept online [1]. The format of metric files is outlined in Section 8.1.

`qload (` *metricName* `)`

metricName – the name of the metric to be loaded.

Example: `> qload ( schw ):`

The `qload()` command searches the directories specified by the `grOptionqloadPath` global variable for the file *metricName*`.mpl`. If `grOptionqloadPath` is not assigned, or if the specified file is not found by one of the directories specified by `grOptionqloadPath`, then the directory specified by the global variable `grOptionMetricPath` is searched.

[6] Inclusion of such descriptions is strongly recommended as it can make large directories of metrics much more manageable. For instance, in Unix systems, some idea of the contents of each file in a metrics directory can be obtained using the command '`grep Info *.mpl`'. This command will list the text descriptions of each metric file (see Section 8.1).

`grload ( ` *metricName*`, ` *metricFile* ` )`

metricName – the name by which the new metric is to be referenced in the current session.

metricFile – the complete file name (including directory path and '`.mpl`' extension) of the metric file to be loaded.

Example: `> grload ( schwarzschild, ‘c:/mydir/metrics/schw.mpl‘ ):`

The `grload()` command loads the file specified by the *metricFile* command. For the remainder of the current session, the metric is referenced by the *metricName* parameter, which (unlike `qload()`'s) need not be related to the name of the metric file.

A spacetime loaded by `qload()` or `grload()` becomes the default metric to be used for subsequent calculations (see Section 3.1).

The following objects are initialized by the loading commands (as well as by `makeg()`), depending on the form of spacetime specified by the input file (ie. metric or basis):[7]

metric:	`ds`, `g(dn,dn)`
general covariant basis:	`eta(up,up)`, `e(bdn,dn)`, `basis(dn)`
general contravariant basis:	`eta(up,up)`, `e(bdn,up)`, `basis(up)`
covariant null tetrad:	`eta(up,up)`, `e(bdn,dn)`, `nullt(dn)`
contravariant null tetrad:	`eta(up,up)`, `e(bdn,up)`, `nullt(up)`

Additionally, for all of the above types of spacetime the objects

`x(up)`, `dimension`,

are also initialized, as well as

`Info`, `constraint`, `sig`,

if applicable.

See Booklet *C: Calculating tensor components* or the online help pages `?grt_objects` and `?grt_basis` for a description of these objects. The global variables `grOptionqloadPath` and `grOptionMetricPath` are described in Booklet *F: Installation and setup* and the `?groptions` online help page.

Once a metric name has been used in a session, it can not be re-used. For instance, if the `schw` spacetime has already been loaded, attempts to use the commands `makeg(schw)` or `qload(schw)` will fail. Metrics can be cleared from a session using the `grclear()` command, described in Booklet *C: Calculating tensor components*.

[7] In addition to the initialization of these objects, the following assumptions regarding the signature are made when loading a four-dimensional spacetime when the global `grOptionLLSC` variable is set `true`. If the spacetime is specified in the form of a metric, g_{ab}, or as a basis with a general inner product, then the signature is by default set to +2. If the spacetime is specified as a null tetrad satisfying an NP inner product, Eq. (1), then the signature is set to -2. The assumed signature of the spacetime is stored in the `sig` object and can be viewed by using the command `grdisplay(sig)`. In this version of GRTensorII, the value of the signature is only used in generating an NP tetrad via the `nptetrad()` command (see Section 6, below). If the `grOptionLLSC` variable is set to `false`, then `sig` is not initialized unless the signature is explicitly given in the metric file.

3.1 The default spacetime

A number of commands described in this booklet (`makeg()`, `qload()`, `grtransform()`, etc.) create new metrics or bases which describe a background geometry for which tensors can be calculated. Generally, once these commands are issued, the spacetime which they specify becomes the *default* spacetime (or background geometry) for future calculations. For instance, once the `qload(schw)` command is issued (the command to load the Schwarzschild metric), all subsequent calculations will assume this background metric until the default is changed.

If a number of different metrics have been created or loaded in single GRTensorII section, one can switch between them by using the `grmetric()` command, which has the form:

`grmetric (` *metricName* `)`

metricName – the name of a metric (or basis) which has been created or previously loaded in the current session.

Example: `> grmetric ( schw ):`

This command makes the metric named in its argument the default metric for subsequent calculations.

Alternatively, one can perform calculations using metrics other than the default metric by specifying the metric name as a parameter to the objects being calculated. The metric name is placed in square brackets following the name of the object. For instance,

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

calculates the covariant Ricci tensor for the `schw` spacetime, even if that spacetime is not the current default. See the Booklet *C: Calculating tensor components* for details on the use of the `grcalc()` command.

4 Coordinate transformations

The command `grtransform()` can be used to perform coordinate transformations of the metric tensor. It takes the following form:

`grtransform` (*oldmetric*, *newmetric*, *xform*, [*newsig*])

oldmetric – the name of the metric to be transformed.

newmetric – the name of by which the transformed metric is to be referenced in the future.

xform – a list specifying the transformation.

newsig – (*optional*) an integer indicating the signature of the transformed metric, if different from the original signature.

Example:
```
> grtransform ( schw, kruskal, xform := [
  u(r,t) = sqrt(r/(2*m)-1)*exp(r/(4*m))*cosh(t/(4*m)),
  v(r,t) = sqrt(r/(2*m)-1)*exp(r/(4*m))*sinh(t/(4*m)),
  Theta(theta) = theta, Phi(phi) = phi ] ):
```

The third argument, *xform*, is specified by a list of functions giving the old coordinates in terms of the new coordinates or vice versa. For instance if we are performing the transformation from coordinates (t, x, y, z) to new coordinates (τ, u, v, w), then we could specify *xform* as a set of functions of old in terms of new,

```
xform := [ t(tau,u,v,w)= ..., x(tau,u,v,w)= ..., y(tau,u,v,w)= ..., z(tau,u,v,w)
= ... ]:
```

or new in terms of old,

```
xform := [ tau(t,x,y,z)= ..., u(t,x,y,z)= ..., v(t,x,y,z)= ..., w(t,x,y,z)= ... ]:
```

Note that for an n-dimensional spacetime, n functions must be specified. Also, none of the new set of coordinates can be the same as any of the old coordinates. For instance, if both spacetimes are spherically symmetric, new coordinate names for the angles (θ, ϕ) must be given for the new spacetime, such as (Θ, Φ).

The following set of commands performs the transformation of the Schwarzschild spacetime from spherical to Kruskal coordinates:[8]

```
> qload ( schw ):
> xform := [ u(r,t) = sqrt(r/(2*m)-1)*exp(r/(4*m))*cosh(t/(4*m)),
             v(r,t) = sqrt(r/(2*m)-1)*exp(r/(4*m))*sinh(t/(4*m)),
             Theta(theta) = theta,
             Phi(phi) = phi ]:
> grtransform ( schw, kruskal, xform ):
```

It is not necessary that each of the transformation functions be specified as a function of all n variables, but each of the variables must be represented in at least one of the functions of the set. For instance, in the example above, at least one of the components of `xform` must be a function of t, at least one must be a function of r, etc.

[8]For a more complete presentation of this example see the demonstration file `kruskalo.ms`, which can be downloaded from the GRTensorII world-wide-web pages [1].

The transformed coordinates can be saved to a file using the `grsaveg()` command (see Section 8). The new metric created by the `grtransform()` command becomes the new default metric for subsequent calculations (see Section 3.1).

5 Constraint equations

The `makeg()` command provides the ability to save constraint equations with metric files. These are auxiliary equations which must be satisfied by functions contained in the metric and can later be applied to objects calculated from the metric or basis.

For example, in Kruskal's coordinates for the Schwarzschild spacetime,

$$ds^2 = 16\frac{m^2(r-2m)}{r(u^2-v^2)}(du^2 - dv^2) + r^2 d\Omega^2,$$

the function $r(u,v)$ is required to satisfy the differential relations

$$\frac{\partial r}{\partial u} = 4\frac{mu(r-2m)}{r(u^2-v^2)}, \qquad \frac{\partial r}{\partial v} = -4\frac{mv(r-2m)}{r(u^2-v^2)}.$$

Constraint equations can be included when the metric is created using `makeg()`. Alternatively, the command `grconstraint()` allows you to add or modify the constraints equations associated with a given metric. It takes the following form:

`grconstraint (` *metricName* `)`
metricName – the name of a previously loaded metric.
Example: `> grconstraint ( schw ):`

Constraint equations may be added to a metric, removed or re-arranged using this command. `grconstraint()` is menu driven and prompts for addition/modification/deletion of constraint equations.

The constraints are not invoked automatically during calculation of tensors, but must be applied explicitly using `gralter()` or `grcalcalter()` (see Booklet *C: Calculating tensor components*). A command sequence which would calculate the Ricci tensor for a metric and then apply the metric constraint equations to the result is:

```
> grcalc ( R(dn,dn) ):
> gralter ( R(dn,dn), consr ):
```

Note that constraint equations modified using `grconstraint()` are not saved automatically in the metric file. The command `grsaveg()` (see Section 8) should be used if the modified constraints are to be associated with the metric in future GRTensorII sessions. They will then be loaded automatically when the metric is loaded using `qload()` or `grload()`.

6 Creating a null tetrad from a metric

The command `nptetrad()` can be used to create a null tetrad (whose basis vectors satisfy the inner product given by the $\eta_{(a)(b)}$ of Eq. (1)) from a 4-dimensional metric, g_{ab}. The format of the command is:

> `nptetrad ( [`*lnSpace*`] )`
>
> *lnSpace* – (*optional*) a pair of coordinates which describe a preferred timelike 2-space in which the l and n vectors are to reside.
>
> Example: `> nptetrad ( [t,r] ):`

The command finds a set of null 1-forms satisfying the NP inner product, Eq. (1), for a given metric, g_{ab}. The forms are saved as the object `e(bdn,dn)`.

The algorithm for constructing the tetrad proceeds as follows. As a first step, a set of vectors corresponding to the columns of the covariant metric are used as a basis,

$$e_{(1)} := g_{1a}, \qquad e_{(2)} := g_{2a}, \qquad e_{(3)} := g_{3a}, \qquad e_{(4)} := g_{4a}.$$

If a pair of coordinates are listed as the optional *lnSpace* argument, then the corresponding vectors are used to construct the l and n vectors. For instance, if the coordinates of the spacetime are given as (t, r, θ, ϕ) and the argument is given as `[t,r]`, then the l and n vectors will be constructed as linear combinations of the vectors

$$\begin{aligned} e_{(1)} &:= [g_{t1}, g_{t2}, g_{t3}, g_{t4}], \\ e_{(2)} &:= [g_{r1}, g_{r2}, g_{r3}, g_{r4}]. \end{aligned}$$

The remaining pair are used to construct m and $\bar{m}$.

If the argument is omitted, a valid null tetrad will still be created, though the construction might be somewhat less efficient if the computer is left to find appropriate vectors on its own.[9] It will first attempt to locate null vectors among the columns of the metric. If none exist, then it picks a pairs of non-null column vectors and attempts to carry out an orthonormalization procedure to construct a tetrad satifying the NP inner product. If the first attempt fails (usually because it has chosen two spacelike vectors to construct l and n), it cycles through pairs of vectors until it is successful, or all unique combinations of columns of the metric have been exhausted, in which case a warning is issued.

A problem arises because of the need to take square roots in the normalization process. Because of the unpredictable usage of the imaginary number $i = \sqrt{-1}$ by the MapleV `sqrt()` command acting on symbolic expressions, it is not always possible for the algorithm to determine when a null tetrad is of the NP form or not. Occasionally a tetrad can be constructed which

[9] In previous versions of GRTensorII, the `nptetrad()` command required the specification of a timelike vector as input. The new algorithm makes better use of the coordinates of the spacetime (especially null coordinates) in constructing the tetrad. Note that in versions of GRTensorII previous to 1.50, the output of `nptetrad()` was in the form of the contravariant vectors `e(bdn,up)`$= e_{(a)}{}^b$. The natural form of output for the new algorithm is the covariant 1-forms, and thus the revised `nptetrad()` has as its output the components of `e(bdn,dn)` $= e_{(a)b}$.

satisfies the NP inner product, but for which l and n contain imaginary components, or for which the vector $\bar{m}$ is not actually the complex conjugate of m. Since the algorithm can not reliably test these criteria, it is the user's responsibility to check that the objects `e(bdn,dn)` conform to these properties of a true NP tetrad.[10]

The appearance of large terms in the components of the basis, especially terms containing radicals, is a common difficulty with tetrads produced by the `nptetrad()` command. To ensure that future calculations are optimized, it is important to express the basis in as simple a form as possible. To do this, apply the relevant `gralter()` commands, especially `radical` and `radsimp` to the basis vectors, `e(bdn,up)` until they seem to be fully simplified. (Simplification using `gralter()` is described more fully in Booklet *C: Calculating tensor components*).

Note that the implementation of the Newman-Penrose formalism in GRTensorII follows that of the original specification of [2]. As such, it requires that the spacetime has a -2 signature. This conflicts with the Landau-Lifshitz spacelike convention for which GRTensorII metrics are generally defined. If neccessary, the `nptetrad()` command will prompt the user, asking if the signature of the spacetime is to be reversed when constructing the tetrad. An affirmative response will cause the sign of the components of g_{ab} and g^{ab} to be reversed.[11] If the spacetime signature is anything other than +2 or -2, the `nptetrad()` command can not be used.

Tetrads created using `nptetrad()` can be saved using `grsaveg()` For more information regarding GRTensorII calculations in a null tetrad, see Booklet *E: Bases and tetrads*. The components of the basis created by `nptetrad()` can be saved using the `grsaveg()` command (Section 8).

6.1 Tetrad rotations

An alternative method of simplifying null tetrads is to perform a rotation of the basis vectors. Such rotations can be divided into three classes [2]:

Class I: – leaves basis vector l unchanged. This rotation is specified by a single complex-valued parameter, a. The basis vectors transform according to:

$$\begin{aligned} l &\longrightarrow l, \\ n &\longrightarrow n + a^*m + a\bar{m} + aa^*l, \\ m &\longrightarrow m + al, \\ \bar{m} &\longrightarrow \bar{m} + a^*l. \end{aligned}$$

Class II: – leaves basis vector n unchanged. This rotation is specified by a single complex-valued parameter, b. The basis vectors transform according to:

$$\begin{aligned} l &\longrightarrow l + b^*m + b\bar{m} + bb^*n, \\ n &\longrightarrow n, \\ m &\longrightarrow m + bn, \\ \bar{m} &\longrightarrow \bar{m} + b^*n. \end{aligned}$$

[10] It is the authors' experience that a true NP tetrad is always produced when the *lnSpace* argument is specified, or when the spacetime contains at least one null coordinate.

[11] Only the signatures of these objects are reversed. Objects previously calculated from the metric will not be updated to the new spacetime signature. For this reason, it is safest to run `nptetrad()` near the beginning of a session before further calculations with the metric are carried out in order to avoid sign conflicts.

Class III: – This rotation is specified by two real-valued parameters: θ determines a rotation in the $(m, \bar{m})$ plane, and A determines a boost in the n direction. The basis vectors transform according to:

$$\begin{aligned} l &\longrightarrow A^{-1} l, \\ n &\longrightarrow A n, \\ m &\longrightarrow \exp(i\theta) m, \\ \bar{m} &\longrightarrow \exp(-i\theta) \bar{m}. \end{aligned}$$

The `nprotate()` command takes the rotation class and its corresponding parameters as arguments, and performs the specified rotation on the basis vectors `e(bdn,up)`. The command has the form:

`nprotate (` *class*`,` *parm1*`,` *parm2* `)`

class – the rotation class, as defined above. This argument takes the value `I`, `II`, or `III`, depending on the desired rotation.

parm1 – the first rotation parameter, $\mathrm{Re}(a)$ for Class I rotations, $\mathrm{Re}(b)$ for Class II rotations, or A for Class III rotations.

parm2 – the second rotation parameter, $\mathrm{Im}(a)$ for Class I rotations, $\mathrm{Im}(b)$ for Class II rotations, or θ for Class III rotations.

Example: `> nprotate ( III, sqrt(2)*sqrt(1-2*m/r), 0 ):`

Note, in the current versions of GRTensorII, only rotations of the basis can be performed. The action of basis rotations on other curvature tensors has not yet been implemented. Thus, curvature tensors must be recalculated from the rotated basis.

The newly created tetrad is initialized as the default for subsequent calculations (see Section 3.1). The components of the rotated basis are not saved automatically, but can be saved for future used with the `grsaveg()` command (Section 8).

7 Metrics from an array

The `grnewmetric()` command defines a new metric by copying the components of a covariant two-index tensor to a new metric tensor. The format of the command is:

`grnewmetric ( `*newMetric*`, `*objectName*`, [`*coords*`] )`

newMetric – the name to be assigned to the newly created metric.

oldObject – the name of a two-index covariant GRTensor object.

coords – *(optional)* a list of coordinate names for the new metric, if they are different from those of the object from which it is created.

Example: `> grnewmetric ( confRW, confg(dn,dn) ):`

The newly created metric becomes the default metric for the current session. It is not saved automatically to a metric file, but can be saved using the `grsaveg()` command.

8 Saving modified spacetimes

The `grsaveg()` command is used to save the information associated with a given metric or basis. The format of the command is:

`grsaveg ( `*saveName*`, [`*metricPath*`] )`

saveName – the name (without the '`.mpl`' extension) of the file in which the metric (or basis) information is to be stored.

metricPath – (*optional*) the directory in which the the metric file is to be placed. If this argument is not specified the directory stored in the global option variable `grOptionMetricPath` is used instead.

Example: `> grsaveg ( newSchw ):`

The current default spacetime is saved to the file specified by *saveName*`.mpl` in the directory specified by the global `grOptionMetricPath` variable or the *metricPath* argument, if it is specified. The information saved includes:

coordinates,	`x(up)`,
metric components,	`g(dn,dn)` *(if they are assigned for the default spacetime)*,
basis components,	`e(bdn,dn)` and/or `e(bdn,up)` *(if assigned)*,
signature,	`sig` *(if assigned)*,
constraint equations,	`constraint` *(if assigned)*,
text description,	`Info` *(if assigned)*.

(See Booklet *C: Calculating tensor components* or the `?grt_objects` and `?grt_basis` online help pages for descriptions of these objects.)

If both metric components (`g(dn,dn)`) and basis components (`e(bdn,dn)` and/or `e(bdn,up)`) have been assigned for the current default spacetime, then the user is prompted as to which types

```
Ndim_ := 4:
x1_ := r:
x2_ := theta:
x3_ := phi:
x4_ := t:
sig_ := 2:
g11_ := diff ( R(r,t),r)^2/(1+f(r) ):
g22_ := R(r,t)^2:
g33_ := R(r,t)^2*sin(theta)^2:
g44_ := -1:
constraint_ :=[ diff(diff(R(r,t),r),t) = (2*diff(m(r),r)/R(r,t)
                - 2*m(r)*diff(R(r,t),r)/R(r,t)^2
                + diff(f(r),r))/(2*sqrt(2*m(r)/R(r,t)+f(r))),
                diff(R(r,t),t) = sqrt(2*m(r)/R(r,t)+f(r)),
                diff(diff(R(r,t),t),t) = -m(r)/R(r,t)^2,
                diff(diff(diff(R(r,t),t),r),t) = -diff(m(r),r)/R(r,t)^2 +
                2*m(r)*diff(R(r,t),r)/R(r,t)^3
              ]:
Info_:= `The Tolman dust solution (Proc.  Nat.  Acad.  Sci.  20, 169,1934)`:
```

Figure 1: The metric file `dust1.mpl` from the standard metric library.

of information are to be saved.

Note that MapleV *can not check* for the existence of files before it performs the `write` operation. Thus if a metric file with the specified name already exists in the `grOptionMetricPath` directory, then *it will be overwritten.*

8.1 Metric files

Metric files are simply ASCII files containing the coordinate components of the metric or basis. They can be modified (or created) with a text editor. By default, files are saved in the directory specified by the `grOptionMetricPath` variable with a file name of the form *metricName*`.mpl`.

Every metric or basis description file must contain a line '`Ndim_ :=` n`:`' where n is a number giving the dimension of the spacetime.

Every metric or basis file must contain a set of assignments to the variables `x1_`, `x2_`, ..., `x`n`_`, giving the names of the coordinates of the spacetime.

If the file describes a covariant metric, g_{ab}, it must contain a set of assignments to the variables `g11_`, `g12_`, ..., `g`nn`_`, giving the components of the metric. Only the upper diagonal of the metric needs to be specified. Any element of the upper diagonal that is not assigned is assumed to have the value zero.

If the file describes a set of basis vectors, it must contain assignments to the variables `b11_`, `b12_`, ..., `b`nn`_`, giving the components of contravariant basis vectors, or assignments to `bd11_`, `bd12_`, ..., `bd`nn`_`, in the case of covariant basis vectors. The inner product must also be specified by assignments to the variables `eta11_`, `eta12_`, ... , `eta`nn`_`. Variables which are not assigned are assumed to have value zero when the basis is loaded.

If the signature of the spacetime has been assigned, it is assigned to the variable `sig_` in the metric file. This variable is of the form of an integer $|s| \leq n$.

If the metric or basis possesses constraint equations (see above), these are represented as a MapleV list assigned to the variable `constraint_`. Text descriptions can be included by assigning a string to the name `Info_`.

An example of a metric file which specifies a spacetime using a covariant metric is given in Fig. (1). Many examples can be found in the metrics directory which is provided with the GRTensorII installation, as well as from the GRTensorII world-wide-web pages [1].

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] E. T. Newman and R. Penrose. An approach to gravitational radiation by a method of spin coefficients. *J. Math. Phys.*, 3:896–902, 1962. (Errata 4:998, 1963).

Commands described in this booklet:

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

`?grt_metrics`, `?grt_objects`, `?grt_basis`, `?groptions`, `?makeg`, `?qload`, `?grload`, `?grmetric`, `?grtransform`, `?grconstraint`, `?nptetrad`, `?nprotate`, `?grnewmetric`, `?grsaveg`.