GRG

Version 3.2

Computer Algebra System for
Differential Geometry,
Gravitation and
Field Theory

Vadim V. Zhytnikov

Moscow, 1992–1997 • Chung-Li, 1994

CHAPTER 1

Introduction

Calculation of various geometrical and physical quantities and equations is the usual technical problem which permanently arises in geometry, field and gravity theory. Numerous indices, contractions and components make these calculations very tedious and error-prone. Since this calculus obeys the well defined rules the idea to automate this kind of problems using computer is quite natural. Now there are several computer algebra systems such as MAPLE, REDUCE, MATHEMATICA or MACSYMA which in principle allow one to do this and it is not so hard to write a program to calculate, for example, the curvature tensor or connection. But suppose that we want to make a non-trivial coordinate transformation or tetrad rotation, calculate covariant or Lie derivative, compute a complicated expression with numerous contraction or raise or lower some indices. All these operations are typical in differential geometry and field theory but their realization with the help of general purpose computer algebra systems requires hard programming since all these systems really know nothing about *covariant properties* of geometrical quantities.

The computer algebra system GRG is designed in such a way to make calculation in differential geometry and field theory as simple and natural as possible. GRG is based on the computer algebra system REDUCE but GRG has its own simple input language whose commands resembles English phrases. Working with GRG no any knowledge of programming is required.

GRG understands tensors, spinors, vectors, differential forms and knows all standard operations with these quantities. Input form for mathematical expressions is very close to traditional mathematical notation including Einstein summation rule. GRG knows the covariant properties of these objects, you can easily raise and lower indices, compute covariant and Lie derivatives, perform coordinate and frame transformations. GRG works in any dimension and allows one to represent tensor quantities with respect to holonomic, orthogonal and even any other arbitrary frame.

One of the useful features of GRG is that it has a large number of built-in standard field-theory and geometrical quantities and formulas for their computation. Thus GRG provides ready solutions to many standard problems.

Another unique feature of GRG is that it can export results of calculations into other computer algebra system. You can save your data in to the file in the format of MAPLE, MATHEMATICA, MACSYMA or REDUCE in order to use this system to proceed analysis of the data. The LaTeX output format is supported as well. In addition GRG is compatible with REDUCE graphics shells providing niece book-quality output with Greek letters, integral signs etc.

The main built-in GRG capabilities are:

- Connection, torsion and nonmetricity.
- Curvature.
- Spinorial formalism.
- Irreducible decomposition of the curvature, torsion, and nonmetricity in any dimension.
- Einstein equations.
- Scalar field with minimal and non-minimal interaction.
- Electromagnetic field.
- Yang-Mills field.
- Dirac spinor field.
- Geodesic equation.
- Null congruences and optical scalars.
- Kinematics for time-like congruences.
- Ideal and spin fluid.
- Newman-Penrose formalism.
- Gravitational equations for the theory with arbitrary gravitational Lagrangian in Riemann and Riemann-Cartan spaces.

I would like to stress that current GRG version is intended for calculations in a concrete coordinate map only. It cannot operate with tensors as with objects having abstract symbolic indices.

This book consist of two main parts. First part contains detailed description of GRG as a programming system. Second part describes all built-in objects and formulas for their computation.

CHAPTER 2

Programming in GRG

Throughout the chapter `commands` are printed in typewriter font. The slanted serif-less font is used for command *parameters*. The optional parts of the commands are enclosed in squared brackets [option] and *id*[,*id*...] stands for one or several repetitions of *id*: *id* or *id*,*id* etc. Examples are separated form the text by horizontal lines ⌐¬ and the user input can be easily distinguished from the GRG output by the prompt `<-` which precedes every input line.

2.1 Session, Tasks and Commands

To start GRG it is necessary to start REDUCE and enter the command `load grg;`

On some systems you have to use `load!_package grg;` *since* `load` *is not defined.*

Sometimes it is better to use two commands `load grg32; grg;` *or* `load grg; grg;` *(See section* ?? *for details.)*

```
REDUCE 3.5, 15 Oct 93, patched to 15 Jun 95 ...

1: load grg;

This is GRG 3.2 release 2 (Feb 9, 1997) ...

System directory: c:\reduce\grg32\
System variables are upper-cased: E I PI SIN ...
Dimension is 4 with Signature (-,+,+,+)

<-
```

Symbol `<-` is the GRG prompt which shows that now GRG waits for your input. The GRG *task* (we prefer this term instead of usual *program*) consist of the sequence of commands terminated by semicolon `;`. Reading the input GRG splits it on *atoms*. There are several types of atoms:

- The identifier or symbol is a sequence of letters and digits starting with a letter:

```
i   I   alpha1   beta   ABC123D   Find
```

The identifiers in GRG may have trailing tilde character ~. Any other character may be incorporated in the identifier if preceded by the exclamation sign:

```
beta~   LIMIT!+
```

The identifiers in GRG play the role of the variables and functions in mathematical expressions and words in commands.

- Integer numbers

```
0   123   104341
```

- String is a sequence of characters enclosed in double quotes

```
"file.txt"   "This is a string"    "dir *.doc"
```

The strings in GRG are used for file names and operating system commands.

- Nine special two-character atoms

```
**   _|   /\   |=   ~~   ..   <=   >=   ->
```

- Any other characters are considered as single-character atoms.

The format of GRG commands is free. They can span one or several lines and any number of spaces and tabulations can be inserted between two neighbor atoms.

The GRG session may consist of several independent tasks. The command

```
Quit;
```

terminates both GRG and REDUCE session and returns the control to the operating system level. The command

```
Stop;
```

terminates current GRG task and brings the session control menu:

```
<- Stop;

    Quit GRG       - 0
    Start Task     - 1
    Exit to REDUCE - 2

  Type 0, 1 or 2:
```

The option `0` terminates REDUCE session similarly to the command `Quit;`. The choice `1` starts new task by bringing GRG to its initial state: all variables, declarations, substitutions and results of calculations are cleared and all switches resume their initial positions.† Finally the option `2` terminates GRG task and returns control to the REDUCE command level. In this case GRG can be restarted later by the command `grg;`.

The commands in GRG are case insensitive, i.e. command `Quit;` is equivalent to `quit;` and `QUIT;` etc. But notice that unlike REDUCE variables and functions in mathematical expressions in GRG *are case sensitive*.

2.1.1 Switches

Switches in GRG and REDUCE are used to control various system modes of operation. They are denoted by identifiers and the commands

```
On switch[,switch...];
Off switch[,switch...];
```

turns the *switch* on and off respectively. Any switch defined by REDUCE is available in GRG as well. In addition GRG defines a couple of its own switches. The full list of GRG switches is presented in appendix A. The command

```
[Show] Switch switch;
```

or equivalently

```
Show switch;
? switch;
```

prints current *switch* position

```
<- Show Switch TORSION;
TORSION is Off.
<- On torsion,gcd;
<- switch torsion;
TORSION is On.
<- switch exp;
GCD is On
```

Switches in GRG are case insensitive.

†Usually GRG does good job by resuming initial state and new task turns out to be independent of previous ones. But on some rare occasions the initial state cannot be completely recovered and it is better to restart REDUCE and GRG completely.

2.1.2 Batch File Execution

Usually GRG works in the interactive mode which is not always convenient. The command

```
[Input] "file";
```

reads the *file* and executes commands stored in it. The file names in GRG are always denoted by strings and exact specification of *file* is operating system dependent. The word `Input` is optional, thus in order to run batch file it suffices to enter its name `"`*file*`";`. The execution of batch file commands can be suspended by the command

```
Pause;
```

After this command GRG enters the interactive mode. One can enter one or several commands interactively and then resume batch file execution by the command

```
Next;
```

In general no any special end-of-file symbol or command is required in the GRG batch *file* but is necessary the symbol `$` is recognized by GRG as the end-of-file mark.

If during the batch file execution an error occurs GRG enter interactive mode and ask user to input the command which is supposed to replace the erroneous one. After the receiving of *one* command GRG automatically resumes the batch file execution. The command `Pause;` can be used if it is necessary to execute *several* commands instead of one.

The command

```
Output "outfile";
```

redirects all GRG output into the *outfile*. The *outfile* can be closed by the equivalent commands

```
EndO;
End of Output;
```

It is convenient to run long-time GRG tasks in background. The way of doing this depend on the operating system. For example to execute GRG task in background in UNIX it is necessary to use the following command

```
reduce < task.grg > grg.out &
```

Here we assume that the REDUCE invoking command is `reduce` and the file `task.grg` contains the GRG task commands:

```
load grg;
grg command;
grg command;
...
grg command;
quit;
```

The output of the session will be written into the file grg.out.

Since no proper reaction on errors is possible during the background execution it is good idea to turn the switch `BATCH` on. This makes GRG to terminate the session immediately in the case of any error.

2.1.3 Operating System Commands

The command

```
System "command";
```

executes the operating system *command*. The same command without parameters

```
System;
```

temporary suspends GRG session and passes the control to the operating system command level. The details may depend on the concrete operating system. In particular in UNIX the command `system;` may fail but UNIX has some general mechanism for suspending running programs: you can press `^Z` to suspend any program and `%+` to resume its execution.

2.1.4 Comments

The comment commands

```
Comment any text;
% any text;
```

are used to supply additional information to GRG tasks and data saved by the `Unload` command. The comment can be also attached to the end of any GRG command

See page `??` *about the* `Unload` *command.*

```
grg command % any text;
```

2.1.5 Timing

The command

```
[Show] Time;
```

prints time elapsed since the beginning of current GRG task including the percentage of so called garbage collections. The garbage collection time can be also printed by the command

```
[Show] GC Time;
```

If percentage of garbage collections grows and exceeds say 30% then memory of your system is running short and you probably need more RAM.

2.2 Declarations

Any object, variable or function in GRG must be declared. This allows to locate misprints and makes the system more reliable. Since GRG always work in some concrete coordinate system (map) the coordinate declaration is the most important one and must be present in every GRG task.

2.2.1 Dimension and Signature

See page ?? to find out how to change the default dimension and signature.

During installation GRG always defines default value of the dimension and signature. The information about this default value is printed upon GRG start in the form of the following (or similar) message line:

```
Dimension is 4 with Signature (-,+,+,+)
```

The following command overrides the default dimension and signature

```
Dimension dim with [Signature] (pm[,pm...]);
```

where *dim* is the number 2 or greater and *pm* is + or -. The *pm* can be preceded or succeeded by a number which denotes several repetitions of this *pm*. For example the declarations

```
Dimension 5 with Signature (+,+,-,-,-);
Dimension 5 with (2+,-3);
```

are equivalent and defines 5-dimensional space with the signature diag$(+1,+1,-1,-1,-1)$.

The important point is that the dimension declaration must be *very first in the task* and goes before any other command. Current dimension and signature

`E I PI INFINITY`	Mathematical constants e, i, π, ∞
`FAILED`	
`ECONST`	Charge of the electron
`DMASS`	Dirac field mass
`SMASS`	Scalar field mass
`GCONST`	Gravitational constant
`CCONST`	Cosmological constants
`LC0 LC1 LC2 LC3` `LC4 LC5 LC6` `MC1 MC2 MC3`	Parameters of the quadratic gravitational Lagrangian
`AC0`	Nonminimal interaction constant

Table 2.1: Predefined constants

can be printed by the command

```
[Show] Status;
```

2.2.2 Coordinates

The coordinate declaration command must be present in every GRG task

```
Coordinates id[,id...];
```

Only few commands such as informational commands, other declarations, switch changing commands may precede the coordinate declaration. The only way to have a tusk without the coordinate declaration is to load the file where coordinates where saved by the `Unload` command. but no any computation can be done before coordinates are declared. Current coordinate list can be printed by the command

See page **??** *to find out how to save data and declarations into a file.*

```
Write Coordinates;
```

2.2.3 Constants

Any constant must be declared by the command

```
Constants id[,id...];
```

The list of currently declared constants can be printed by the command

```
Write Constants;
```

There are also a number of built-in constants which are listed in table ??.

2.2.4 Functions

Functions in GRG are the analogues of the REDUCE *operators* but we prefer to use this traditional mathematical term. The function must be declared by the command

```
Functions f[(x[,x...])][,f[(x[,x...])]...];
```

Here `f` is the function identifier. The optional list of parameters `x` defines function with *implicit* dependence. The `x` must be either coordinate or constant. The construction `f(*)` is a shortcut which declares the function `f` depending on *all coordinates*.

The following example declares three functions `fun1`, `fun2` and `fun3`. The function `fun1`, which was declared without implicit coordinate list, must be always used in mathematical expressions together with the explicit arguments like `fun1(x+y)` etc. The functions `fun2` and `fun3` can appear in expressions in similar fashion but also as a single symbol `fun2` or `fun3`

```
<- Coordinates t, x, y, z;
<- Constant a;
<- Functions fun1, fun2(x,y), fun3(*);
<- Write functions;
Functions:

fun1 fun2(x,y) fun3(t,x,y,z)

<- d fun1(x+a);

DF(fun1(a + x),x) d x

<- d fun2;

DF(fun2,x) d x + DF(fun2,y) d y

<- d fun3;

DF(fun3,t) d t + DF(fun3,x) d x + DF(fun3,y) d y + DF(fun3,z) d z
```

The functions may have particular properties with respect to their arguments permutation and sign. The corresponding declarations are

```
Symmetric f[,f...];
Antisymmetric f[,f...];
Odd f[,f...];
Even f[,f...];
```

Notice that these commands are valid only after function *f* was declared by the command `Function`.

In addition to user-defined there is also large number of functions predefined in REDUCE. All these functions can be used in GRG without declaration. The complete list of these functions depends on REDUCE versions. Any function defined in the REDUCE package (module) is available too if the package is loaded before GRG was started or during GRG session. For example the package specfn contains definitions for various special functions.

See page **??** *to find out how to load the* REDUCE *packages.*

Finally there is also special declaration

```
Generic Functions f(a[,a...])[,f(a[,a...])...];
```

This command is valid iff the package dfpart.red is installed on your REDUCE system. Here unlike the usual function declaration the list of parameters must be always present and *a* can be any identifier preferably distinct from any other variable. The role of *a* is also completely different and is explained later.

See page **??** *to find out about the generic functions.*

The list of declared functions can be printed by the command

```
Write Functions;
```

Generic functions in this output are marked by the label `*`.

2.2.5 Affine Parameter

The variable which plays the role of affine parameter in the geodesic equation must be declared by the command

```
Affine Parameter s;
```

and can be printed by the command

```
Write Affine Parameter;
```

2.2.6 Case Sensitivity

Usually REDUCE is case insensitive which means for example that expression `x-X` will be evaluated by REDUCE as zero. On the contrary all coordinates, constants and functions in GRG are case sensitive, e.g. `alpha`, `Alpha` and `ALPHA` are all different. Notice that commands and switches in GRG 3.2 remain case insensitive.

Therefore all predefined by GRG constants and all built-in objects must be used exactly as they presented in this manual `GCONST`, `SMASS` etc. The situation with the constants and functions which predefined by REDUCE is different. The point is that in spite of its default case insensitivity internally REDUCE converts everything into some default case which may be upper or lower. Therefore depending on the particular REDUCE system they must be typed either as

```
E   I   PI   INFINITY   SIN   COS   ATAN
```

or in lower case

```
e   i   pi   infinity   sin   cos   atan
```

For the sake of definiteness throughout this book we chose the first upper case convention.

When GRG starts it informs you about internal case of your particular REDUCE system by printing the message

```
System variables are upper-cased: E I PI SIN ...
```

or

```
System variables are lower-cased: e i pi sin ...
```

You can find out about the internal case using the command

```
[Show] Status;
```

2.2.7 Complex Conjugation

By default all variables and functions in GRG are considered to be real excluding the imaginary unit constant `I` (or `i` as explained above). But if two identifiers differ only by the trailing character `~` they are considered as a pair of complex variables which are conjugated to each other. In the following example coordinates `z` and `z~` comprise such a pair:

```
<- Coordinates u, v, z, z~;

z & z~ - conjugated pair.

<- Re(z);

 z + z~
--------
   2

<- Im(z~);

 I*(z - z~)
------------
     2
```

2.3 Objects

Objects play a fundamental role in GRG. They represent mathematical quantities such as metric, connection, curvature and any other spinor or tensor geometrical and physical fields and equations. GRG has quite large number of built-in objects and knows many formulas for their calculation. But you are not obliged to use the built-in quantities and can declare your own. The purpose of the declaration is to tell GRG basic properties of a new quantity.

2.3.1 Built-in Objects

An object is characterized by the following properties and attributes:

- Name
- Identifier or symbol
- Type of the component

- List of indices
- Symmetries with respect to index permutation
- Density and pseudo-tensor property
- Built-in ways of calculation
- Value

The object *name* is a sequence of words which are usually the common English name of corresponding quantity. The name is case insensitive and is used to denote a particular object in commands. So called *group names* refer to a collection of closely related objects. In particular the name `Curvature Spinors` (see page **??**) refers to the irreducible components of the curvature tensor in spinorial representation. Actual content of the group may depend on the environment. In particular the group `Curvature Spinors` includes three objects in the Riemann space (Weyl spinor, traceless Ricci spinor and scalar curvature) while in the space with torsion we have six irreducible curvature spinors.

The object *identifier* or *symbol* is an identifier which denotes the object in mathematical expressions. Object symbols are case sensitive.

The object *type* is the type of its component: objects can be scalar, vector or p-form valued. The *density* and *pseudo-tensor* properties of the object characterizes its behaviour under coordinate and frame transformations.

Objects can have the following types of indices:

- Upper and lower holonomic coordinate indices.
- Upper and lower frame indices.
- Upper and lower spinorial indices.
- Upper and lower conjugated spinorial indices.
- Enumerating indices.

See page **??** *about the frame in* GRG.

The major part of GRG built-in objects has frame indices. The frame in GRG can be arbitrary but you can easily specify the frame to be holonomic or say orthogonal. Then built-in object indices become holonomic or orthogonal respectively.

See page **??** *about the spinorial formalism in* GRG.

GRG deals only with the SL(2,C) spinors which are restricted to the 4-dimensional spaces of Lorentzian signature. The corresponding SL(2,C) indices take values 0 and 1. The conjugated indices are transformed with the help of the complex conjugated SL(2,C) matrix. If some spinor is totally symmetric in the group of n spinorial indices (irreducible spinor) then these indices can

be replaced by a single so called *summed spinorial index* of rank n which take values from 0 to n. The summed spinorial indices provide the most economic way to store the irreducible spinor components.

Enumerating indices just label a collection of values and have no any covariant meaning. Accordingly there is no difference between upper and lower enumerating indices.

Notice that an index of any type in GRG always runs from 0 up to some maximal value which depend on the index type and dimensionality: $d-1$ for frame and coordinate indices, and n the spinor indices of the rank n.

GRG understands various types of index symmetries: symmetry, antisymmetry, cyclic symmetry and Hermitian symmetry. These symmetries can apply not only to single indices but to any group of indices as well. GRG uses object symmetries to decrease the amount of memory required to store the object components. It stores only components with the indices in certain *canonical* order and any other component are automatically restored if necessary by appropriate index permutation. The canonical order of indices is defined as follows: for symmetry, antisymmetry or Hermitian symmetry indices are sorted in such a way that index values grows from left to the right. For cyclic symmetry indices are shifted to minimize the numerical value of the whole list of indices.

Finally there are two special types of objects: equations and connection 1-forms. Equations have all the same properties as any other object but in addition they have left and right hand side and are printed in the form of equalities. The connections are used by GRG to construct covariant derivatives. There are only four types of connections: holonomic connection 1-form, frame connection 1-form, spinor connection 1-form and conjugated spinor connection 1-form.

See page **??** *about the connections.*

Almost all built-in objects have associated built-in *ways of calculation* (one or several). Each way is nothing but a formula which can be used to obtain the object value.

Every object can be in two states. Initially when GRG starts all objects are in *indefinite* state, i.e. nothing is known about their value. Since GRG always works in some concrete frame and coordinate system the object value is a table of the components. As soon as the value of certain object is obtained either by direct assignment or using some built-in formula (way of calculation) GRG remember this value and store it in some internal table. Later this value can be printed, re-evaluated used in expression etc. The object can be returned to its initial indefinite state using the command `Erase`. GRG uses object symmetries to reduce total number of components to store.

The complete list of built-in GRG objects is given in appendix C. The chapter 3 also describes built-in objects but in the usual mathematical style. The equivalent commands

```
Show object;
? object;
```

prints detailed information about the object *object* including object name, identifier, list of indices, type of the component, current state (is the value of an object known or not), symmetries and ways of calculation. Here *object* is either object name or its identifier.

The command

```
Show *;
```

prints complete list of built-in object names. This list is quite long and the command

```
Show c*;
```

gives list of objects whose names begin with the character *c* (a–z).

Finally the command

```
Show All;
```

prints list of objects whose values are currently known.

Notice that some built-in objects has limited scope. In particular some objects exists only in certain dimensionality, the quantities which are specific to spaces with torsion are defined iff switch TORSION is turned on etc.

Let us consider some examples. We begin with the curvature tensor $R^a{}_{bcd}$

```
<- Show Riemann Tensor;

Riemann tensor RIM'a.b.c.d is Scalar
  Value: unknown
  Symmetries: a(3,4)
  Ways of calculation:
    Standard way (D,OMEGA)
```

This object has name Riemann Tensor and identifier RIM. The object is Scalar (0-form) valued and has four frame indices. Frame indices are denoted by the lower-case characters and their upper or lower position are denoted by ' or . respectively. The Riemann tensor is antisymmetric in two last indices which is denoted by a(3,4).

The curvature 2-form $\Omega^a{}_b$

```
<- ? OMEGA;

Curvature OMEGA'e.f is 2-form
  Value: unknown
```

```
  Ways of calculation:
    Standard way (omega)
    From spinorial curvature (OMEGAU*,OMEGAD)
```

has name `Curvature` and the identifier `OMEGA` and is 2-form valued.

The traceless Ricci spinor (the quantity which is usually denoted in the Newman-Penrose formalism as $\Phi_{AB\dot{C}\dot{D}}$)

```
<- ? Traceless Ricci Spinor;

Traceless ricci spinor RC.AB.CD~ is Scalar
  Value: unknown
  Symmetries: h(1,2)
  Ways of calculation:
    From spinor curvature (OMEGAU,SD,VOL)
```

Spinorial indices are denoted by upper case characters with the trailing `~` for conjugated indices. Usual spinorial indices are denoted by a *single* upper case letter while summed indices are denoted by several characters. Thus, the traceless Ricci spinor has two summed spinorial indices of rank 2 each taking the values from 0 to 2. The spinor is hermitian `h(1,2)`.

The Einstein equation is an example of equation

```
<- ? Einstein Equation;

Einstein equation EEq.g.h is Scalar Equation
  Value: unknown
  Symmetries: s(1,2)
  Ways of calculation:
    Standard way (G,RIC,RR,TENMOM)
```

and 1-form $\Gamma^{\alpha}{}_{\beta}$ is an example of the connection

```
<- Show Holonomic Connection;



Holonomic connection GAMMA^x_y is 1-form Holonomic Connection
  Value: unknown
  Ways of calculation:
    From frame connection (T,D,omega)
```

The coordinate indices are denoted by the lower-case letters with labels `^` and `_` denoting upper and lower index position respectively. Notice that above the first "`Holonomic connection`" is the name of the object while second

"`Holonomic Connection`" means that GRG recognizes it as the connection and will use `GAMMA` to construct covariant derivatives for quantities having the coordinate indices. You can define any number of other holonomic connections and use them in the covariant derivatives on the equal footing with the built-in object `GAMMA`.

See page ?? about the covariant derivatives.

The notation in which command `Show` prints information about a particular object is the same as in the new object declaration and is explained in details below.

2.3.2 Macro Objects

There is also another class of built-in objects which are called *macro objects*. The main difference between the usual and macro objects is that macro quantities has no permanent storage to their components instead they are calculated dynamically only when its component is required in some expression. In addition they do not have names and are denoted only by the identifier only. Usually macro objects play auxiliary role. The complete list of macro objects can be found in appendix B.

The example of macro objects are the Christoffel symbols of second and first kind $\{^{\alpha}_{\beta\gamma}\}$ and $[_{\alpha,\beta\gamma}]$ having identifiers `CHR` and `CHRF` respectively

```
<- Show CHR;

CHR^x_y_z is Scalar Macro Object
  Symmetries: s(2,3)

<- ? CHRF;

CHRF_u_v_w is Scalar Macro Object
  Symmetries: s(2,3)
```

2.3.3 New Object Declaration

GRG has very large number of built-in quantities but you are not obliged to use them in your calculations instead you can define new quantities. The command

`New Object` *ID* [*ilst*] [`is` *ctype*] [`with` [`Symmetries`] *slst*] `;`

declares a new object. The words `New` or `Object` are optional (but not both) so the above command are equivalent to

```
Object ID [ilst] [is ctype] [with [Symmetries] slst];
New ID [ilst] [is ctype] [with [Symmetries] slst];
```

Here *ID* is an identifier of a new object. The identifier can contain letters `a`–`z`, `A`–`Z` but neither digits nor any other symbols. The identifier must be unique and cannot coincide with the identifier of any other built-in or user-defined object.

The *ilist* is the list of indices having the form

```
ipos itype[,ipos itype...]
```

where *ipos* defines the index position and *itype* specifies its type. The coordinate holonomic and frame indices are denoted by single lower-case letters with *ipos*

```
'  upper frame index
.  lower frame index
^  upper holonomic index
_  lower holonomic index
```

The frame and holonomic indices in GRG take values from 0 to $d-1$ where d is the current space dimensionality.

Spinorial indices are denoted by upper case letters with trailing `~` for conjugated spinorial indices: `A`, `B~` etc. Summed spinorial index of rank n is denoted by n upper-case letters. For example `ABC` denotes summed spinorial index of the rank 3 (runs from 0 to 3) and `AB~` denotes conjugated summed index of the rank 2 (values 0, 1, 2). The upper position for spinorial indices are denoted either by `'` or `^` and lower one by `.` or `_`.

Finally the enumerating indices are denoted by a single lower-case letter followed either by digits or by `dim`. For example the index declared as `i2` runs from 0 to 2 while specification `a13` denotes index whose values runs from 0 to 13. The specification `idim` denotes enumerating index which takes the values from 0 to $d-1$. Upper of lower position for enumerating indices are identical, thus in this case symbols `'` `.` `^` `_` are equivalent.

The *ctype* defines the type of new object component:

```
Scalar [Density dens]
p-form [Density dens]
Vector [Density dens]
```

This part of the declaration can be omitted and then the object is assumed to be scalar-valued. The *dens* defines pseudo-scalar and density properties of the object with respect to coordinate and frame transformations:

```
[sgnL][*sgnD][*L^n][*D^m]
```

where `D` and `L` is the coordinate transformation determinant $\det(\partial x^{\alpha'}/\partial x^{\beta})$

and frame transformation determinant $\det(L^a{}_b)$ respectively. If `sgnL` or `sgnD` is specified then under appropriate transformation the object must be multiplied on the sign of the corresponding determinant (pseudo tensor). The specification `L^`*n* or `D^`*m* means that the quantity must be multiplied on the appropriate degree of the corresponding determinant (tensor density). The parameters *p*, *n* and *m* may be given by expressions (must be enclosed in brackets) but value of these expressions must be always integer and positive in the case of *p*.

The symmetry specification *slst* is a list

slst1[`,`*slst1*...]

where each element *slst1* describes symmetries for one group of indices and has the form

sym`(`*slst2*[`,`*slst2*...]`)`

The *sym* determines type of the symmetry

`s`	symmetry
`a`	antisymmetry
`c`	cyclic symmetry
`h`	Hermitian symmetry

and *slst2* is either index number *i* or list of index numbers (`(`*i*[`,`*i*...]`)`) or another symmetry specification of the form *slst1*. Notice that nth object index can be present only in one of the *slst1*.

Let us consider an object having four indices. Then the following symmetry specifications are possible

`s(1,2,3,4)`	total symmetry
`a(1,2),s(3,4)`	antisymmetry in first pair of indices and symmetry in second pair
`s((1,2),(3,4))`	symmetry in pair permutation
`s(a(1,2),a(3,4))`	antisymmetry in first and second pair of indices and symmetry in pair permutation

The last example is the well known symmetry of Riemann curvature tensor. The specification `a(1,2),s(2,3)` is erroneous since second index present in both parts of the specification which is not allowed.

Declaration for new equations is completely similar

[`New`] `Equation` *ID* [*ilst*] [`is` *ctype*] [`with` [`Symmetries`] *slst*]`;`

GRG knows four types of connections:

- Frame Connection 1-form $\omega^a{}_b$ having first upper and second lower frame indices

- Holonomic Connection 1-form $\Gamma^{\alpha}{}_{\beta}$ having first upper and second lower coordinate indices
- Spinor Connection 1-form ω_{AB} with lower spinor index of rank 2
- Conjugated Spinor Connection $\omega_{\dot{A}\dot{B}}$ 1-form with lower conjugated spinor index of rank 2

Each of these connections are used to construct covariant derivatives with respect to corresponding indices. In addition they are properly transformed under the coordinate change and frame rotation. There are complete set of built-in connections but you can declare a new one by the command

```
[New] Connection ID'a.b [is 1-form];
[New] Connection ID^m_n [is 1-form];
[New] Connection ID.AB [is 1-form];
[New] Connection ID.AB~ [is 1-form];
```

Notice that any new connection must belong to one of the listed above types and have indicated type and position of indices. This representation of connection is chosen in GRG for the sake of definiteness.

There is one special case when new object can be declared without explicit `New Object` declaration. Let us consider the following example:

```
<- Coordinates t, x, y, z;
<- www=d x;
<- Show www;

www is 1-form
  Value: known
```

If we assign the value to some identifier *id* (`www` in our example) and this identifier is not reserved yet by any other object then GRG automatically declares a new object without indices labeled by the identifier *id* and having the type of the expression in the right-hand side of the assignment (1-form in our example). Notice that the *id* must not include digits since digits represent indices and any new object with indices must be declared explicitly.

See page **??** *about assignment command.*

The command

```
Forget ID;
```

completely removes the user-defined object with the identifier *ID*.

Finally let us consider some examples:

```
<- Coordinates t, x, y, z;
```

```
<- New RNEW'a.b_c_d is scalar density sgnD with a(3,4);
<- Show RNEW;

RNEW'a.b_x_y is Scalar Density sgnD
  Value: unknown
  Symmetries: a(3,4)

<- Null Metric;
<- Connection omnew.AA;
<- Show omnew;

omnew.AB is 1-form Spinor Connection
  Value: unknown
```

Here the first declaration defines a new scalar valued pseudo tensor $\texttt{RNEW}^{a}{}_{b\gamma\delta}$ which is antisymmetric in the last pair of indices. Second declaration introduce new spinor connection `omnew`. Notice that new connection is automatically declared 1-form and the type of connection is derived by the type of new object indices (lower spinorial index of rank 2 in our example).

2.4 Assignment Command

The assignment command sets the value to the particular components of the object. In general it has the form

[Name] comp = expr[,comp = expr...];

or for equations

[Name] comp = lhs=rhs[,comp = lhs=rhs...];

Here *Name* is the optional object name. If the object has no indices then *comp* is the object identifier. If the object has indices then *comm* consist of identifier with additional digits denoting indices. For example the following command assigns standard spherical flat value to the frame θ^a

```
Frame
  T0 = d t,
  T1 = d r,
  T2 = r*d theta,
  T3 = r*SIN(theta)*d phi;
```

and the command

```
RIM0123 = 100;
```

assigns the value to the $R^0{}_{123}$ component of the Riemann tensor. Notice that in this notation each digit is considered as one index, thus it does not work if the value of some index is greater than 9 (e.g. if dimensionality is 10 or greater). In this case another notation can be used in which indices are added to the object identifier as a list of digits enclosed in brackets

[Name] ID`(`*n*`[,`*n*`...])` `=` *expr*`;`

In particular the command

```
RIM(0,1,2,3) = 100;
```

is equivalent to the example above.

The assignment set value only to the certain components of an object leaving other components unchanged. But if before assignment the object was in indefinite state (no value is known) then assignment turns it to the definite state and all other components of the object are assumed to be zero.

The digits standing for object indices in the left-hand side of an assignment can be replaced by identifiers

[Name] ID`(`*id*`[,`*id*`...])` `=` *expr*`;`

Such assignment is called *tensorial* one. For example the following tensorial assignment set the value to the curvature 2-form $\Omega^a{}_b$

```
OMEGA(a,b) = d omega(a,b) + omega(a,m)/\omega(m,b);
```

This command is equivalent to $d \times d$ of assignments where `a` and `b` take values from 0 to $d-1$ (d is the space dimensionality). Notice that identifiers in the left-hand side of tensorial assignment must not coincide with any predefined or declared by the user constant or coordinate. It is possible to mix digits and identifiers:

```
FT(0,a) = 0;
```

Here `FT` is identifier of the built-in object `EM Tensor` which is the electromagnetic strength tensor F_{ab} and this command sets the electric part of the tensor to zero.

The assignment command takes into account symmetries of the objects. For example `EM Tensor` is antisymmetric and in order to assign value say to the components $F_{01} = -F_{10}$ it suffices to do this just for one of them

```
<- Coordinates t, x, y, z;
<- EM Tensor FT01=111, FT(3,2)=222;
<- Write FT;
```

```
EM tensor:

FT     = 111
   t x

FT     = -222
   y z
```

We can see that GRG automatically transforms indices to the *canonical* order. This rule works in the case or tensorial assignment as well

```
<- Coordinates t, x, y, z;
<- Function ff;
<- EM Tensor FT(a,b)=ff(a,b);
<- Write FT;
EM tensor:

FT     = ff(0,1)
   t x

FT     = ff(0,2)
   t y

FT     = ff(0,3)
   t z

FT     = ff(1,2)
   x y

FT     = ff(1,3)
   x z

FT     = ff(2,3)
   y z

<- FT(2,1);

 - ff(1,2)
```

In this case both parameters `a` and `b` runs from 0 to 3 but GRG assigns the value only to the components having indices in the canonical order `a<b`. GRG follows this rule also if in the left-hand side of tensorial assignment digits are mixed with parameters which may sometimes produce unexpected result:

```
<- Coordinates t, x, y, z;
<- Function ee;
```

```
<- FT(0,a)=ee(a);
<- Write FT;
EM tensor:

FT     = ee(1)
   t x

FT     = ee(2)
   t y

FT     = ee(3)
   t z

<- Erase FT;
<- FT(3,a)=ee(a);
<- Write FT;
EM tensor:

0
```

Observe the difference between these two assignments (the command `Erase FT;` destroys the previously assigned value). In fact second assignment assigns no values since `3` and `a` are not in the canonical order `3`$\geq$`a` for `a` running from 0 to 3. Notice the difference from the case when all indices in the left-hand side are given by the explicit numerical values. In this case GRG automatically transforms the indices to their canonical order and `FT(3,2)=222;` is equivalent to `FT(2,3)=-222;`.

Finally there is one more form of the tensorial assignment which can be applied to the summed spinorial indices. Let us consider the spinorial analogue of electromagnetic strength tensor Φ_{AB}. This spinor is irreducible (i.e. symmetric in $_{AB}$). The corresponding GRG built-in object `Undotted EM Spinor` (identifier `FIU`) has one summed spinorial index of rank 2. Let us consider two different assignment commands

```
<- Coordinates u, v, z, z~;

z & z~ - conjugated pair.

<- Null Metric;
<- Function ee;
<- FIU(a)=ee(a);
<- Write FIU;
Undotted EM spinor:
```

```
FIU  = ee(0)
   0

FIU  = ee(1)
   1

FIU  = ee(2)
   2

<- Erase FIU;
<- FIU(a+b)=ee(a,b);
<- Write FIU;
Undotted EM spinor:

FIU  = ee(0,0)
   0

FIU  = ee(0,1)
   1

FIU  = ee(1,1)
   2
```

In the first case `a` is treated as a summed index and runs from 0 to 2 but in the second case `a` and `b` are considered as usual single SL(2,C) spinorial indices each having values 0 and 1.

The notation for the object components in the left-hand side of assignment do not distinguishes upper and lower indices. Actually the indices are always assumed to be in the default position. You can always check the default index types and positions using the command `Show` *object*`;`. For example the `Riemann Tensor` has first upper and three lower frame indices and the command `RIM0123=100;` and `RIM(0,1,2,3)=100;` both assign value to the $R^0{}_{123}$ component of the tensor where indices are represented with respect to the current frame.

2.5 Geometry

The number of built-in objects in GRG is rather large. They all described in chapter 3 and appendices B and C. In this section we consider only the most important ones.

2.5.1 Metric, Frame and Line-Element

The line-element in GRG is defined by the following equation

$$ds^2 = g_{ab}\,\theta^a \otimes \theta^b \tag{2.1}$$

where $\theta^a = h^a_\mu dx^\mu$ is the frame 1-form and g_{ab} is the frame metric. The corresponding built-in objects are `Frame` (identifier `T`) and `Metric` (identifier `G`). There are also the "inverse" counterparts $\partial_a = h^\mu_a \partial_\mu$ (`Vector Frame`, identifier `D`) and g^{ab} (`Inverse Metric`, identifier `GI`). To determine the metric properties of the space you can assign some values to both the metric and the frame. There are two well known special cases. First is the usual coordinate formalism in which frame is holonomic $\theta^a = dx^\alpha$. In this case there is no difference between frame and coordinate indices. Another representation is known as the tetrad (in dimension 4) formalism. In this case frame metric equals to some constant matrix $g_{ab} = \eta_{ab}$ and significant information about line-element "is encoded" in the frame.

In general both metric and frame can be nontrivial but not necessarily. If no any value is given by user to the frame when GRG automatically assumes that frame is *holonomic*

$$\theta^a = dx^\alpha \tag{2.2}$$

Thus if we assign the value to metric only we automatically get standard coordinate formalism. On the contrary if no value is assigned to the metric then GRG automatically assumes

$$g_{ab} = \mathrm{diag}(+1, -1, \ldots) \tag{2.3}$$

where $+1$ and -1 on the diagonal of the matrix correspond to the current signature specification.

Notice that current signature is printed among other information by the command

```
[Show] Status;
```

and current line-element is printed by the command

```
ds2;
```

or equivalently

```
Line-Element;
```

Finally if neither frame nor metric are specified by user then both these quantities acquire default value and we automatically obtain flat space of the default signature:

```
<- Dimension 4 with Signature(-,+,+,+);
<- Coordinates t, x, y, z;
<- ds2;
Assuming Default Metric.
Metric calculated By default. 0.05 sec
Assuming Default Holonomic Frame.
Frame calculated By default. 0.05 sec

   2          2       2       2       2
 ds  =  -  d t  +  d x  +  d y  +  d z
```

2.5.2 Spinors

Spinorial representations exist in spaces of various dimensions and signatures but in GRG spinors are restricted to the 4-dimensional spaces of Lorentzian signature (−,+,+,+) or (+,−,−,−) only. Another restriction is that in the spinorial formalism the metric must be the *standard null metric*:

$$g_{ab} = g^{ab} = \pm \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.4}$$

where upper sign correspond to the signature (−,+,+,+) and lower sign to the signature (+,−,−,−). There is special command

```
Null Metric;
```

which assigns this standard value to the metric.

Thus spinorial frame (tetrad) in GRG must be null

$$ds^2 = \pm(-\theta^0 \otimes \theta^1 - \theta^1 \otimes \theta^0 + \theta^2 \otimes \theta^3 + \theta^3 \otimes \theta^2) \tag{2.5}$$

and conjugation rules for this tetrad must be

$$\overline{\theta^0} = \theta^0, \quad \overline{\theta^1} = \theta^1, \quad \overline{\theta^2} = \theta^3, \quad \overline{\theta^3} = \theta^2 \tag{2.6}$$

For the sake of efficiency the sigma-matrices $\sigma^a{}_{A\dot{B}}$ for such a tetrad are chosen in the simplest form. The only nonzero components of the matrices are

$$\sigma_0{}^{1\dot{1}} = \sigma_1{}^{0\dot{0}} = \sigma_2{}^{1\dot{0}} = \sigma_3{}^{0\dot{1}} = 1 \tag{2.7}$$

$$\sigma^0{}_{1\dot{1}} = \sigma^1{}_{0\dot{0}} = \sigma^2{}_{1\dot{0}} = \sigma^3{}_{0\dot{1}} = \mp 1 \tag{2.8}$$

2.5.3 Connection, Torsion and Nonmetricity

As was explained above GRG recognizes four types of connections: holonomic $\Gamma^{\alpha}{}_{\beta}$, frame $\omega^{a}{}_{b}$, spinorial ω_{AB} and conjugated spinorial $\omega_{\dot{A}\dot{B}}$. Accordingly there are four built-in objects: `Holonomic Connection` (id. `GAMMA`), `Frame Connection` (id. `omega`), `Undotted Connection` (id. `omegau`), `Dotted Connection` (id. `omegad`). Connections are used in GRG in covariant derivatives. In addition they are properly transformed under frame and coordinate transformations.

By default the connection in GRG are assumed to be Riemannian. In particular in this case holonomic connection is nothing but Christoffel symbols $\Gamma^{\alpha}{}_{\beta} = \{^{\alpha}_{\beta\pi}\}dx^{\pi}$. If it is necessary to work with torsion and/or nonmetricity then the switches `TORSION` and/or `NONMETR` must be turned on. In this case the Riemannian analogues or the aforementioned four connections are available as well.

See page **??** *about the built-in connections.*

2.6 Expressions

Expressions in GRG can be algebraic (scalar), vector or p-form valued. GRG knows all the usual mathematical operations on algebraic expressions, exterior forms and vectors.

2.6.1 Operations and Operators

The operations known to GRG are presented in the form of the table. Operations are subdivided into six groups separated by horizontal lines. Operations in each group have equal level of precedence and the precedence level decreases from the top to the bottom of the table. As in usual mathematical notation we can use brackets `( )` to change operation precedence.

Other constructions which can be used in expression are described below.

2.6.2 Variables and Functions

Operator listed in the table 2.2 act on the following types of the operands:

(i) integer numbers (e.g. `0`, `123`),

(ii) symbols or identifiers (e.g. `I`, `phi`, `RIM0103`),

(iii) functional expressions (e.g. `SIN(x)`, `G(0,1)` etc).

Valid identifier must belong to one of the following types:

Operation	**Description**	**Grouping**
[v_1,v_2]	Vector bracket	
@ x	Holonomic vector ∂_x	
d a d ω	Exterior differential	d ~a ⇔ (d(~a))
# a # ω	Dualization	
~ e	Complex conjugation	
a_1**a_2 a_1^ a_2	Exponention	
e / a	Division	e/a_1/a_2 ⇔ (e/a_1)/a_2
a * e	Multiplication	v|ω_1/\ω_2*a ⇳ v|(ω_1/\(ω_2*a))
v | a	Vector acting on scalar	
v _| ω	Interior product	
v_1 . v_2 v . o o_1 . o_2	Scalar product	
ω_1 /\ ω_2	Exterior product	
+ e	Prefix plus	
- e	Prefix minus	
e_1 + e_2	Addition	
e_1 - e_2	Subtraction	

Table 2.2: Operation and operators. Here: e is any expression, a is any scalar valued (algebraic) expressions, v is any vector valued expression, x is a coordinate, o is any 1-form valued expression, ω is any form valued expression.

- Coordinate.
- User-defined or built-in constant.
- Function declared with the implicit dependence list.
- Component of an object.

Any valid functional expression must belong to one of the following types:

- User-defined function.
- Function defined in REDUCE (operator).
- Component of built-in or user-defined object in functional notation.
- Some special functional expressions listed below.

2.6.3 Derivatives

The derivatives in GRG and REDUCE are written as

```
DF(a,x[,n][,x[,n]...])
```

where *a* is the differentiated expression, *x* is the differentiation variable and integer number *n* is the repetition of the differentiation. For example

$$\texttt{DF(f(x,y),x,2,y)} = \frac{\partial^3 f(x,y)}{\partial^2 x \partial y}$$

There are also another type of derivatives

```
DFP(a,x[,n][,x[,n]...])
```

They are valid only after `Generic Function` declaration if the package dfpart is installed on your system.

See section ?? about the generic functions.

2.6.4 Complex Conjugation

Symbol `~~` in the sum of terms is an abbreviation:

```
e + ~~ = e + ~e
e - ~~ = e - ~e
```

Functions `Re` and `Im` gives real and imaginary parts of an expression:

```
Re(e) = (e+~e)/2
Im(e) = I*(-e+~e)/2
```

2.6.5 Sums and Products

The following expressions represent sum and product

```
Sum(iter[,iter...],e)
Prod(iter[,iter...],e)
```

where *e* is the summed expression and *iter* defines summation variables. The range of summation can be specified by two methods. First "long" notation is

```
id = low..up
```

and the identifier *id* runs from *low* up to *up*. Both *low* and *up* can be given by arbitrary expressions but value of these expressions must be integer. The *low* can be omitted

```
id = up
```

and in this case *id* runs from 0 to *up*. The identifier *id* should not coincide with any built-in or user-defined variable.

In "short" notation *iter* is just identifier *id* and its range is determined using the following rules

- Mixed letter-digit *id* runs from 0 to $d-1$ where d is the space dimensionality.

  ```
  Aid  j2s
  ```

- The *id* consisting of lower-case letters runs from 0 to $d-1$

  ```
  j  a  abc  kkk
  ```

- The *id* consisting of upper-case letters runs from 0 to the number of letters in *id*, e.g. the following identifiers run from 0 to 1 and from 0 to 3 respectively

  ```
  B  ABC
  ```

- Letters with one trailing digit run from 0 to the value of this digit. Both *id* below runs from 0 to 3:

  ```
  j3  A3
  ```

- Letters with two digits run from the value of the first digit to the value of the second digit. The *id* below run from 2 to 3:

  ```
  j23  A23
  ```

- Letters with 3 or more digits are incorrect

```
j123
```

Two or more summation parameters are separated either by commas or by one of the relational operators

```
<    >    <=    =>
```

This means that only the terms satisfying these relations will be included in the sum. For example

$$\texttt{Sum(i24<=ABC,k=1..d-1,f(i24,ABC,k))} = \sum_{i=2}^{4} \sum_{\substack{a=0 \\ i \le a}}^{3} \sum_{k=1}^{d-1} f(i,a,k)$$

GRG's `Sum` and `Prod` should not be confused with REDUCE's `SUM` and `PROD` which are also available in GRG. GRG's `Sum` apply to any scalar, vector or form-valued expressions and always expanded by GRG into the appropriate explicit sum of terms. On the contrary `SUM` defined in REDUCE can be applied to the algebraic expressions only. GRG leaves such expression unchanged and passes it to the REDUCE algebraic evaluator. Unlike `Sum` the summation limits in `SUM` can be given by algebraic expressions. If value of these expressions is integer then result of the `SUM` will be the same as for `Sum` but if summation limits are symbolic sometimes REDUCE is capable to find a closed expression for such a sum but not always. See the following example

Use `SUM`, `PROD` *or* `sum`, `prod` *depending on* REDUCE *internal case as explained on page* **??**.

```
<- Coordinates t, x, y, z;
<- Function f;
<- Constants n, m;
<- Sum(k=1..3,f(k));

f(3) + f(2) + f(1)

<- SUM(f(n),n,1,3);

f(3) + f(2) + f(1)

<- SUM(n,n,1,m);

 m*(m + 1)
-----------
     2

<- SUM(f(n),n,1,m);

SUM(f(n),n,1,m)
```

2.6.6 Einstein Summation Rule

According to the Einstein summation rule if GRG encounters some unknown repeated identifier *id* then summation over this *id* is performed. The range of the summation variable is determined according to the "short" notation explained in the previous section.

2.6.7 Object Components and Index Manipulation

The components of built-in or user-defined object can be denoted in expressions by two methods which are similar to the notation used in the left-hand side of the assignment command. The first method uses the object identifier with additional digits denoting the indices `T0`, `RIM0213`. The second method uses the functional notation `T(0)`, `RIM(0,2,1,3)`, `OMEGA(j,k)`.

In functional notation the default index type and position can be changed using the markers: `'` upper frame, `.` lower frame, `^` upper holonomic, `_` lower holonomic. For example expression `RIM(a,b,m,n)` gives components of Riemann tensor with the default indices $R^a{}_{bmn}$ (first upper frame and three lower frame indices) while expression `RIM('a,'b,_m,_n)` gives $R^{ab}{}_{\mu\nu}$ with two upper frame and two lower coordinate indices. For enumerating indices position markers are ignored and only `'` and `.` works for spinorial indices.

See page ?? about spinorial formalism.

In the spinorial formalism each frame index can be replaced by a pair if spinorial indices according to the formulas:

$$A^a\sigma_a{}^{B\dot{D}} = A^{B\dot{D}}, \qquad B_a\sigma^a{}_{B\dot{D}} = B_{B\dot{D}}$$

Accordingly any frame index can be replaced by a pair of spinorial indices. Similarly one summed spinorial index or rank n can be replaced by n single spinor indices. There is only one restriction. If an object has several frame and/or summed spinorial indices then *all* must be represented in such expanded form. In the following example the null frame θ^a is printed in the usual and spinorial $\theta^{B\dot{C}}$ representations. The relationship $\theta^a\sigma_a{}^{B\dot{C}} - \theta^{B\dot{C}} = 0$ is verifies as well

```
<- Coordinates u, v, z, z~;

z & z~ - conjugated pair.

<- Null Metric;
<- Frame T(a)=d x(a);
<- ds2;
```

```
   2
 ds  =  (-2) d u d v + 2 d z d z~

<- T(a);

a=0 :  d u

a=1 :  d v

a=2 :  d z

a=3 :  d z~

<- T(B,C);

B=0 C=0 :  d v

B=0 C=1 :  d z~

B=1 C=0 :  d z

B=1 C=1 :  d u

<- T(a)*sigmai(a,B,C)-T(B,C);

0
```

2.6.8 Parts of Equations and Solutions

The functional expressions

LHS(*eqcomp*) RHS(*eqcomp*)

give access to the left-hand and right-hand side of an equation respectively. Here *eqcomp* is the component of the equation as explained in the previous section.

The LHS, RHS also provide access to the *n*'th solution if *eqcomp* is Sol(*n*). *See page ?? about solutions.*

2.6.9 Lie Derivatives

The Lie derivative is given by the expression

`Lie(`*v*`,`*objcomp*`)`

where *objcomp* is the component of an object in functional notation. For example the following expression is the Lie derivative of the metric $\pounds_v g_{ab}$

```
Lie(vec,G(a,b));
```

The index manipulations in the Lie derivatives are permitted. In particular the expression

```
Lie(vec,G(^m,b));
```

is the Lie derivative of the frame $\pounds_v g^{\mu}{}_{b} \equiv \pounds_v h^{\mu}_{a}$ and must vanish.

2.6.10 Covariant Derivatives and Differentials

The covariant differential

`Dc(`*objcomp*`[,`*conn*`[,`*conn*`...]])`

and covariant derivative

`Dfc(`*v*`,`*objcomp*`[,`*conn*`[,`*conn*`...]])`

Here *objcomp* is an object component in functional notation and *v* is a vector-valued expression. The optional parameters *conn* are the identifiers of connections. If *conn* is omitted then GRG uses default connection for each type of indices: frame, coordinate, spinor and conjugated spinor. If *conn* is indicated then GRG uses this connection instead of default one for appropriate type of indices. For example expression

See page ?? about the built-in connections.

```
Dc(OMEGA(a,b))
```

is the covariant differential of the curvature 2-form $D\Omega^{a}{}_{b}$. This expression should vanish in Riemann space and should be proportional to the torsion in Riemann-Cartan space. Here GRG will use default object `Frame connection` (id. `omega`). The expression

```
Dc(OMEGA(a,b),romega)
```

is similar but it uses another built-in connection `Riemann frame connection` (id. `romega`) which are different if torsion or nonmetricity are nonzero. The index manipulations are allowed in the covariant derivatives. For example the expression

```
Dfc(v,RIC(^m,_n))
```

gives the covariant derivative of the curvature of the Ricci tensor with first coordinate upper and second coordinate lower indices $\nabla_v R^{\mu}{}_{\nu}$.

2.6.11 Symmetrization

The functional expressions works iff the switch `EXPANDSYM` is on

```
Asy(i[,i...],e)
Sy(i[,i...],e)
Cy(i[,i...],e)
```

They produce antisymmetrization, symmetrization and cyclic symmetrization of the expression *e* with respect to *i* without corresponding $1/n$ or $1/n!$.

2.6.12 Substitutions

The expression

```
SUB(sub[,sub...],e)
```

is similar to the analogous expression in REDUCE with two generalizations: (i) it applies not only to algebraic but to form and vector valued expression *e* as well, (ii) as in `Let` command *sub* can be either the relation *l = r* or solution `Sub(n)`.

See page `??` *about solutions.*

2.6.13 Conditional Expressions

The conditional expression

```
If(cond,e1,e2)
```

chooses *e1* or *e2* depending on the value of the boolean expression *cond*.

Boolean expression appears in (i) the conditional expression `If`, (ii) in `For all Such That` substitutions. Any nonzero expression is considered as **true** and vanishing expression as **false**. Boolean expressions may contain the following usual relations and logical operations: `< > <= >= = |= not and or`. They also may contain the following predicates

`OBJECT(`*obj*`)`	Is *obj* an object identifier or not
`ON(`*switch*`)` `OFF(`*switch*`)`	Test position of the *switch*
`ZERO(`*object*`)`	Is the value of the *object* zero or not
`HASVALUE(`*object*`)`	Whether the *object* has any value or not
`NULLM(`*object*`)`	Is the *object* the standard null metric

Here *object* is an object identifier.

The expression `ERROR("`*message*`")` causes an error with the "*message*". It can be used to test any required conditions during the batch file execution.

2.6.14 Functions in Expressions

Any function which appear in expression must be either declared by the `Function` declaration or be defined in REDUCE (in REDUCE functions are called operators). In general arguments of functions in GRG must be algebraic expression with one exception. If one (and only one) argument of some function f is form-valued $\omega = adx + bdy$ then GRG applies f to the algebraic multipliers of the form $f(\omega) = f(a)dx + f(b)dy$. The same rule works for vector-valued arguments. Let us consider the example in the REDUCE operator `LIMIT` is applied to the form-valued expression

```
<- Coordinates t, x, y, z;
<- www=(x+y)^2/(x^2-1)*d x+(x+y)/(x-z)*d y;
<- www;

   2           2
  x  + 2*x*y + y            x + y
(-----------------) d x + (-------) d y
       2                    x - z
      x  - 1

<- LIMIT(www,x,INFINITY);

 d x +  d y
```

I would like to remind also that depending on the particular REDUCE system REDUCE operators must be used in GRG in upper `LIMIT` or lower case `limit`. See page **??** for more details.

Any function or operator defined in the REDUCE package can be used in GRG as well. Some examples are considered in section **??**.

2.6.15 Expression Evaluation

GRG evaluates expressions in several steps:

(1) All GRG-specific constructions such as `Sum`, `Prod`, `Re`, `Im` etc are explicitly expanded.

(2) If expression contains components of some built-in or user defined object they are replaced by the appropriate value. If the object is in indefinite state (no value of the object is known) then GRG tries to calculate its value by the method used by the `Find` command. The automatic object calculation can be prevented by turning the switch `AUTO` off. If due to some reason the object

See page **??** *about the* `Find` *command.*

cannot be calculated then expression evaluation is terminated with the error message.

(3) After all object components are replaced by their values GRG performs all "geometrical" operations: exterior and interior products, scalar products etc. If expression is form-valued when it is reduced to the form $a\,dx^0 \wedge dx^1 \ldots + b\,dx^1 \wedge + \ldots$ where a and b are algebraic expressions (similarly for the vector-valued expressions).

(4) The REDUCE algebraic simplification routine is applied to the algebraic expressions a, b. Final expression consist of exterior products of basis coordinate differentials $dx^i \wedge dx^j \ldots$ (or basis vectors ∂_{x^i}) multiplied by the algebraic expressions. The algebraic expressions contain only the coordinates, constants and functions.

In the anholonomic mode the basis $b^i \wedge b^j \ldots$ is used instead. See section **??**.

2.6.16 Controlling Expression Evaluation

There are many REDUCE switches which control algebraic expression evaluation. The number of these switches and details of their work depend on the REDUCE version. Here we consider some of these switches. All examples below are made with the REDUCE 3.5. On other REDUCE versions result may be a bit different.

Switches `EXP` and `MCD` control expansion and reduction of rational expressions to a common denominator respectively.

```
<- (x+y)^2;

 2            2
x  + 2*x*y + y

<- Off EXP;
<- (x+y)^2;

       2
(x + y)

<- On EXP;
<- 1/x+1/y;

 x + y
-------
  x*y

<- Off MCD;
<- 1/x+1/y;
```

```
 -1    -1
x   + y
```

These switches are normally on.

Switches PRECISE and REDUCED control evaluation of square roots:

```
<- SQRT(-8*x^2*y);

2*SQRT( - 2*y)*x

<- On REDUCED;
<- SQRT(-8*x^2*y);

2*SQRT(y)*SQRT(2)*I*x

<- Off REDUCED;
<- On PRECISE;
<- SQRT(-8*x^2*y);

2*SQRT(y)*SQRT(2)*I*x

<- On REDUCED, PRECISE;
<- SQRT(-8*x^2*y);

2*SQRT(y)*SQRT(2)*ABS(x)
```

Combining rational expressions the system by default calculates the least common multiple of denominators but turning the switch LCM off prevents this calculation.

Switch GCD (normally off) makes the system search and cancel the greatest common divisor of the numerator and denominator of rational expressions. Turning GCD on may significantly slow down the calculations. There is also another switch EZGCD which uses other algorithm for g.c.d. calculation.

Switches COMBINELOGS and EXPANDLOGS control the evaluation of logarithms

```
<- On EXPANDLOGS;
<- LOG(x*y);

LOG(x) + LOG(y)

<- LOG(x/y);
```

```
LOG(x) - LOG(y)

<- Off EXPANDLOGS;
<- On COMBINELOGS;
<- LOG(x)+LOG(y);

LOG(x*y)
```

By default all polynomials are considered by REDUCE as the polynomials with integer coefficients. The switches `RATIONAL` and `COMPLEX` allow rational and complex coefficients in polynomials respectively:

```
<- (x^2+y^2+x*y/3)/(x-1/2);

       2            2
 2*(3*x  + x*y + 3*y )
-----------------------
      3*(2*x - 1)

<- On RATIONAL;
<- (x^2+y^2+x*y/3)/(x-1/2);

  2    1         2
 x  + ---*x*y + y
       3
-------------------
           1
      x - ---
           2

<- Off RATIONAL;
<- 1/I;

 1
---
 I

<- (x^2+y^2)/(x+I*y);

  2    2
 x  + y
---------
 I*y + x
```

```
<- On COMPLEX;
<- 1/I;

 - I

<- (x^2+y^2)/(x+I*y);

x - I*y
```

Switch RATIONALIZE removes complex numbers from the denominators of the expressions but it works even if COMPLEX is off.

Turning off switch EXP and on GCD one can make the system to factor expressions

```
<- Off EXP;
<- On GCD;
<- x^2+y^2+2*x*y;

       2
(x + y)
```

Similar effect can be achieved by turning on switch FACTOR. Unfortunately this works only when GRG prints expressions and internally expressions remain in the expanded form. To make GRG to work with factored expressions internally one must turn on FACTOR and AEVAL. The GRG switch AEVAL make GRG to use an alternative REDUCE routine for algebraic expression evaluation and simplification. This routine works well with FACTOR on. Possibly it is good idea to turn switch AEVAL on by default. This can be done using GRG configuration files.

See section ?? about configuration files.

2.6.17 Substitutions

The substitution commands in GRG are the same as the corresponding REDUCE instructions

```
[For All x[,x...][Such That cond]] Let sub[,sub...];
[For All x[,x...][Such That cond]] Match sub[,sub...];
```

where *sub* is either relation *l* = *r* or the solution in the form `Sol(n)`. After the substitution is activated every appearance of *l* will be replaced by *r*. The `For All` substitutions have additional list of parameters *x* and will work for any value of *x*. The optional condition *cond* imposes restrictions on the value of the parameters *x*. The *cond* is the boolean expression (see page ??).

See page ?? about solutions.

The substitution can be deactivated by the command

[For All *x*[,*x*...][Such That *cond*]] Clear *sub*[,*sub*...];

Notice that the variables *x* must be exactly the same as in the corresponding `For All Let` command.

The difference between `Match` and `Let` is that the former matches the degrees of the expressions exactly while `Let` matches all powers which are greater than one indicated in the substitution:

```
<- Const a;
<- (a+1)^8;

 8      7       6       5       4       3       2
a  + 8*a  + 28*a  + 56*a  + 70*a  + 56*a  + 28*a  + 8*a + 1

<- Let a^3=1;
<- (a+1)^8;

    2
85*a  + 86*a + 85

<- Clear a^3;
<- Match a^3=1;
<- (a+1)^8;

 8      7       6       5       4       2
a  + 8*a  + 28*a  + 56*a  + 70*a  + 28*a  + 8*a + 57
```

Substitutions can be used for various purposes, for example: (i) to define additional mathematical relations such as trigonometric ones; (ii) to "assign" value to the user-defined and built-in constants; (iii) to define differentiation rules for functions.

After some substitution is activated it applies to every evaluated expression but value of the objects calculated *before* remain unchanged. The command `Evaluate` re-simplifies the value of the object

Evaluate *object*;

here *object* is the object name, or identifier, or the group object name. Let us consider a simple GRG task which calculates the volume 4-form of some metric

```
<- Coordinates t, x, y, z;
<- Constant a;
<- Tetrad T0=d t, T1=d x, T2=SIN(a)*d y+COS(a)*d z,
```

```
          T3=-COS(a)*d y+SIN(a)* d z;
<- Find and Write Volume;
Volume :

              2         2
VOL =  (SIN(a)  + COS(a) ) d t /\ d x /\ d y /\ d z
```

We see that REDUCE do not know the appropriate trigonometric rule. Thus we are going to apply substitution

```
<- For all x let SIN(x)^2 = 1-COS(x)^2;
<- Write Volume;
Volume :

VOL =  d t /\ d x /\ d y /\ d z
```

The situation has been improved. But actually, the *internal* representation of `VOL` remains unchanged. `Write` by default re-simplifies expressions before printing. By turning switch `WRS` off we can prevent this re-simplification:

```
<- Off WRS;
<- Write Volume;
Volume :
              2         2
VOL =  (SIN(a)  + COS(a) ) d t /\ d x /\ d y /\ d z
```

Now we can apply `Evaluate`:

```
<- Evaluate Volume;
<- Write Volume;
Volume :

VOL =  d t /\ d x /\ d y /\ d z
```

We see that the internal value of `VOL` now has been replaced by re-simplified expression.

Notice that the command

```
Evaluate All;
```

applies `Evaluate` to all objects whose value is currently known.

2.6.18 Generic Functions

Unfortunately REDUCE lacks the notion of partial derivative of a function. The expression `DF(f(x,y),x)` is treated by REDUCE as the "derivative of the

expression `f(x,y)` with respect to the variable `x`" rather than the "derivative of the function `f` with respect to its first argument". Due to this REDUCE cannot handle chain differentiation rule etc. This problem is fixed by the package dfpart written by H. Melenk. This package introduces notion of generic function and partial derivative `DFP`. If dfpart is installed on your REDUCE system GRG provides the interface to these facilities.

Let us consider an example. First we declare one usual and two generic functions

```
<- Coordinates t, x, y, z;
<- Function f;
<- Generic Function g(a,b), h(b);
<- Write Functions;
Functions:

g*(a,b) h*(b) f
```

Generic functions must be always declared with the list of parameters (`a` and `b` in our example). These parameters play the role of labels which denotes arguments of the generic function and the partial derivatives with respect to these arguments are defined. Due to this generic functions allow the chain differentiation rule

```
<- DF(f(SIN(x),y),x);

DF(f(SIN(x),y),x)

<- DF(g(SIN(x),y),x);

COS(x)*g (SIN(x),y)
        a
```

Here subscript `a` denotes the derivative of the function `g` with respect to the first argument. The operator `DFP` is introduced to denotes such derivatives in expressions:

```
<- DF(g(x,y)*h(y),b);

0

<- DFP(g(x,y)*h(y),b);

g (x,y)*h(y) + h (y)*g(x,y)
 b              b
```

If switch `DFPCOMMUTE` is turned on then `DFP` derivatives commute.

2.7 Using Built-in Formulas In Calculations

GRG has large number of built-in objects and almost each object has built-in formulas or so called *ways of calculation* which can be used to find the value of the object. This section explains how these formulas (ways) can be used.

2.7.1 `Find` *Command*

Almost each GRG built-in object has associated *ways of calculation*. Each way is nothing but a formula or equation which allows to compute the value of the object. All these formulas are described in the usual mathematical style in chapter 3. The command

`Show` *object*`;`

or equivalently

`?` *object*`;`

prints information about object's ways of calculation.

The command `Find` applies built-in formulas to calculate the object value

`Find` *object* [*way*]`;`

where *object* is the object name, or identifier, or group object name. The optional specification *way* indicates the particular way if the *object* has several built-in ways of calculation.

Consider the curvature 2-form $\Omega^a{}_b$ (object `Curvature`, id. `OMEGA`):

```
<- Show Curvature;

Curvature OMEGA'a.b is 2-form
  Value: unknown
  Ways of calculation:
    Standard way (omega)
    From spinorial curvature (OMEGAU*,OMEGAD)
```

We can see that this object has two built in ways of calculation. First way named `Standard way` is the usual equation $\Omega^a{}_b = d\omega^a{}_b + \omega^a{}_m \wedge \omega^m{}_b$. Second way under the name `From spinorial curvature` uses spinor $\longleftrightarrow$ tensor relationship to compute the curvature 2-form using its spinor analogues Ω_{AB} and $\Omega_{\dot{A}\dot{B}}$ as the source data. The ways of calculation are printed by the command

`Show` in the form

wayname (*SI*[,*SI*...])

where *wayname* is the way name and the *SI* are the identifiers of the *source* objects which are present in the right-hand side of the equation. The value of these objects must be known before the formula can be applied.

See Eq. (??) on page ??.

The *way* in the `Find` command allows one to choose the particular way which can be done by two methods. In the first form *way* is just the name exactly as it printed by the `Show` command

wayname

or `Using standard way` or `By standard way` if the way name is `Standard way`. Another method to specify the way is to indicate the appropriate source object

From *object* Using *object*

where *object* is the name or the identifier of the source object. For example second (spinorial) way of calculation for the curvature 2-form can be chosen by the following equivalent commands

```
Find curvature from spinorial curvature;
Find curvature using OMEGAU;
```

while first way is activated by the commands

```
Find curvature by standard way;
Find curvature using omega;
```

Recall that object identifiers are case sensitive and `omega` is the identifier of the frame connection 1-form $\omega^a{}_b$ and should not be confused with `OMEGA`.

The *way* specification in the `Find` can be omitted and in this case GRG uses the following algorithm to choose a particular way of calculation. Observe that the identifier of the undotted curvature 2-form Ω_{AB} is marked by the symbol $*$. This label marks so called *main* objects. If no way of calculation is specified when GRG tries to choose the way, browsing the way list form top to the bottom, for which the value of the *main* object is already known. If no switch way exists then GRG just picks up the first way in the list. Therefore in our example the command

```
Find curvature;
```

will use the second way if the value of the object Ω_{AB} (id. `OMEGAU`) is known and second way otherwise.

As soon as some way of calculation is chosen GRG tries to calculate the values of the source objects which are present in the right-hand side of corresponding equations. GRG tries to do this by applying the `Find` command without way specification to these objects. Thus a single `Find` can cause quite long chain of calculations. This recursive work is reflected by the appropriate tracing messages. The tracing can be eliminated by turning off switch `TRACE`.

Here we present the sample GRG session which computes curvature 2-form for the flat gravitational waves

---

```
<- Cord u, v, z, z~;

z & z~ - conjugated pair.

<- Null Metric;
<- Function H(u,z,z~);
<- Frame T0=d u, T1=d v+H*d u, T2=d z, T3=d z~;
<- ds2;

   2               2
 ds  =  ( - 2*H) d u  + (-2) d u d v + 2 d z d z~

<- Find Curvature;
Sqrt det of metric calculated. 0.16 sec
Volume calculated. 0.16 sec
Vector frame calculated From frame. 0.16 sec
Inverse metric calculated From metric. 0.16 sec
Frame connection calculated. 0.22 sec
Curvature calculated. 0.22 sec
<- Write Curvature;
Curvature:

     1
OMEGA   = ( - DF(H,z,2)) d u /\ d z + ( - DF(H,z,z~)) d u /\ d z~
      2

     1
OMEGA   = ( - DF(H,z,z~)) d u /\ d z + ( - DF(H,z~,2)) d u /\ d z~
      3

     2
OMEGA   = ( - DF(H,z,z~)) d u /\ d z + ( - DF(H,z~,2)) d u /\ d z~
      0
```

```
     3
OMEGA   = ( - DF(H,z,2)) d u /\ d z + ( - DF(H,z,z~)) d u /\ d z~
      0
```

Finally we want to emphasize that ways associated with some object may depend on the concrete environment. In particular the `Standard way` for the curvature 2-form is always available but second way which is essentially related to spinors works only in the 4-dimensional spaces of Lorentzian signature and iff the metric is null. If some way is not valid in the current environment it simply disappears from the way list printed by the `Show`.

See page ?? about the spinorial formalism.

It should be noted also that the `Find` *object*`;` command works only if the *object* is in the indefinite state and is rejected if the value of the *object* is already known. If you want to re-calculate the object then previous value must be cleared by the `Erase` command.

2.7.2 `Erase` *command*

The command

`Erase` *object*`;`

destroys the *object* value and returns it to initial indefinite state. It can be used also to free the memory.

2.7.3 `Zero` *command*

Command

`Zero` *object*`;`

assigns zero values to all *object* components.

2.7.4 `Normalize` *command*

Command

`Normalize` *object*`;`

applies to equations. It replaces equalities of the form $l = r$ by the equalities $l - r = 0$ and re-simplifies the result.

2.7.5 `Evaluate` *command*

The command

```
Evaluate object;
```

re-simplifies existing value of the *object*. This command is useful if we want to apply new substitutions to the object whose value is already known. The command

See page ?? *about substitutions.*

```
Evaluate All;
```

re-simplifies all objects whose value is currently known.

2.8 Printing Result of Calculations

2.8.1 `Write` *Command*

The command

```
Write object;
```

prints value of the *object*. Here *object* id the object name or identifier. Group names denoting a collection of several objects and macro object identifiers can be used in the `Write` command as well. In addition word `All` can be used to print all currently known objects.

See page ?? *about macro objects.*

The command `Write` can print declarations as well if *object* is `functions`, `constants`, or `affine parameter`.

The command

```
Write object[,object...] to "file";
```

or equivalently

```
Write object[,object...] > "file";
```

writes result into the "*file*". Notice that `Write` always destroys previous contents of the file. Therefore we have another command

```
Write to "file";
Write > "file";
```

which redirects all output into the file. The standard output can be restored by the commands

```
EndW;
End of Write;
```

By default `Write` re-simplifies the expressions before printing them. This

See page ?? *about substitutions.*

is convenient when substitutions are activated but slows down the printing especially for very large expressions. The re-simplification can be abolished by turning off switch `WRS`. If switch `WMATR` is turned on then GRG prints all 2-index scalar-valued objects in the matrix form

```
<- Coordinates t, x, y, z;
<- On wmatr;
<- Find and Write metric;
Assuming Default Metric.
Metric calculated By default. 0.06 sec
Metric:

[-1  0  0  0]
[            ]
[0   1  0  0]
[            ]
[0   0  1  0]
[            ]
[0   0  0  1]
```

`Write` prints frame, spinor and enumerating indices as numerical subscripts while holonomic indices are printed as the coordinate identifiers. If frame is holonomic and there is no difference between frame and coordinate indices then by default all frame indices are also labelled by the appropriate identifiers. But is switch `HOLONOMIC` is turned off they are still printed as numbers.

2.8.2 `Print` *Command*

The `Write` command described in the previous section prints value of an object. This value must be calculated beforehand by the `Find` command or established by the assignment. The command `Print` evaluates expression and immediately prints its value. It has several forms

> [`Print`] *expr* [`For` *iter*];
> `For` *iter* `Print` *expr*;

Here *expr* is expression to be evaluated and *iter* indicates that expression must be evaluated for several value of some variable. The specification *iter* is completely the same as is the `Sum` expression and is described in details in section **??** on page **??**. It consists of the list of parameters separated by commas `,` or relational operators `< > => =<`. For example the command

```
G(a,b) for a<b;
```

prints off-diagonal components of the metric.

Both word `Print` and `For` parts of the command can be omitted and it is possible just to enter an expression

expr`;`

and it will be evaluated and printed. The expression can contain indefinite identifiers and by default GRG treats them similarly to the variables in the `For` part of the `Print` command. The range of such parameters are determined by the short summation variable specification as explained on page **??**. For example the following four commands are equivalent. they all print the components of the holonomic metric $g_{\alpha\beta}$

```
Print g(a,b) for a,b;
For a,b Print g(a,b);
g(a,b) for a,b;
g(a,b);
```

Here the parameters `a`, `b` run from 0 to $d-1$.

Unfortunately such treatment of unknown variables may create some confusion since occasionally misprinted identifier may be recognizes by GRG as an iteration variable. If switch `NOFREEVARS` is turned on then GRG becomes more scrupulous and any unknown variable will cause the error.

2.8.3 Controlling the Output

There are several switches and commands which allow one to change output form of expressions. One needs to stress that all these facilities have no influence on the *internal form* of expressions, they alter the *printout only*.

Switches `ALLFAC` and command `Factor` control factoring of subexpressions. In the on default position `ALLFAC` makes the system search for a common factor and print it outside the expression. The command

`Factor` *expr*[`,`*expr*`...`]`;`

makes the system collect together terms with different powers of subexpressions *expr*. Command

`RemFac` *expr*[`,`*expr*`...`]`;`

removes the action of the previous `Factor` command.

```
<- Constants a,b,c;
<- a*(a+b+1)^2;
```

```
   2                 2
a*(a  + 2*a*b + 2*a + b  + 2*b + 1)

<-  Off ALLFAC;
<-  a*(a+b+1)^2;

 3      2        2      2
a  + 2*a *b + 2*a  + a*b  + 2*a*b + a

<-  Factor b;
<-  a*(a+b+1)^2;

 2           2           3      2
b *a + b*(2*a  + 2*a) + a  + 2*a  + a

<-  On ALLFAC;
<-  a*(a+b+1)^2;

 2                          2
b *a + 2*b*a*(a + 1) + a*(a  + 2*a + 1)
```

Normally REDUCE prints terms in some canonical order. The switch `REVPRI` prints terms in reverse order and command

`Order` *expr*[,*expr*...];

specifies the required order of subexpressions explicitly.

```
<-  Constants a,b,c;
<-  (a+b*c)^3;

 3      2            2  2    3  3
a  + 3*a *b*c + 3*a*b *c  + b *c

<-  On REVPRI;
<-  (a+b*c)^3;

 3  3        2  2      2        3
b *c  + 3*a*b *c  + 3*a *b*c + a

<-  Order c,a,b;
<-  (a+b*c)^3;

 3        2        2    2    3  3
a  + 3*c*a *b + 3*c *a*b  + c *b
```

```
<-  Off REVPRI;
<-  (a+b*c)^3;

 3  3      2    2        2      3
c *b  + 3*c *a*b  + 3*c*a *b + a
```

By default REDUCE prints fractions in two-dimensional format but turning off switch `RATPRI` prevents this facility. Switch `DIV` in the on position makes the system divide each term of the numerator by the denominator and to print the denominator in the form of negative powers. Switch `RAT` works in combination with the `Factor` command. In the on position it makes the system divide each term collected by the `Factor` in the numerator by the denominator.

```
<-  Const a,b,c;
<-  (a+b+1)^2/a;

  2                  2
 a  + 2*a*b + 2*a + b  + 2*b + 1
---------------------------------
                a

<-  Off RATPRI;
<-  (a+b+1)^2/a;

  2                  2
(a  + 2*a*b + 2*a + b  + 2*b + 1)/a

<-  On DIV;
<-  (a+b+1)^2/a;

     -1  2      -1      -1
a + a  *b  + 2*a  *b + a   + 2*b + 2

<-  Factor b;
<-  (a+b+1)^2/a;

 2  -1         -1             -1
b *a   + 2*b*(a   + 1) + a + a   + 2

<-  Off DIV;
<-  (a+b+1)^2/a;

  2                  2
(b  + 2*b*(a + 1) + a  + 2*a + 1)/a
```

```
<-  On RAT;
<-  (a+b+1)^2/a;

 2                       2
b /a + 2*b*(a + 1)/a + (a  + 2*a + 1)/a

<-  On RATPRI;
<-  (a+b+1)^2/a;

  2                    2
 b          a + 1     a  + 2*a + 1
---- + 2*b*------- + --------------
 a            a             a
```

One needs to realize that output form transformations may require a long time and memory expense. There is a special switch `PRI` which allows one to minimize this expense. If `PRI` is turned off then the system will print all expressions exactly in their internal form and output control does not work. This is the fastest way to print result of calculations.

The command `Line Length` *n*; sets the output line length to *n*.

2.8.4 LaTeX and Graphics Output

Some versions of REDUCE running under Windows, OS/2 or X-windows are equipped with the graphic shells which provide book-style output with Greek characters, integral signs etc. GRG is compatible with these systems. This graphic regime is activated by switch `FANCY`.

Graphic output mode internally uses some subset of the LaTeX language. Switch `LATEX` makes GRG to print the output in the LaTeX format. This output can be written into a file and later directly inserted in a document. Notice that turning off switch `LATEX` returns graphic output mode with switch `FANCY` on while turning off `FANCY` automatically turns off `LATEX` as well and returns usual character output mode.

In graphic regime the derivatives are printed in $\partial f/\partial x$ notation. Switch `DFINDEXED` makes the system to print derivatives in the indexed notation f_x.

The following expressions is the scalar curvature of the Bondi metric obtained by GRG and directly inserted in this manual

$$R \;=\; \big(4\,e^{2\,\beta+2\,\gamma}\,\cos(\theta)\,\frac{\partial\,U}{\partial\,r}\,r^2 \;-\; 8\,e^{4\,\beta}\,\cos(\theta)\,\frac{\partial\,\beta}{\partial\,\theta} \;-$$

$$4\,e^{2\,\beta+2\,\gamma}\,\cos(\theta)\,\frac{\partial\,\gamma}{\partial\,r}\,U\,r^2\,+\,12\,e^{4\,\beta}\,\cos(\theta)\,\frac{\partial\,\gamma}{\partial\,\theta}\,+$$
$$12\,e^{2\,\beta+2\,\gamma}\,\cos(\theta)\,U\,r\,+\,4\,e^{2\,\beta+2\,\gamma}\,\frac{\partial^2\,U}{\partial\,r\,\partial\,\theta}\,\sin(\theta)\,r^2\,+$$
$$e^{4\,\gamma}\,(\frac{\partial\,U}{\partial\,r})^2\,\sin(\theta)\,r^4\,+\,4\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,U}{\partial\,r}\,\frac{\partial\,\beta}{\partial\,\theta}\,\sin(\theta)\,r^2\,+$$
$$4\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,U}{\partial\,\theta}\,\frac{\partial\,\gamma}{\partial\,r}\,\sin(\theta)\,r^2\,+\,12\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,U}{\partial\,\theta}\,\sin(\theta)\,r\,-$$
$$4\,e^{2\,\beta+2\,\gamma}\,\frac{\partial^2\,V}{\partial\,r^2}\,\sin(\theta)\,r\,-\,8\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,V}{\partial\,r}\,\frac{\partial\,\beta}{\partial\,r}\,\sin(\theta)\,r\,-$$
$$8\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,V}{\partial\,r}\,\sin(\theta)\,+\,8\,e^{2\,\beta+2\,\gamma}\,\frac{\partial^2\,\beta}{\partial\,r\,\partial\,\theta}\,\sin(\theta)\,U\,r^2\,-$$
$$8\,e^{2\,\beta+2\,\gamma}\,\frac{\partial^2\,\beta}{\partial\,r^2}\,\sin(\theta)\,V\,r\,+\,8\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,\beta}{\partial\,r}\,\sin(\theta)\,V\,-$$
$$8\,e^{4\,\beta}\,\frac{\partial^2\,\beta}{\partial\,\theta^2}\,\sin(\theta)\,-\,12\,e^{4\,\beta}\,(\frac{\partial\,\beta}{\partial\,\theta})^2\,\sin(\theta)\,+\,16\,e^{4\,\beta}\,\frac{\partial\,\beta}{\partial\,\theta}\,\frac{\partial\,\gamma}{\partial\,\theta}\,\sin(\theta)\,-$$
$$8\,e^{2\,\beta+2\,\gamma}\,(\frac{\partial\,\gamma}{\partial\,r})^2\,\sin(\theta)\,V\,r\,+\,8\,e^{2\,\beta+2\,\gamma}\,\frac{\partial\,\gamma}{\partial\,r}\,\frac{\partial\,\gamma}{\partial\,\theta}\,\sin(\theta)\,U\,r^2\,+$$
$$4\,e^{4\,\beta}\,\frac{\partial^2\,\gamma}{\partial\,\theta^2}\,\sin(\theta)\,-\,8\,e^{4\,\beta}\,(\frac{\partial\,\gamma}{\partial\,\theta})^2\,\sin(\theta)\,+\,4\,e^{4\,\beta}\,\sin(\theta)\big)/$$
$$\big(2\,e^{4\,\beta+2\,\gamma}\,\sin(\theta)\,r^2\big)$$

2.8.5 Exporting Data Into Other Systems

Capabilities of major modern computer algebra systems are approximately equivalent but not quite. One system is better in doing one things and other is better for other purposes. It may happen that tools which you need are available only in one particular systems. GRG provides quite unique facility to export the data into other computer algebra systems. Turning on one of the following switches establishes the *output mode* in which all expressions are printed in the *input* language of other CAS. This output can be saved into a file and later you can use this CAS to proceed you analysis of the data. At present GRG supports five output modes which are controlled by the switches

`MACSYMA`	for Macsyma
`MAPLE`	for Maple
`MATH`	for Mathematica
`REDUCE`	for Reduce
`GRG`	for GRG

Notice the last switch allows one to print the data in the form which can be later inserted into GRG task.

2.9 Advanced Facilities

2.9.1 Solving Equations

GRG provides simple interface to the REDUCE algebraic equation solver. The command

```
Solve l=r[,l=r...] for expr[,expr...];
```

resolves equations *l*=*r* with respect to expressions *expr*. This command has also other form

```
Solve equation for expr[,expr...];
```

where *equation* is the name or identifier of some built-in or user-defined equation. Both form of the `Solve` command works with form and scalar valued equations as well but *expr* must be algebraic. The resulting solutions are stored in the special object `Solutions` (identifier `Sol`). They can be printed by the command

```
Write Solutions;
```

Left and right hand sides of *n*'th solution can be used in expression as `LHS(Sol(`*n*`))` or `RHS(Sol(`*n*`))`. The expression `Sol(`*n*`)` referring to the *n*'th solution can be used in the `SUB` and `Let` substitutions as well:

```
<- Coordinates t, x, y, z;
<- Solve x^2-2*x=5, y=9 for x, y;
<- Write Solutions;
Solutions:

Sol(0) : y = 9

Sol(1) : x =  - SQRT(6) + 1

Sol(2) : y = 9

Sol(3) : x = SQRT(6) + 1

<- SUB(Sol(1),(x-1)^2);

6

<- Let Sol(3);
<- (x-1)^2;

6
```

Solutions can be cleared by the command

```
Erase Solutions;
```

One need to stress that `Solve` is capable to solve algebraic relations only. Solving algebraic relations REDUCE knows already that the function `ASIN` is inverse to `SIN`. The command

```
Inverse f1,f2;
```

tells the system that functions *f1* and *f2* are inverse to each other.

2.9.2 Saving Data for Later Use

It is very convenient to have facilities to save results of calculations in a form fitted for restoring and further manipulation. For this purpose GRG has two special commands: `Unload` and `Load`.

The command

```
Unload object > "file";
Unload object To "file";
```

writes *object* value into "*file*" in some special format. Here *object* is name or identifier of an object.

The data can be later restored with help of the command

```
Load "file";
```

The command `Unload` always overwrites previous "*file*" contents. To save several objects in one file one must use the following sequence of commands

```
Unload > "file";
Unload object;
Unload object;
...
Unload object;
End Of Unload;
```

Here command `Unload > "file";` opens "*file*" and `End Of Unload;` closes it. The last command has the short form

```
EndU;
```

In fact presented above sequence of commands can be abbreviated as

```
Unload object[,object...] > "file";
```

One needs to stress that only the commands `Unload ...;` can be used between `Unload >` ... and `End Of Unload;`. If this rule does not hold then `Load` may fail to restore the file. The only additional command which can be used among these `Unload` *object*`;` commands is the comment `%` *text*`;`. This command insertes the comment *text* into the "*file*". Later when "*file*" will be restored by the `Load` the *text* message will be printed. This allows one to attach comments to unreadable files produced by `Unload` command.

As in other commands *object* in `Unload` command is either the name or identifier of an object. Names `Coordinates`, `Constants` and `Functions` can also be used to save declarations. And finally, the command

```
Unload All > "file";
```

saves all objects whose value is currently known and all declarations. Moreover, in the anholonomic basis mode this command saves full information about an anholonomic basis.

See section **??** *about anholonomic basis.*

When data or coordinates declarations are restored from a file they replace current values. Function and constant declarations are added to current declarations.

One should realize that serious troubles may appear when different coordinates are used in the current session and in the restored file. Even the order of coordinates is extremely important. We strongly recommend saving all declarations (especially coordinates) in addition to other objects. It ensures at least that will GRG print a warning message if some contradictions are detected between current declarations and declarations stored into a file. The best way to avoid these troubles is to use the command

```
Unload All > "file";
```

Loading the file saved by this command at the very beginning of a new GRG task completely restores the previous GRG state with all data and declarations.

Sometimes one needs to prevent the `Load`/`Unload` operations with coordinates. If switch `UNLCORD` is turned off (normally on) then all `Load` and `Unload` operations with coordinates are blocked.

Since `Unload` writes data in human-unreadable form there is the command

```
Show [File] "file";
```

or equivalently

```
? [File] "file";
File "file";
```

which prints short information about objects and declarations contained in the "*file*". It also prints comments contained in the file.

2.9.3 Coordinate Transformations

Command

```
New Coordinates new[,new...] with old=expr[,old=expr...];
```

introduces new coordinates *new* and defines how old coordinates *old* are expressed in terms of new ones. If the specified transformation is nonsingular GRG converts all existing objects to the new coordinate system.

The `New Coordinates` command properly transforms all objects having coordinate indices. The transformation of frame indices depend on the switch `HOLONOMIC`. In general case when frame is not holonomic then objects having frame indices remain unchanged and only their components are transformed into the new coordinate system. But if frame is holonomic then by default all frame indices are transformed similarly to the coordinate ones. Notice that in such situation the frame after transformation once again will be holonomic in the new coordinate system. But if switch `HOLONOMIC` is turned off the system distinguishes frame and coordinate indices in spite of the current frame type. In such situation the holonomic frame ceases to be holonomic after coordinate transformation.

2.9.4 Frame Transformations

Spinorial rotations are performed by the command

```
[Make] Spinorial Rotation [ ((expr00,expr01), (expr10,expr11))];
```

where expressions $expr_{AB}$ comprise the SL(2,C) transformation matrix

$$\phi'_A = L_A{}^B \phi_B, \quad expr_{AB} = L_A{}^B$$

If the specified matrix is really a SL(2,C) one then GRG performs appropriate transformation on all objects whose value is currently known.

Matrix specification in the command can be omitted

```
[Make] Spinorial Rotation;
```

In this case the SL(2,C) matrix $L_A{}^B$ must be specified as the value of a special object `Spinorial Transformation LS.A'B` (identifier `LS`).

Command for frame rotation is analogously

```
[Make] Rotation [ ((expr00,expr01,...), (expr10,expr11,...),...)];
```

with the nonsingular $d \times d$ rotation matrix

$$A'^a = L^a{}_b A^b, \quad expr_{ab} = L^a{}_b$$

GRG verifies that this matrix is a valid *rotation* by checking that frame metric g_{ab} *remains unchanged* under this transformation

$$g'_{ab} = L^m{}_a L^n{}_b g_{mn} = g_{ab}$$

Once again the matrix specification can be omitted and transformation $L^a{}_b$ can be specified as the value of the object `Frame Transformation L'a.b` (identifier `L`)

```
[Make] Rotation;
```

Frame rotation commands correctly transform frame and spinor connection 1-forms.

Finally, there is a special form of the frame transformation command

```
Change Metric [ ((expr00,expr01,...), (expr10,expr11,...),...)];
```

The only difference between this command and `Make Rotation` is that `Change Metric` does not impose any restriction on the transformation matrix and transformed metric does not necessary coincides with the original one.

Sometimes it is convenient to keep some object unchanged under the frame transformation. The command

```
Hold object;
```

makes the system to keep the *object* unchanged during frame and spinor transformations. The command

```
Release object;
```

discards the action of the `Hold` command.

2.9.5 Algebraic Classification

The command

```
Classify object;
```

performs algebraic classification of the *object* specified by its name or identifier. Currently GRG knows algorithms for classifying the following irreducible spinors

X_{ABCD}	Weyl spinor type
$X_{AB\dot{C}\dot{D}}$	Traceless Ricci spinor type
X_{AB}	Electromagnetic stress spinor type
$X_{A\dot{B}}$	Vector in the spinorial representation

See page ?? about summed spinor indices.

The `Classify` command can be applied to any built-in or user-defined object having one of the listed above types of indices. Notice that all spinors must be irreducible (totally symmetric in dotted and undotted indices) and $X_{AB\dot{C}\dot{D}}$, $X_{A\dot{B}}$ must be Hermitian. Groups of the irreducible indices must be represented as a single summed index.

GRG uses the algorithm by F. W. Letniowski and R. G. McLenaghan [Gen. Rel. Grav. 20 (1988) 463-483] for Petrov-Penrose classification of Weyl spinor X_{ABCD}. The obvious simplification of this algorithm is applied to the spinor analog of electromagnetic strength tensor X_{AB}. The spinor $X_{AB\dot{C}\dot{D}}$ is classified by the algorithm by G. C. Joly, M. A. H. McCallum and W. Seixas [Class. Quantum Grav. 7 (1990) 541-556, Class. Quantum Grav. 8 (1991) 1577-1585].

The classification process is accompanied by the tracing messages which can be eliminated by turning off the switch `TRACE`. On the contrary if one turns on the switch `SHOWEXPR` then GRG prints all expressions which appear during the classification to let you check whether the decision about nonvanishing of these expressions is really correct or not. This facility is important also in classifying $X_{AB\dot{C}\dot{D}}$ and $X_{A\dot{B}}$ since algebraic type for this objects may depend on the *sign* of some expressions which cannot be determined by GRG correctly.

2.9.6 REDUCE *Packages and Functions in* GRG

Any procedure or function defined in REDUCE package can be used in GRG. The package must be loaded either before GRG is started or during GRG session by one of the equivalent commands

```
[Use] Package package;
Load package;
```

where *package* is the package name. Notice that an identifier must be used for the package name unlike the `Load "file";` command described in section **??**. Let us consider some examples. The REDUCE package `specfn` contains definitions of various special functions and below we demonstrate 11th Legendre polynomial

```
<- Coordinates t, x, y, z;
<- package specfn;
<- LEGENDREP(11,x);

          10           8           6          4          2
 x*(88179*x   - 230945*x  + 218790*x  - 90090*x  + 15015*x  - 693)
-------------------------------------------------------------------
                               256
```

Another example demonstrates the taylor package

```
<- Coordinates t, x, y, z;
<- www=d(E^(x+y)*SIN(x));
<- www;

  x + y                              x + y
(E     *(COS(x) + SIN(x))) d x + (E     *SIN(x)) d y

<- load taylor;
<- TAYLOR(www,x,0,5);

                          y         y
  y      y      y  2    E    4    E    5      6           y      y  2
(E  + 2*E *x + E *x  - ----*x  - ----*x  + O(x )) d x + (E *x + E *x
                          6         15

     y         y
    E    3    E    5      6
 + ----*x  - ----*x  + O(x )) d y
    3         30
```

You can also define your own operators and procedures in REDUCE and later use them in GRG. In the following example file lasym.red contains a definition of little REDUCE procedure which computes a leading term of asymptotic expansion of the rational function at large values of some variable. This file is inputted in REDUCE before GRG is started

```
1: in "lasym.red";

procedure leadingterm(w,x);
  lterm(num(w),x)/lterm(den(w),x);

leadingterm

end;

2: load grg;

This is GRG 3.2 release 2 (Feb 9, 1997) ...

System directory: c:\red35\grg32\
System variables are upper-cased: E I PI SIN ...
```

```
Dimension is 4 with Signature (-,+,+,+)

<- Coordinates t, r, theta, phi;
<- OMEGA01=(123*r^3+2*r+t)/(r+t)^5*d theta/\d phi;
<- OMEGA01;

                    3
                 123*r  + 2*r + t
(-------------------------------------------------) d theta /\ d phi
   5      4         3  2       2  3        4    5
  r  + 5*r *t + 10*r *t  + 10*r *t  + 5*r*t  + t

<- LEADINGTERM(OMEGA01,r);

  123
(-----) d theta /\ d phi
   2
  r
```

2.9.7 Anholonomic Basis Mode

GRG may work in both holonomic and anholonomic basis modes. In the first default case, values of all expressions are represented in a natural holonomic (coordinate) basis: dx^μ, $dx^\mu \wedge x^\nu \ldots$ for exterior forms and $\partial_\mu = \partial/\partial x^\mu$ for vectors. In the second case an arbitrary basis $b^i = b^i_\mu dx^\mu$ is used for forms and inverse vector basis $e_i = e^\mu_i \partial_\mu$ for vectors ($b^i_\mu e^\mu_j = \delta^i_j$). You can specify this basis assigning a value to built-in object `Basis` (identifier `b`). If `Basis` is not specified by user then GRG assumes that it coincides with the frame $b^i = \theta^i$.

Frame should not be confused with basis. Frame θ^a is used only for "external" purposes to represent tensor indices while basis b^i and vector basis e_i is used for "internal" purposes to represent form and vector valued object components.

The command

```
Anholonomic;
```

switches the system to the anholonomic basis mode and the command

```
Holonomic;
```

switches it back to the standard holonomic mode.

Working in anholonomic mode GRG creates some internal tables for efficient calculation of exterior differentiation and complex conjugation. In anholonomic mode the command

```
Unload All > "file";
```

automatically saves these tables into the "*file*". Subsequent

```
Load "file";
```

restores the tables and automatically switches the current mode to anholonomic one. Note that automatic anholonomic mode saving/restoring works only if `All` is used in `Unload` command.

One can find out the current mode with the help of the command

```
[Show] Status;
```

2.9.8 Synonymy

Sometimes GRG commands may be rather long. For instance, in order to find the curvature 2-form Ω_{ab} from the spinorial curvature Ω_{AB} and $\Omega_{\dot{A}\dot{B}}$ the following command should be used

```
Find Curvature From Spinorial Curvature;
```

Certainly, this command is clear but typing of such long phrases may be very dull. GRG has synonymy mechanism which allows one to make input much shorter.

The synonymous words in commands and object names are considered to be equivalent. The complete list of predefined GRG synonymy is given in appendix D. Here we present just the most important ones

```
Connection Con
Constants Const Constant
Coordinates Cord
Curvature Cur
Dotted Do
Equation Equations Eq
Find F Calculate Calc
Functions Fun Function
Next N
Show ?
Spinor Spin Spinorial Sp
Switch Sw
Symmetries Sym Symmetric
Undotted Un
Write W
```

Words in each line are considered as equivalent in all commands. Thus the above command can be abbreviated as

```
F cur from sp cur;
```

Section ?? explains how to change built-in synonymy and how to define a new one.

2.9.9 Compound Commands

Sometime one may need to perform several consecutive actions with one object. In this case we can use so called *compound commands* to shorten the input. Internally GRG replaces each compound command by several usual ones. For example the compound command

```
Find and Write Einstein Equation;
```

to a pair of usual ones

```
Find Einstein Equation;
Write Einstein Equation;
```

Actions (commands) can be attached to the end of the compound command as well:

```
Find, Write Curvature and Erase It;
      ⇕
Find & Write & Erase Curvature;
      ⇕
Find Curvature;
Write Curvature;
Erase Curvature;
```

Note that we have used `,` and `&` instead of `and` in this example. All these separators are equivalent in compound commands.

Now let us consider the case when one needs to perform a single action with several objects. The command

```
Write Frame, Vector Frame and Metric;
```

is equivalent to

```
Write Frame;
Write Vector Frame;
Write Metric;
```

Way specification can be attached to the `Find` command:

```
Find QT, QP From Torsion using spinors;
      ⇳
Find QT From Torsion using spinors;
Find QP From Torsion using spinors;
```

One can combine several actions and several objects. For example, the command

```
Find omega, Curvature by Standard Way and Write and Erase Them;
```

is equivalent to the sequence of (2 objects) × (3 commands) = 6 commands

```
Find omega by Standard Way;
Find Curvature by Standard Way;
Write omega;
Write Curvature;
Erase omega;
Erase Curvature;
```

Note that the way specification is attached only to "left" commands (`Find` in our case).

The compound commands mechanism works only with `Find`, `Erase`, `Write` and `Evaluate` commands.

And finally, GRG always replaces `Re-`*command*`;` by `Erase and` *command*`;`. For example

```
Re-Calculate Maxwell Equations;
      ⇳
Erase and Calculate Maxwell Equations;
```

You can see how GRG expand compound commands into the usual ones by turning switch `SHOWCOMMANDS` on.

2.10 Tuning GRG

GRG can be tuned according to your needs and preferences. The configuration files allow one to change some default settings and the environment variable `grg` defines the system directory which can be used as the depository for frequently used files.

2.10.1 Configuration Files

The configuration files allows one to establish

- Default dimension and signature.
- Initial position of switches.
- REDUCE packages which must be preloaded.
- Synonymy.
- Default GRG start up method.

There are two configuration files. First *global* configuration file grgcfg.sl defines the settings during system installation when GRG is compiled. These global settings become permanent and can be changed only if GRG is recompiled. The *local* configuration file grg.cfg allows one to override global settings locally. When GRG starts it search the file grg.cfg in current directory (folder) and if it is present uses the corresponding settings.

Below we are going to explain how to change settings in both global and local configuration files but before doing this we must emphasize that this need some care. First, the configuration files use LISP command format which differs from usual GRG commands. Second, is something is wrong with configuration file then no clear diagnostic is provided. Finally, if global configuration is damaged you will not be able to compile GRG. The best strategy is to make a back-up copy of the configuration files before start editing them. Notice that lines preceded by the percent sign % are ignored by the system (comments).

Both local grg.cfg and global grgcfg.sl configuration files have similar structure and can include the following commands.

Command

```
(signature!> - + + + +)
```

establishes default dimension 5 with the signature (−,+,+,+,+). Do not forget ! and spaces between + and -. This command *must be present* in the global configuration file grgcfg.sl otherwise GRG cannot be compiled.

The commands

```
(on!> page)
(off!> allfac)
```

change default switch position. In this example we turn on the switch PAGE (this switch is defined in DOS REDUCE only and allows one to scroll back and forth through input and output) and turn off switch ALLFAC.

The command

```
(package!> taylor)
```

makes the system to load REDUCE package taylor during GRG start.

The command of the form

```
(synonymous!>
  ( affine aff                          )
  ( antisymmetric asy                   )
  ( components comp                     )
  ( unload save                         )
)
```

defines synonymous words. The words in each line will be equivalent in all GRG commands.

Finally the command

```
(setq ![autostart!] nil)
```

alters default GRG start up method. It makes sense only in the global configuration file grgcfg.sl. By default GRG is launched by single command

```
load grg;
```

which firstly load the program into memory and then automatically starts it. Unfortunately on some systems this short method does not work properly: GRG shows wrong timing during computations, the `quit;` command returns the control to REDUCE session instead of terminating the whole program. If the aforementioned option is activated then GRG must be launched by two commands

```
load grg;
grg;
```

which fixes the problems. Here first command just loads the program into memory and second one starts it manually. Notice that one can always use commands

```
load grg32;
grg;
```

to start GRG manually. Command `load grg32;` always loads GRG into memory without starting it independently on the option under consideration.

2.10.2 System Directory

The environment variable `grg` or `GRG` defines so called GRG system directory (folder). The way of setting this variable is operating system dependent. For example the following commands can be used to set `grg` variable in DOS, UNIX and VAX/VMS respectively:

```
set grg=d:\xxx\yyy\                 DOS
setenv grg /xxx/yyy/                UNIX
define grg SYS$USER:[xxx.yyy]       VAX/VMS
```

The value of the variable `grg` must point out to some directory. In DOS and UNIX the directory name must include trailing `\` or `/` respectively. The command

```
[Show] Status;
```

prints current system directory.

When GRG tries to input some batch file containing GRG commands it first searches it in the current working directory and if the file is absent then it tries to find it in the system directory. Therefore if you have some frequently used files you can define the system directory and move these files there. In this case it is not necessary to keep them in each working directory. Notice GRG uses the same strategy when opening local configuration file grg.cfg. Thus if system directory is defined and it contains the file grg.cfg the settings contained in this file effectively overrides global settings without recompiling GRG.

2.11 Examples

In this section we want to demonstrate how GRG can be applied to solve simple but realistic problem. We want to calculate the Ricci tensor for the Robertson-Walker metric by three different methods.

First GRG task (program)

```
Coordinates t,r,theta,phi;
Function a(t);
Frame T0=d t, T1=a*d r, T2=a*r*d theta, T3=a*r*SIN(theta)*d phi;
ds2;
Find and Write Ricci Tensor;
RIC(_j,_k);
```

defines the Robertson-Walker metric using the tetrad formalism with the orthonormal Lorentzian tetrad θ^a. Using built-in formulas for the Ricci tensor the

only one command is required to accomplish out goal `Find and Write Ricci Tensor;`. The command `ds2;` just shows the metric we are dealing with. Notice that command `Find ...` gives the *tetrad* components of the Ricci tensor R_{ab}. Thus, in addition we print coordinate components of the tensor $R_{\mu\nu}$ by the command `RIC(_j,_k);`. The hard-copy of the corresponding GRG session is presented below

```
<- Coordinates t, r, theta, phi;
<- Function a(t);
<- Frame T0=d t, T1=a*d r, T2=a*r*d theta, T3=a*r*SIN(theta)*d phi;
<- ds2;
Assuming Default Metric.
Metric calculated By default. 0.16 sec

   2          2     2     2     2  2         2              2  2  2       2
 ds  =  -  d t  + (a ) d r  + (a *r ) d theta  + (SIN(theta) *a *r ) d phi

<- Find and Write Ricci Tensor;
Sqrt det of metric calculated. 0.21 sec
Volume calculated. 0.21 sec
Vector frame calculated From frame. 0.21 sec
Inverse metric calculated From metric. 0.21 sec
Frame connection calculated. 0.38 sec
Curvature calculated. 0.49 sec
Ricci tensor calculated From curvature. 0.54 sec
Ricci tensor:

          - 3*DF(a,t,2)
RIC   = ----------------
   00          a
```

```
                                  2
         DF(a,t,2)*a + 2*DF(a,t)
RIC   = --------------------------
   11                2
                    a

                                  2
         DF(a,t,2)*a + 2*DF(a,t)
RIC   = --------------------------
   22                2
                    a

                                  2
         DF(a,t,2)*a + 2*DF(a,t)
RIC   = --------------------------
   33                2
                    a

<- RIC(_j,_k);

            - 3*DF(a,t,2)
j=0 k=0 : ----------------
                  a

                                   2
j=1 k=1 : DF(a,t,2)*a + 2*DF(a,t)

           2                          2
j=2 k=2 : r *(DF(a,t,2)*a + 2*DF(a,t) )

                    2  2                          2
j=3 k=3 : SIN(theta) *r *(DF(a,t,2)*a + 2*DF(a,t) )
```

Tracing messages demonstrate that GRG automatically applied several built-in equations to obtain required value of R_{ab}. The metric is automatically assumed to be Lorentzian $g_{ab} = \mathrm{diag}(-1, 1, 1, 1)$. First GRG computed the frame connection 1-form $\omega^a{}_b$. Next the curvature 2-form $\Omega^a{}_b$ was computed using standard equation (**??**) on page **??**. Finally the Ricci tensor was obtained using relation (**??**) on page **??**.

Second GRG task is similar to the first one:

```
Coordinates t,r,theta,phi;
Function a(t);
Metric G00=-1, G11=a^2, G22=(a*r)^2, G33=(a*r*SIN(theta))^2;
ds2;
```

```
   Find and Write Ricci Tensor;
```

The only difference is that now we work in the coordinate formalism by assigning value to the metric rather than frame. The frame is assumed to be holonomic automatically.

```
<- Coordinates t, r, theta, phi;
<- Function a(t);
<- Metric G00=-1, G11=a^2, G22=(a*r)^2, G33=(a*r*SIN(theta))^2;
<- ds2;
Assuming Default Holonomic Frame.
Frame calculated By default. 0.11 sec

   2          2     2     2     2  2         2              2  2  2       2
 ds  =  -  d t  + (a ) d r  + (a *r ) d theta  + (SIN(theta) *a *r ) d phi

<- Find and Write Ricci Tensor;
Sqrt det of metric calculated. 0.22 sec
Volume calculated. 0.22 sec
Vector frame calculated From frame. 0.22 sec
Inverse metric calculated From metric. 0.27 sec
Frame connection calculated. 0.33 sec
Curvature calculated. 0.60 sec
Ricci tensor calculated From curvature. 0.60 sec
Ricci tensor:

            - 3*DF(a,t,2)
RIC     = ----------------
    t t           a

                                  2
RIC     = DF(a,t,2)*a + 2*DF(a,t)
    r r

                   2                          2
RIC             = r *(DF(a,t,2)*a + 2*DF(a,t) )
    theta theta

                         2  2                          2
RIC         = SIN(theta) *r *(DF(a,t,2)*a + 2*DF(a,t) )
    phi phi
```

Once again GRG uses the same built-in formulas to compute the Ricci tensor but now all quantities have holonomic indices instead of tetrad ones.

Finally the third task demonstrate how GRG can be used without built-in equations. Once again we use coordinate formalism and declare two new

objects the Christoffel symbols `Chr` and Ricci tensor `Ric` (since GRG is case sensitive they are different from the built-in objects `CHR` and `RIC`). Next we use well-known equations to compute these quantities

```
Coordinates t,r,theta,phi;
Function a(t);
Metric G00=-1, G11=a^2, G22=(a*r)^2, G33=(a*r*SIN(theta))^2;
ds2;
New Chr^a_b_c with s(2,3);
Chr(j,k,l)= 1/2*GI(j,m)*(@x(k)|G(l,m)+@x(l)|G(k,m)-@x(m)|G(k,l));
New Ric_a_b with s(1,2);
Ric(j,k) = @x(n)|Chr(n,j,k) - @x(k)|Chr(n,j,n)
           + Chr(n,m,n)*Chr(m,j,k) - Chr(n,m,k)*Chr(m,n,j);
Write Ric;
```

The hard-copy of the corresponding session is

```
<- Coordinates t, r, theta, phi;
<- Function a(t);
<- Metric G00=-1, G11=a^2, G22=(a*r)^2, G33=(a*r*SIN(theta))^2;
<- ds2;
Assuming Default Holonomic Frame.
Frame calculated By default. 0.16 sec

   2          2     2     2     2  2         2              2  2  2       2
 ds  =  -  d t  + (a ) d r  + (a *r ) d theta  + (SIN(theta) *a *r ) d phi

<- New Chr^a_b_c with s(2,3);
<- Chr(j,k,l)=1/2*GI(j,m)*(@x(k)|G(l,m)+@x(l)|G(k,m)-@x(m)|G(k,l));
Inverse metric calculated From metric. 0.27 sec
<- New Ric_a_b with s(1,2);
<- Ric(j,k)=@x(n)|Chr(n,j,k)-@x(k)|Chr(n,j,n)+Chr(n,m,n)*Chr(m,j,k)
   -Chr(n,m,k)*Chr(m,n,j);
<- Write Ric;
The Ric:

            - 3*DF(a,t,2)
Ric     = ----------------
    t t           a

                                  2
Ric     = DF(a,t,2)*a + 2*DF(a,t)
    r r
```

```
                  2                          2
Ric             = r *(DF(a,t,2)*a + 2*DF(a,t) )
    theta theta

                        2  2                          2
Ric         = SIN(theta) *r *(DF(a,t,2)*a + 2*DF(a,t) )
    phi phi
```

CHAPTER 3

Formulas

This chapter describes in usual mathematical manner all GRG built-in objects and formulas. The description is extremely short since it is intended for reference only. If not stated explicitly we use lower case greek letters $_{\alpha,\beta,...}$ for holonomic (coordinate) indices; $_{a,b,c,d,m,n}$ for anholonomic frame indices and $_{i,j,k,l}$ for enumerating indices.

To establish the relationship between GRG built-in object6s and mathematical quantities we use the following notation

$$\texttt{Frame Connection omega'a.b} = \omega^a{}_b$$

This equality means that there is built-in object named `Frame Connection` having identifier `omega` which represent the frame connection 1-form $\omega^a{}_b$. If the name is omitted then we deal with *macro* object (see page **??**). The notation for indices in the left-hand side of such equalities is the same as in the `New object` declaration and is explained on page **??**.

This chapter contains not only definitions of all built-in objects but all formulas which GRG knows and can apply to find their value. If an object has several formulas for its computation when each formula is given together with the corresponding name which is printed in the typewriter font. In the case then an object has only one associated formula the way name is usually omitted.

3.1 Dimension and Signature

Let us denote the space-time dimensionality by d and n'th element of the signature specification $\mathrm{diag}_{(+1,-1,...)}$ by diag_n (n runs from 0 to $d-1$).

There are several macro objects which gives access to the dimension and signature

$$\texttt{dim} = d \tag{3.1}$$

$$\texttt{sdiag.idim} = \mathrm{diag}_i \tag{3.2}$$

$$\texttt{sgnt = sign} = \ s = \prod_{i=0}^{d-1} \mathrm{diag}_i \tag{3.3}$$

$$\texttt{mpsgn} = \ \mathrm{diag}_0 \tag{3.4}$$

$$\texttt{pmsgn} = \ -\mathrm{diag}_0 \tag{3.5}$$

The macros (two equivalent ones) which give access to coordinates

$$\texttt{X\^{}m = x\^{}m} = \ x^\mu \tag{3.6}$$

3.2 Metric, Frame and Basis

Frame θ^a and metric g_{ab} plays the fundamental role in GRG. Together they determine the space-time line element

$$ds^2 = g_{ab}\,\theta^a \otimes \theta^b = g_{\mu\nu}\,dx^\mu \otimes dx^\nu \tag{3.7}$$

The corresponding objects are

$$\texttt{Frame T'a} = \ \theta^a = h^a_\mu dx^\mu \tag{3.8}$$

$$\texttt{Metric G.a.b} = \ g_{ab} \tag{3.9}$$

and "inverse" objects are

$$\texttt{Vector Frame D.a} = \ \partial_a = h^\mu_a \partial_\mu \tag{3.10}$$

$$\texttt{Inverse Metric GI'a'b} = \ g^{ab} \tag{3.11}$$

The frame can be computed by two ways. First, `By default` frame is assumed to be holonomic

$$\theta^a = dx^\alpha \tag{3.12}$$

and `From vector frame`

$$\theta^a = |h^\mu_a|^{-1} dx^\mu \tag{3.13}$$

The vector frame can be obtained `From frame`

$$\partial_a = |h^a_\mu|^{-1} \partial_\mu \tag{3.14}$$

The metric can be computed `By default`

$$g_{ab} = \text{if } a = b \text{ then } \mathrm{diag}_a \text{ else } 0 \tag{3.15}$$

or `From inverse metric`

$$g_{ab} = |g^{ab}|^{-1} \tag{3.16}$$

The inverse metric can be computed `From metric`

$$g^{ab} = |g_{ab}|^{-1} \tag{3.17}$$

The holonomic metric $g_{\mu\nu}$ and frame h^a_μ are given by the macro objects:

$$\texttt{g\_m\_n} = \; g_{\mu\nu} \tag{3.18}$$

$$\texttt{gi\^{}m\^{}n} = \; g^{\mu\nu} \tag{3.19}$$

$$\texttt{h'a\_m} = \; h^a_\mu \tag{3.20}$$

$$\texttt{hi.a\^{}m} = \; h^\mu_a \tag{3.21}$$

The metric determinants and related densities

$$\texttt{Det of Metric detG} = \; g = \det|g_{ab}| \tag{3.22}$$

$$\texttt{Det of Holonomic Metric detg} = \; \det|g_{\mu\nu}| \tag{3.23}$$

$$\texttt{Sqrt Det of Metric sdetG} = \; \sqrt{sg} \tag{3.24}$$

The volume d-form

$$\texttt{Volume VOL} = \; \upsilon = \sqrt{sg}\,\theta^0 \wedge \ldots \wedge \theta^{d-1} = \frac{1}{d!}\mathcal{E}_{a_0\ldots a_{d-1}}\,\theta^{a_0} \wedge \ldots \wedge \theta^{a_{d-1}} \tag{3.25}$$

The so called s-forms play the role of basis in the space of the 2-forms

$$\texttt{S-forms S'a'b} = \; S^{ab} = \theta^a \wedge \theta^b \tag{3.26}$$

The basis and corresponding inverse vector basis are used when GRG works in the anholonomic mode *See page* **??**.

$$\texttt{Basis b'idim} \; = \; b^i = b^i_\mu dx^\mu \tag{3.27}$$

$$\texttt{Vector Basis e.idim} \; = \; e_i = b^\mu_i \partial_\mu \tag{3.28}$$

The basis can be computed `From frame`

$$b^i = \theta^i \tag{3.29}$$

or `From vector basis`

$$b^i = |b^\mu_i|^{-1} dx^\mu \tag{3.30}$$

The vector basis can be computed `From basis`

$$e_i = |b^i_\mu|^{-1} \partial_\mu \tag{3.31}$$

3.3 Delta and Epsilon Symbols

Macro objects for Kronecker delta symbols

$$\texttt{del\^{}m\_n} = \delta^{\mu}_{\nu} \tag{3.32}$$

$$\texttt{delh'a.b} = \delta^{a}_{b} \tag{3.33}$$

and totally antisymmetric tensors

$$\texttt{eps.a.b.c.d} = \mathcal{E}_{abcd}, \quad \mathcal{E}_{0123} = \sqrt{sg} \tag{3.34}$$

$$\texttt{epsi'a'b'c'd} = \mathcal{E}^{abcd}, \quad \mathcal{E}_{0123} = \frac{s}{\sqrt{sg}} \tag{3.35}$$

$$\texttt{epsh\_m\_n\_k\_l} = \mathcal{E}_{\mu\nu\kappa\lambda}, \quad \mathcal{E}_{0123} = \sqrt{s\det|g_{\mu\nu}|} \tag{3.36}$$

$$\texttt{epsih\^{}m\^{}n\^{}k\^{}l} = \mathcal{E}^{\mu\nu\kappa\lambda}, \quad \mathcal{E}_{0123} = \frac{s}{\sqrt{s\det|g_{\mu\nu}|}} \tag{3.37}$$

The definition for epsilon-tensors is given for dimension 4. The generalization to other dimensions is obvious.

3.4 Dualization

We use the following definition for the dualization operation. For any p-form

$$\omega_p = \frac{1}{p!}\omega_{\alpha_1\ldots\alpha_p} dx^{\alpha_1} \wedge \ldots \wedge dx^{\alpha_p} \tag{3.38}$$

the dual $(d-p)$-form is

$$*\omega_p = \frac{1}{p!(d-p)!}\mathcal{E}_{\alpha_1\ldots\alpha_{d-p}}{}^{\beta_1\ldots\beta_p}\,\omega_{\beta_1\ldots\beta_p}\, dx^{\alpha_1} \wedge \ldots \wedge dx^{\alpha_{d-p}} \tag{3.39}$$

The equivalent relation which also uniquely defines the $*$ operation is

$$*(\theta^{a_1} \wedge \ldots \wedge \theta^{a_p}) = (-1)^{p(d-p)}\partial_{a_p} \lrcorner \ldots \partial_{a_1} \lrcorner\, \upsilon \tag{3.40}$$

With such convention we have the following identities

$$** \omega_p = s(-1)^{p(d-p)}\,\omega_p \tag{3.41}$$

$$*\upsilon = s \tag{3.42}$$

$$*1 = \upsilon \tag{3.43}$$

3.5 Spinors

The notion of spinors in GRG is restricted to 4-dimensional spaces of Lorentzian signature (−,+,+,+) or (+,−,−,−) only. In this section the upper sign relates to the signature (−,+,+,+) and lower one to (+,−,−,−).

In addition to work with spinors the metric must have the following form which we call the *standard null metric*

$$g_{ab} = g^{ab} = \pm \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.44}$$

Such value of the metric can be established by the command `Null metric;`.

Therefore the line-element for spinorial formalism has the form

$$ds^2 = \pm(-\theta^0 \otimes \theta^1 - \theta^1 \otimes \theta^0 + \theta^2 \otimes \theta^3 + \theta^3 \otimes \theta^2) \tag{3.45}$$

We require also the conjugation rules for this null tetrad (frame) be

$$\overline{\theta^0} = \theta^0, \quad \overline{\theta^1} = \theta^1, \quad \overline{\theta^2} = \theta^3, \quad \overline{\theta^3} = \theta^2 \tag{3.46}$$

For such a metric and frame we fix sigma-matrices in the following form

$$\sigma_0{}^{1\dot{1}} = \sigma_1{}^{0\dot{0}} = \sigma_2{}^{1\dot{0}} = \sigma_3{}^{0\dot{1}} = 1 \tag{3.47}$$

$$\sigma^0{}_{1\dot{1}} = \sigma^1{}_{0\dot{0}} = \sigma^2{}_{1\dot{0}} = \sigma^3{}_{0\dot{1}} = \mp 1 \tag{3.48}$$

The sigma-matrices obey the rules

$$g_{mn}\sigma^m{}_{A\dot{B}}\sigma^n{}_{C\dot{D}} = \mp\epsilon_{AC}\epsilon_{\dot{B}\dot{D}} \tag{3.49}$$

$$\sigma^{aM\dot{N}}\sigma^b{}_{M\dot{N}} = \mp g^{ab} \tag{3.50}$$

The antisymmetric SL(2,C) spinor metric

$$\epsilon_{AB} = \epsilon^{AB} = \epsilon_{\dot{A}\dot{B}} = \epsilon^{\dot{A}\dot{B}} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{3.51}$$

can be used to raise and lower spinor indices

$$\varphi^A = \varphi_B\,\epsilon^{BA}, \qquad \varphi_A = \epsilon_{AB}\,\varphi^B \tag{3.52}$$

The following macro objects represent standard spinorial quantities

$$\texttt{DEL'A.B} = \delta^A_B \tag{3.53}$$

$$\texttt{EPS.A.B} = \epsilon_{AB} \tag{3.54}$$

$$\texttt{EPSI'A'B} = \epsilon^{AB} \tag{3.55}$$

$$\texttt{sigma'a.A.B~} = \sigma^a{}_{A\dot{B}} \tag{3.56}$$

$$\texttt{sigmai.a'A'B~} = \sigma_a{}^{A\dot{B}} \tag{3.57}$$

The relationship between tensors and spinors is established by the sigma-matrices

$$X^a \longleftrightarrow X^{A\dot{A}} = A^a \sigma_a{}^{A\dot{A}} \tag{3.58}$$

$$X_a \longleftrightarrow X_{A\dot{A}} = A_a \sigma^a{}_{A\dot{A}} \tag{3.59}$$

where sigma-matrices are given by Eq. (**??**) We shall denote similar equations by the sign $\longleftrightarrow$ conserving alphabetical relationship between tensor indices in the left-hand side and spinorial one in the right-hand side: $a \longleftrightarrow A\dot{A}$, $b \longleftrightarrow B\dot{B}$.

There is one quite important special case. Any real antisymmetric tensor X_{ab} are equivalent to the pair of conjugated irreducible (symmetric) spinors

$$\begin{aligned} &X_{ab} = X_{[ab]} \longleftrightarrow X_{A\dot{A}B\dot{B}} = \epsilon_{AB} X_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}} X_{AB} \\ &X_{AB} = \frac{1}{2} X_{A\dot{A}B\dot{B}} \epsilon^{\dot{A}\dot{B}}, \; X_{\dot{A}\dot{B}} = \frac{1}{2} X_{A\dot{A}B\dot{B}} \epsilon^{AB} \end{aligned} \tag{3.60}$$

The explicit form of these relations for the sigma-matrices (**??**) is

$$\begin{aligned} &X_0 = X_{13} && X_{\dot{0}} = X_{12} \\ &X_1 = -\tfrac{1}{2}(X_{01} - X_{23}) && X_{\dot{1}} = -\tfrac{1}{2}(X_{01} + X_{23}) \\ &X_2 = -X_{02} && X_{\dot{2}} = -X_{03} \end{aligned} \tag{3.61}$$

and the "inverse" relation

$$\begin{aligned} &X_{01} = -X_1 - X_{\dot{1}}, && X_{23} = X_1 - X_{\dot{1}}, \\ &X_{02} = -X_2, && X_{12} = X_{\dot{0}}, \\ &X_{03} = -X_{\dot{2}}, && X_{13} = X_0 \end{aligned} \tag{3.62}$$

We shall apply the relations (**??**) and (**??**) to various antisymmetric quantities. In particular the `Spinorial S-forms`

$$\texttt{Undotted S-forms SU.AB} = S_{AB} \tag{3.63}$$

$$\texttt{Dotted S-forms SD.AB~} = S_{\dot{A}\dot{B}} \tag{3.64}$$

The `Standard way` to compute these quantities uses relations (**??**)

$$S_{ab} = \theta_a \wedge \theta_b \longleftrightarrow \epsilon_{AB} S_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}} S_{AB} \tag{3.65}$$

Spinorial S-forms are self dual

$$*S_{AB} = iS_{AB}, \qquad *S_{\dot{A}\dot{B}} = -iS_{\dot{A}\dot{B}} \tag{3.66}$$

and exteriorly orthogonal

$$S_{AB} \wedge S_{CD} = -\frac{i}{2}v(\epsilon_{AC}\epsilon_{BD} + \epsilon_{AD}\epsilon_{BC}), \quad S_{AB} \wedge S_{\dot{C}\dot{D}} = 0 \tag{3.67}$$

There is one subtle pint concerning tensor quantities in the spinorial formalism. Since spinorial null tetrad is complex with the conjugation rule $\overline{\theta^2} = \theta^3$ all tensor quantities represented in this frame also becomes complex with similar conjugation rules for any tensor index. There is special macro object `cci` which performs such "index conjugation": `cci0=0`, `cci(1)=1`, `cci2=3`, `cci(3)=2`. Therefore the correct expression for the $\overline{\theta^a}$ is `~T(cci(a))` but not `~T(a)`.

3.6 Connection, Torsion and Nonmetricity

Covariant derivatives and differentials for quantities having frame and coordinate indices are

$$DX^a{}_b = dX^a{}_b + \omega^a{}_m \wedge X^m{}_b - \omega^m{}_b \wedge X^a{}_m \tag{3.68}$$

$$DX^\mu{}_\nu = dX^\mu{}_\nu + \Gamma^\mu{}_\pi \wedge X^\pi{}_\nu - \Gamma^\pi{}_\nu \wedge X^\mu{}_\pi \tag{3.69}$$

The corresponding built-in connection 1-forms are

$$\texttt{Frame Connection omega'a.b} = \ \omega^a{}_b = \omega^a{}_{b\mu}dx^\mu \tag{3.70}$$

$$\texttt{Holonomic Connection GAMMA\^{}m\_n} = \ \Gamma^\mu{}_\nu = \Gamma^\mu{}_{\nu\pi}dx^\pi \tag{3.71}$$

Frame connection can be computed `From holonomic connection`

$$\omega^a{}_b = \Gamma^a{}_b + dh^\mu_b\, h^a_\mu \tag{3.72}$$

and inversely holonomic connection can be obtained `From frame connection`

$$\Gamma^\mu{}_\nu = \omega^\mu{}_\nu + dh^b_\nu\, h^\mu_b \tag{3.73}$$

By default these connections are Riemannian (i.e. symmetric and metric compatible). To work with nonsymmetric connection with torsion the switch `TORSION` must be turned on. Then the torsion 2-form is

$$\texttt{Torsion THETA'a} = \ \Theta^a = \frac{1}{2}Q^a{}_{pq}S^{pq}, \quad Q^a{}_{bc} = \Gamma^a{}_{bc} - \Gamma^a_{cb} \tag{3.74}$$

Finally to work with non metric-compatible spaces with nonmetricity the switch NONMETR must be turned on. The nonmetricity 1-form is

$$\texttt{Nonmetricity N.a.b} = \; N_{ab} = N_{ab\mu}dx^{\mu}, \quad N_{ab\mu} = -\nabla_{\mu} g_{ab} \tag{3.75}$$

In general any torsion or nonmetricity related object is defined iff the corresponding switch is on.

If either TORSION or NONMETR is on then Riemannian versions of the connection 1-forms are available as well

$$\texttt{Riemann Frame Connection romega'a.b} = \overset{\{\}}{\omega}{}^{a}{}_{b} \tag{3.76}$$

$$\texttt{Riemann Holonomic Connection RGAMMA\^{}m\_n} = \overset{\{\}}{\Gamma}{}^{\mu}{}_{\nu} \tag{3.77}$$

The Riemann holonomic connection can be obtained From Riemann frame connection

$$\overset{\{\}}{\Gamma}{}^{\mu}{}_{\nu} = \overset{\{\}}{\omega}{}^{\mu}{}_{\nu} + dh^{b}_{\nu}\, h^{\mu}_{b} \tag{3.78}$$

If torsion is nonzero but nonmetricity vanishes (TORSION is on, NONMETR is off) then the difference between the connection and Riemann connection is called the contorsion 1-form

$$\texttt{Contorsion KQ'a.b} = \overset{Q}{K}{}^{a}{}_{b} = \overset{Q}{K}{}^{a}{}_{b\mu}dx^{\mu} = \Gamma^{a}{}_{b} - \overset{\{\}}{\Gamma}{}^{a}{}_{b} \tag{3.79}$$

If nonmetricity is nonzero but torsion vanishes (TORSION is off, NONMETR is on) then the difference between the connection and Riemann connection is called the nonmetricity defect

$$\texttt{Nonmetricity Defect KN'a.b} = \overset{N}{K}{}^{a}{}_{b} = \overset{N}{K}{}^{a}{}_{b\mu}dx^{\mu} = \Gamma^{a}{}_{b} - \overset{\{\}}{\Gamma}{}^{a}{}_{b} \tag{3.80}$$

Finally if both torsion and nonmetricity are nonzero (TORSION and NONMETR are on) then we

$$\texttt{Connection Defect K'a.b} = \; K^{a}{}_{b} = K^{a}{}_{b\mu}dx^{\mu} = \Gamma^{a}{}_{b} - \overset{\{\}}{\Gamma}{}^{a}{}_{b} \tag{3.81}$$

$$K^{a}{}_{b} = \overset{Q}{K}{}^{a}{}_{b} + \overset{N}{K}{}^{a}{}_{b} \tag{3.82}$$

For the sake of convenience we introduce also macro objects which compute the usual Christoffel symbols

$$\texttt{CHR\^{}m\_n\_p} \;\; = \;\; \{^{\mu}_{\nu\pi}\} = \frac{1}{2} g^{\mu\tau}(\partial_{\pi} g_{\nu\tau} + \partial_{\nu} g_{\pi\tau} - \partial_{\tau} g_{\nu\pi}) \tag{3.83}$$

$$\texttt{CHRF\_m\_n\_p} \;=\; [_{\mu},{}_{\nu\pi}] = \frac{1}{2}(\partial_\pi g_{\nu\mu} + \partial_\nu g_{\pi\mu} - \partial_\mu g_{\nu\pi}) \tag{3.84}$$

$$\texttt{CHRT\_m} \;=\; \{^{\pi}_{\pi\mu}\} = \frac{1}{2\det|g_{\alpha\beta}|}\partial_\mu\left(\det|g_{\alpha\beta}|\right) \tag{3.85}$$

The connection, frame, metric, torsion and nonmetricity are related to each other by the so called structural equations which in the most general case read

$$\begin{aligned} D\theta^a + \Theta^a &= 0 \\ Dg_{ab} + N_{ab} &= 0 \end{aligned} \tag{3.86}$$

or in the equivalent "explicit" form

$$\begin{aligned} \omega^a{}_b \wedge \theta^b &= -t^a, \qquad & t^a &= d\theta^a + \Theta^a, \\ \omega_{ab} + \omega_{ba} &= n_{ab}, \qquad & n_{ab} &= dg_{ab} + N_{ab} \end{aligned} \tag{3.87}$$

The solution to equations (**??**) are given by the relation

$$\omega^a{}_b = \frac{1}{2}\left[-\partial^a \lrcorner t_b + \partial_b \lrcorner t^a + n^a{}_b + \left(\partial^a \lrcorner (\partial_b \lrcorner t_c - n_{bc}) + \partial_b \lrcorner n^a{}_c\right)\theta^c\right] \tag{3.88}$$

For various specific values of n_{ab} and t^a equations (**??**) and (**??**) can be used for different purposes.

In the most general case (**??**) is the `Standard way` to compute connection 1-form $\omega^a{}_b$. The torsion and nonmetricity are included in these equations depending on the switches `TORSION` and `NONMETR`.

The same equation (**??**) with $n_{ab} = dg_{ab}$ and $t^a = d\theta^a$ is the `Standard way` to find Riemann frame connection $\overset{\{\}}{\omega}{}^a{}_b$.

If torsion is nonzero then $\omega^a{}_b$ can be computed `From contorsion`

$$\omega^a{}_b = \overset{\{\}}{\omega}{}^a{}_b + \overset{Q}{K}{}^a{}_b \tag{3.89}$$

where $\overset{\{\}}{\omega}{}^a{}_b$ is given by Eq. (**??**).

Similarly if nonmetricity is nonzero then $\omega^a{}_b$ can be computed `From nonmetricity defect`

$$\omega^a{}_b = \overset{\{\}}{\omega}{}^a{}_b + \overset{N}{K}{}^a{}_b \tag{3.90}$$

where $\overset{\{\}}{\omega}{}^a{}_b$ is given by Eq. (**??**).

Finally if both torsion and nonmetricity are nonzero then $\omega^a{}_b$ can be computed `From connection defect`

$$\omega^a{}_b = \overset{\{\}}{\omega}{}^a{}_b + K^a{}_b \tag{3.91}$$

where $\overset{\{\}}{\omega}{}^a{}_b$ is given by Eq. (??).

The Riemannian part of connection in Eqs. (??), (??), (??) are directly computed by Eq. (??) (not via the object `romega`).

The contorsion $\overset{Q}{K}{}^a{}_b$ is obtained `From torsion` by (??) with $t^a = \Theta^a$, $n_{ab} = 0$.

The nonmetricity defect $\overset{N}{K}{}^a{}_b$ is obtained `From nonmetricity` by (??) with $t^a = 0$, $n_{ab} = N_{ab}$.

Analogously (??) with $t^a = \Theta^a$, $n_{ab} = N_{ab}$ is the `Standard way` to compute the connection defect $K^a{}_b$.

The torsion Θ^a can be calculated `From contorsion`

$$\Theta^a = -\overset{Q}{K}{}^a{}_b \wedge \theta^b \tag{3.92}$$

or `From connection defect`

$$\Theta^a = -K^a{}_b \wedge \theta^b \tag{3.93}$$

The nonmetricity N_{ab} can be computed `From nonmetricity defect`

$$N_{ab} = \overset{N}{K}_{ab} + \overset{N}{K}_{ba} \tag{3.94}$$

or `From connection defect`

$$N_{ab} = K_{ab} + K_{ba} \tag{3.95}$$

3.7 Spinorial Connection and Torsion

Spinorial connection is defined in GRG iff nonmetricity is zero and switch `NONMETR` is turned off. The upper sign in this section correspond to the signature (−,+,+,+) while lower one to the signature (+,−,−,−).

Spinorial connection is defined by the equation

$$DX^A_{\dot B} = dX^A{}_{\dot B} \mp \omega^A{}_M \, X^M{}_{\dot B} \pm \omega^{\dot M}{}_{\dot B} \, X^A{}_{\dot M} \tag{3.96}$$

where due to antisymmetry of the frame connection $\omega_{ab} = \omega_{[ab]}$ we have `Spinorial connection` 1-forms

$$\omega_{ab} \longleftrightarrow \epsilon_{AB}\omega_{\dot A\dot B} + \epsilon_{\dot A\dot B}\omega_{AB} \tag{3.97}$$

$$\texttt{Undotted Connection omegau.AB} = \omega_{AB} \tag{3.98}$$

$$\texttt{Dotted Connection omegad.AB}\tilde{} = \omega_{\dot A\dot B} \tag{3.99}$$

The spinorial connection 1-forms ω_{AB} and $\omega_{\dot{A}\dot{B}}$ can be calculated `From frame connection` by the standard spinor $\longleftrightarrow$ tensor relation (**??**).

Inversely the frame connection ω_{ab} can be found `From spinorial connection` by relation (**??**).

Since ω_{ab} is real the spinorial equivalents ω_{AB} and $\omega_{\dot{A}\dot{B}}$ can be computed from each other `By conjugation`

$$\omega_{\dot{A}\dot{B}} = \overline{\omega_{AB}}, \qquad \omega_{AB} = \overline{\omega_{\dot{A}\dot{B}}} \tag{3.100}$$

If torsion is nonzero (`TORSION` is on) when we have in addition the `Riemann spinorial connection`

$$\texttt{Riemann Undotted Connection romegau.AB} = \overset{\{\}}{\omega}_{AB} \tag{3.101}$$

$$\texttt{Riemann Dotted Connection romegad.AB}\tilde{} = \overset{\{\}}{\omega}_{\dot{A}\dot{B}} \tag{3.102}$$

The Riemann spinorial connection $\overset{\{\}}{\omega}_{AB}$ can be calculated by `Standard way`

$$\overset{\{\}}{\omega}_{AB} = \pm i * [dS_{AB} \wedge \theta_{C\dot{C}} - \epsilon_{C(A} dS_{B)M} \wedge \theta^{M}{}_{\dot{C}}]\theta^{C\dot{C}} \tag{3.103}$$

The conjugated relation is used for $\overset{\{\}}{\omega}_{\dot{A}\dot{B}}$.

The `Spinorial contorsion` 1-forms

$$\texttt{Undotted Contorsion KU.AB} = \overset{Q}{K}_{AB} \tag{3.104}$$

$$\texttt{Dotted Contorsion KD.AB}\tilde{} = \overset{Q}{K}_{\dot{A}\dot{B}} \tag{3.105}$$

are the spinorial analogues of the contorsion 1-form

$$\overset{Q}{K}_{ab} \longleftrightarrow \epsilon_{AB}\, \overset{Q}{K}_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}}\, \overset{Q}{K}_{AB} \tag{3.106}$$

The spinorial contorsion 1-forms $\overset{Q}{K}_{AB}$ and $\overset{Q}{K}_{\dot{A}\dot{B}}$ can be calculated `From contorsion` by the standard spinor $\longleftrightarrow$ tensor relation (**??**).

Inversely the contorsion $\overset{Q}{K}_{ab}$ can be found `From spinorial contorsion` by relation (**??**).

The spinorial equivalents $\overset{Q}{K}_{AB}$ and $\overset{Q}{K}_{\dot{A}\dot{B}}$ can be computed from each other `By conjugation`

$$\overset{Q}{K}_{\dot{A}\dot{B}} = \overline{\overset{Q}{K}_{AB}}, \qquad \overset{Q}{K}_{AB} = \overline{\overset{Q}{K}_{\dot{A}\dot{B}}} \tag{3.107}$$

The `Standard way` to find ω_{AB} is

$$\omega_{AB} = \overset{\{\}}{\omega}_{AB} + \overset{Q}{K}_{AB} \tag{3.108}$$

where $\overset{\{\}}{\omega}_{AB}$ is given directly by Eq. (**??**). The conjugated Eq. is used for $\omega_{\dot{A}\dot{B}}$.

3.8 Curvature

The curvature 2-form

$$\texttt{Curvature OMEGA'a.b} = \ \Omega^a{}_b = \frac{1}{2} R^a_{bcd}\, S^{cd} \tag{3.109}$$

can be computed `By standard way`

$$\Omega^a{}_b = d\omega^a{}_b + \omega^a{}_n \wedge \omega^n{}_b \tag{3.110}$$

The Riemann curvature tensor is given by the relation

$$\texttt{Riemann Tensor RIM'a.b.c.d} = \ R^a{}_{bcd} = \partial_d \lrcorner \partial_c \lrcorner \Omega^a{}_b \tag{3.111}$$

The Ricci tensor

$$\texttt{Ricci Tensor RIC.a.b} = \ R_{ab} \tag{3.112}$$

can be computed `From Curvature`

$$R_{ab} = \partial_b \lrcorner \partial_m \lrcorner \Omega^m{}_a \tag{3.113}$$

or `From Riemann tensor`

$$R_{ab} = R^m{}_{amb} \tag{3.114}$$

The

$$\texttt{Scalar Curvature RR} = \ R \tag{3.115}$$

can be computed `From Ricci Tensor`

$$R = R_{mn}\, g^{mn} \tag{3.116}$$

The Einstein tensor is given by the relation

$$\texttt{Einstein Tensor GT.a.b} = \ G_{ab} = R_{ab} - \frac{1}{2} g_{ab} R \tag{3.117}$$

If nonmetricity is nonzero (`NONMETR` is on) then we have

$$\texttt{Homothetic Curvature OMEGAH} = \overset{\text{h}}{\Omega} \tag{3.118}$$

$$\texttt{A-Ricci Tensor RICA.a.b} = \overset{\text{A}}{R}_{ab} \tag{3.119}$$

$$\texttt{S-Ricci Tensor RICS.a.b} = \overset{\text{S}}{R}_{ab} \tag{3.120}$$

They can be calculated `From curvature` by the relations

$$\overset{\text{h}}{\Omega} = \Omega^{n}{}_{n} \tag{3.121}$$

$$\overset{\text{A}}{R}_{ab} = \partial_b \lrcorner \partial^m \lrcorner \Omega_{[ma]} \tag{3.122}$$

$$\overset{\text{S}}{R}_{ab} = \partial_b \lrcorner \partial^m \lrcorner \Omega_{(ma)} \tag{3.123}$$

and the scalar curvature can be computed `From A-Ricci tensor`

$$R = \overset{\text{A}}{R}_{mn} g^{mn} \tag{3.124}$$

3.9 Spinorial Curvature

Spinorial curvature is defined in GRG iff nonmetricity is zero and switch `NONMETR` is turned off. The upper sign in this section correspond to the signature (−,+,+,+) while lower one to the signature (+,−,−,−).

The `Spinorial curvature` 2-forms

$$\texttt{Undotted Curvature OMEGAU.AB} = \Omega_{AB} \tag{3.125}$$

$$\texttt{Dotted Curvature OMEGAD.AB}\tilde{} = \Omega_{\dot{A}\dot{B}} \tag{3.126}$$

is related to the curvature 2-form Ω_{ab} by the standard relation

$$\Omega_{ab} \longleftrightarrow \epsilon_{AB}\Omega_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}}\Omega_{AB} \tag{3.127}$$

The spinorial curvature 1-forms Ω_{AB} and $\Omega_{\dot{A}\dot{B}}$ can be calculated `From curvature` by the relation (**??**).

The frame curvature Ω_{ab} can be found `From spinorial curvature` by relation (**??**).

The Ω_{AB} and $\Omega_{\dot{A}\dot{B}}$ can be computed from each other `By conjugation`

$$\Omega_{\dot{A}\dot{B}} = \overline{\Omega_{AB}}, \qquad \Omega_{AB} = \overline{\Omega_{\dot{A}\dot{B}}} \tag{3.128}$$

The `Standard way` to calculate Ω_{AB} is

$$\Omega_{AB} = d\omega_{AB} \pm \omega_A{}^M \wedge \omega_{MB} \tag{3.129}$$

The conjugated relation is used for $\Omega_{\dot{A}\dot{B}}$.

3.10 Curvature Decomposition

In general curvature consists of 11 irreducible pieces. If nonmetricity is nonzero then one can perform decomposition

$$R_{abcd} = \overset{\mathrm{A}}{R}_{abcd} + \overset{\mathrm{S}}{R}_{abcd}, \qquad \overset{\mathrm{A}}{R}_{abcd} = R_{[ab]cd}, \qquad \overset{\mathrm{S}}{R}_{abcd} = R_{(ab)cd} \tag{3.130}$$

Here the S-part of the curvature vanishes identically if nonmetricity is zero and we consider further decomposition of A and S parts independently.

First we consider the A-part of the curvature. It can be decomposed into 6 pieces

$$\overset{\mathrm{A}}{R}_{abcd} = \overset{\mathrm{w}}{R}_{abcd} + \overset{\mathrm{c}}{R}_{abcd} + \overset{\mathrm{r}}{R}_{abcd} + \overset{\mathrm{a}}{R}_{abcd} + \overset{\mathrm{b}}{R}_{abcd} + \overset{\mathrm{d}}{R}_{abcd} \tag{3.131}$$

Here first three terms are the well-known irreducible pieces of the Riemannian curvature while last three terms vanish if torsion is zero. The corresponding 2-forms are

$$\texttt{Weyl 2-form OMW.a.b} = \overset{\mathrm{w}}{\Omega}_{ab} = \frac{1}{2}\,\overset{\mathrm{w}}{R}_{abcd}\,S^{cd} \tag{3.132}$$

$$\texttt{Traceless Ricci 2-form OMC.a.b} = \overset{\mathrm{c}}{\Omega}_{ab} = \frac{1}{2}\,\overset{\mathrm{c}}{R}_{abcd}\,S^{cd} \tag{3.133}$$

$$\texttt{Scalar Curvature 2-form OMR.a.b} = \overset{\mathrm{r}}{\Omega}_{ab} = \frac{1}{2}\,\overset{\mathrm{r}}{R}_{abcd}\,S^{cd} \tag{3.134}$$

$$\texttt{Ricanti 2-form OMA.a.b} = \overset{\mathrm{a}}{\Omega}_{ab} = \frac{1}{2}\,\overset{\mathrm{a}}{R}_{abcd}\,S^{cd} \tag{3.135}$$

$$\texttt{Traceless Deviation 2-form OMB.a.b} = \overset{\mathrm{b}}{\Omega}_{ab} = \frac{1}{2}\,\overset{\mathrm{b}}{R}_{abcd}\,S^{cd} \tag{3.136}$$

$$\texttt{Antisymmetric Curvature 2-form OMD.a.b} = \overset{\mathrm{d}}{\Omega}_{ab} = \frac{1}{2}\,\overset{\mathrm{d}}{R}_{abcd}\,S^{cd} \tag{3.137}$$

The `Standard way` to find these quantities is given by the following formulas.

$$\overset{\mathrm{r}}{\Omega}_{ab} = \frac{1}{d(d-1)} R\,S_{ab} \tag{3.138}$$

$$\overset{\mathrm{c}}{\Omega}_{ab} = \frac{1}{(d-2)}\left[C_{am}\,\theta^m \wedge \theta_b - C_{bm}\,\theta^m \wedge \theta_a\right], \quad C_{ab} = \overset{\mathrm{A}}{R}_{(ab)} - \frac{1}{d} g_{ab} R \tag{3.139}$$

$$\overset{\mathrm{a}}{\Omega}_{ab} = \frac{1}{(d-2)}\left[A_{am}\,\theta^m \wedge \theta_b - A_{bm}\,\theta^m \wedge \theta_a\right], \quad A_{ab} = \overset{\mathrm{A}}{R}_{[ab]} \tag{3.140}$$

$$\overset{\mathrm{d}}{\Omega}_{ab} = \frac{1}{12}\partial_b \lrcorner \partial_a \lrcorner (\overset{\mathrm{A}}{\Omega}_{mn} \wedge \theta^m \wedge \theta^n) \tag{3.141}$$

$$\overset{\mathrm{b}}{\Omega}{}_{ab} = \frac{1}{2}\left[\partial_b \lrcorner (\theta^m \wedge \overset{\mathrm{A0}}{\Omega}{}_{am}) - \partial_a \lrcorner (\theta^m \wedge \overset{\mathrm{A0}}{\Omega}{}_{bm})\right] \tag{3.142}$$

where

$$\overset{\mathrm{A0}}{\Omega}{}_{ab} = \overset{\mathrm{A}}{\Omega}{}_{ab} - \overset{\mathrm{c}}{\Omega}{}_{ab} - \overset{\mathrm{r}}{\Omega}{}_{ab} - \overset{\mathrm{a}}{\Omega}{}_{ab} - \overset{\mathrm{d}}{\Omega}{}_{ab}$$

And finally

$$\overset{\mathrm{w}}{\Omega}{}_{ab} = \overset{\mathrm{A}}{\Omega}{}_{ab} - \overset{\mathrm{c}}{\Omega}{}_{ab} - \overset{\mathrm{r}}{\Omega}{}_{ab} - \overset{\mathrm{a}}{\Omega}{}_{ab} - \overset{\mathrm{b}}{\Omega}{}_{ab} - \overset{\mathrm{d}}{\Omega}{}_{ab} \tag{3.143}$$

If $d = 2$ then $\overset{\mathrm{A}}{\Omega}{}_{ab}$ turns out to be irreducible and coincides with the scalar curvature irreducible piece

$$\overset{\mathrm{A}}{\Omega}{}_{ab} = \overset{\mathrm{r}}{\Omega}{}_{ab} \tag{3.144}$$

Now we consider the decomposition of the S curvature part which is nonzero iff nonmetricity is nonzero. First we consider the case $d \geq 3$. In this case we have 5 irreducible components

$$\overset{\mathrm{S}}{R}{}_{abcd} = \overset{\mathrm{h}}{R}{}_{abcd} + \overset{\mathrm{sc}}{R}{}_{abcd} + \overset{\mathrm{sa}}{R}{}_{abcd} + \overset{\mathrm{v}}{R}{}_{abcd} + \overset{\mathrm{u}}{R}{}_{abcd} \tag{3.145}$$

The corresponding 2-forms are

$$\texttt{Homothetic Curvature 2-form OSH.a.b} = \overset{\mathrm{h}}{\Omega}{}_{ab} = \frac{1}{2}\overset{\mathrm{h}}{R}{}_{abcd}\, S^{cd} \tag{3.146}$$

$$\texttt{Antisymmetric S-Ricci 2-form OSA.a.b} = \overset{\mathrm{sa}}{\Omega}{}_{ab} = \frac{1}{2}\overset{\mathrm{sa}}{R}{}_{abcd}\, S^{cd} \tag{3.147}$$

$$\texttt{Traceless S-Ricci 2-form OSC.a.b} = \overset{\mathrm{sc}}{\Omega}{}_{ab} = \frac{1}{2}\overset{\mathrm{sc}}{R}{}_{abcd}\, S^{cd} \tag{3.148}$$

$$\texttt{Antisymmetric S-Curvature 2-form OSV.a.b} = \overset{\mathrm{v}}{\Omega}{}_{ab} = \frac{1}{2}\overset{\mathrm{v}}{R}{}_{abcd}\, S^{cd} \tag{3.149}$$

$$\texttt{Symmetric S-Curvature 2-form OSU.a.b} = \overset{\mathrm{u}}{\Omega}{}_{ab} = \frac{1}{2}\overset{\mathrm{u}}{R}{}_{abcd}\, S^{cd} \tag{3.150}$$

The `Standard way` to compute the decomposition is

$$\overset{\mathrm{h}}{\Omega}{}_{ab} = -\frac{1}{(d^2-4)}\left[\theta_a \wedge \partial_b \lrcorner \overset{\mathrm{h}}{\Omega} + \theta_b \wedge \partial_a \lrcorner \overset{\mathrm{h}}{\Omega} - g_{ab}\overset{\mathrm{h}}{\Omega} d\right] \tag{3.151}$$

$$\overset{\mathrm{sa}}{\Omega}{}_{ab} = \frac{d}{(d^2-4)}\left[\theta_a \wedge (\overset{\mathrm{S}}{R}{}_{[bm]} \wedge \theta^m) + \theta_b \wedge (\overset{\mathrm{S}}{R}{}_{[am]} \wedge \theta^m) - \frac{2}{d} g_{ab}\ \overset{\mathrm{S}}{R}{}_{cd} S^{cd}\right] \tag{3.152}$$

$$\overset{\mathrm{sc}}{\Omega}{}_{ab} = \frac{1}{d}\left[\theta_a \wedge (\overset{\mathrm{S}}{R}{}_{(bm)} \wedge \theta^m) + \theta_b \wedge (\overset{\mathrm{S}}{R}{}_{(am)} \wedge \theta^m)\right] \tag{3.153}$$

$$\overset{\text{v}}{\Omega}{}_{ab} = \frac{1}{4}\left[\partial_a \lrcorner (\overset{\text{S0}}{\Omega}{}_{bm} \wedge \theta^m) + \partial_b \lrcorner (\overset{\text{S0}}{\Omega}{}_{am} \wedge \theta^m)\right] \tag{3.154}$$

where

$$\overset{\text{S0}}{\Omega}{}_{ab} = \overset{\text{S}}{\Omega}{}_{ab} - \overset{\text{h}}{\Omega}{}_{ab} - \overset{\text{sa}}{\Omega}{}_{ab} - \overset{\text{sc}}{\Omega}{}_{ab}$$

And finally

$$\overset{\text{u}}{\Omega}{}_{ab} = \overset{\text{S}}{\Omega}{}_{ab} - \overset{\text{h}}{\Omega}{}_{ab} - \overset{\text{sa}}{\Omega}{}_{ab} - \overset{\text{sc}}{\Omega}{}_{ab} - \overset{\text{v}}{\Omega}{}_{ab} \tag{3.155}$$

If $d = 2$ then only the h- and sc-components are nonzero. The $\overset{\text{sc}}{\Omega}{}_{ab}$ are given by (**??**) and

$$\overset{\text{h}}{\Omega}{}_{ab} = \overset{\text{S}}{\Omega}{}_{ab} - \overset{\text{sc}}{\Omega}{}_{ab} \tag{3.156}$$

object	exists if	and has n components
R_{abcd}		$\frac{d^3(d-1)}{2}$
$\overset{\{\}}{R}{}_{abcd}$		$\frac{d^2(d^2-1)}{12}$
$\overset{\text{A}}{R}{}_{abcd}$		$\frac{d^2(d-1)^2}{4}$
$\overset{\text{S}}{R}{}_{abcd}$		$\frac{d^2(d^2-1)}{4}$
$\overset{\text{w}}{R}{}_{abcd}$	$d \geq 4$	$\frac{d(d+1)(d+2)(d-3)}{12}$
$\overset{\text{c}}{R}{}_{abcd}$	$d \geq 3$	$\frac{(d+2)(d-1)}{2}$
$\overset{\text{r}}{R}{}_{abcd}$		1
$\overset{\text{a}}{R}{}_{abcd}$	$d \geq 3$	$\frac{d(d-1)}{2}$
$\overset{\text{b}}{R}{}_{abcd}$	$d \geq 4$	$\frac{d(d-1)(d+2)(d-3)}{8}$
$\overset{\text{d}}{R}{}_{abcd}$	$d \geq 4$	$\frac{d(d-1)(d-2)(d-3)}{24}$
$\overset{\text{h}}{R}{}_{abcd}$		$\frac{d(d-1)}{2}$
$\overset{\text{sa}}{R}{}_{abcd}$	$d \geq 3$	$\frac{d(d-1)}{2}$
$\overset{\text{sc}}{R}{}_{abcd}$		$\frac{(d+2)(d-1)}{2}$
$\overset{\text{v}}{R}{}_{abcd}$	$d \geq 4$	$\frac{d(d+2)(d-1)(d-3)}{8}$
$\overset{\text{u}}{R}{}_{abcd}$	$d \geq 3$	$\frac{(d-2)(d+4)(d^2-1)}{8}$

3.11 Spinorial Curvature Decomposition

Spinorial curvature is defined in GRG iff nonmetricity is zero and switch `NONMETR` is turned off. The upper sign in this section correspond to the signa-

ture $(-,+,+,+)$ while lower one to the signature $(+,-,-,-)$.

Let us introduce the spinorial analog of the curvature tensor

$$\begin{aligned} R_{abcd} \longleftrightarrow \quad & R_{ABCD}\epsilon_{\dot{A}\dot{B}}\epsilon_{\dot{C}\dot{D}} + R_{\dot{A}\dot{B}\dot{C}\dot{D}}\epsilon_{AB}\epsilon_{CD} \\ & + R_{AB\dot{C}\dot{D}}\epsilon_{\dot{A}\dot{B}}\epsilon_{CD} + R_{\dot{A}\dot{B}CD}\epsilon_{AB}\epsilon_{\dot{C}\dot{D}}, \end{aligned} \tag{3.157}$$

$$R_{ABCD} = -i*(\Omega_{AB}\wedge S_{CD}), \quad R_{AB\dot{C}\dot{D}} = i*(\Omega_{AB}\wedge S_{\dot{C}\dot{D}}) \tag{3.158}$$

$$R_{\dot{A}\dot{B}\dot{C}\dot{D}} = \overline{R_{ABCD}}, \quad R_{\dot{A}\dot{B}CD} = \overline{R_{AB\dot{C}\dot{D}}} \tag{3.159}$$

The quantities R_{ABCD} and $R_{AB\dot{C}\dot{D}}$ can be used to compute the `Curvature spinors` ($\equiv$ `Curvature components`)

$$\texttt{Weyl Spinor RW.ABCD} = C_{ABCD} \tag{3.160}$$

$$\texttt{Traceless Ricci Spinor RC.AB.CD}\tilde{} = C_{AB\dot{C}\dot{D}} \tag{3.161}$$

$$\texttt{Scalar Curvature RR} = R \tag{3.162}$$

$$\texttt{Ricanti Spinor RA.AB} = A_{AB} \tag{3.163}$$

$$\texttt{Traceless Deviation Spinor RB.AB.CD}\tilde{} = B_{AB\dot{C}\dot{D}} \tag{3.164}$$

$$\texttt{Scalar Deviation RD} = D \tag{3.165}$$

All these spinors are irreducible (totally symmetric).

Weyl spinor C_{ABCD} `From spinor curvature` is

$$C_{abcd} \longleftrightarrow C_{ABCD}\epsilon_{\dot{A}\dot{B}}\epsilon_{\dot{C}\dot{D}} + C_{\dot{A}\dot{B}\dot{C}\dot{D}}\epsilon_{AB}\epsilon_{CD} \tag{3.166}$$

$$C_{ABCD} = R_{(ABCD)} \tag{3.167}$$

Traceless Ricci spinor $C_{AB\dot{A}\dot{B}}$ `From spinor curvature` is

$$C_{ab} \longleftrightarrow C_{AB\dot{A}\dot{B}} \tag{3.168}$$

$$C_{AB\dot{C}\dot{D}} = \pm(R_{AB\dot{C}\dot{D}} + R_{\dot{C}\dot{D}AB}) \tag{3.169}$$

Scalar curvature `From spinor curvature` is

$$R = 2(R^{MN}{}_{MN} + R^{\dot{M}\dot{N}}{}_{\dot{M}\dot{N}}) \tag{3.170}$$

Antisymmetric Ricci spinor A_{AB} `From spinor curvature` is

$$A_{ab} \longleftrightarrow A_{AB}\epsilon_{\dot{A}\dot{B}} + A_{\dot{A}\dot{B}}\epsilon_{AB} \tag{3.171}$$

$$A_{AB} = \mp R_{(A|}{}^{M}{}_{M|B)} \tag{3.172}$$

Traceless deviation spinor $B_{AB\dot{A}\dot{B}}$ `From spinor curvature` is

$$B_{ab} \longleftrightarrow B_{AB\dot{A}\dot{B}} \tag{3.173}$$

$$B_{AB\dot{C}\dot{D}} = \pm i(R_{AB\dot{C}\dot{D}} - R_{\dot{C}\dot{D}AB}) \tag{3.174}$$

Deviation trace `From spinor curvature` is

$$D = -2i(R^{MN}{}_{MN} - R^{\dot{M}\dot{N}}{}_{\dot{M}\dot{N}}) \tag{3.175}$$

Note that spinors $C_{AB\dot{A}\dot{B}}, B_{AB\dot{A}\dot{B}}$ are Hermitian

$$C_{AB\dot{C}\dot{D}} = \overline{C_{CD\dot{A}\dot{B}}}, \quad B_{AB\dot{C}\dot{D}} = \overline{B_{CD\dot{A}\dot{B}}} \tag{3.176}$$

Finally we introduce the decomposition for the spinorial curvature 2-form

$$\Omega_{AB} = \overset{\text{w}}{\Omega}{}_{AB} + \overset{\text{c}}{\Omega}{}_{AB} + \overset{\text{r}}{\Omega}{}_{AB} + \overset{\text{a}}{\Omega}{}_{AB} + \overset{\text{b}}{\Omega}{}_{AB} + \overset{\text{c}}{\Omega}{}_{AB} \tag{3.177}$$

where the `Undotted curvature 2-forms`

$$\texttt{Undotted Weyl 2-form OMWU.AB} = \overset{\text{w}}{\Omega}{}_{AB} \tag{3.178}$$

$$\texttt{Undotted Traceless Ricci 2-form OMCU.AB} = \overset{\text{c}}{\Omega}{}_{AB} \tag{3.179}$$

$$\texttt{Undotted Scalar Curvature 2-form OMRU.AB} = \overset{\text{r}}{\Omega}{}_{AB} \tag{3.180}$$

$$\texttt{Undotted Ricanti 2-form OMAU.AB} = \overset{\text{a}}{\Omega}{}_{AB} \tag{3.181}$$

$$\texttt{Undotted Traceless Deviation 2-form OMBU.AB} = \overset{\text{b}}{\Omega}{}_{AB} \tag{3.182}$$

$$\texttt{Undotted Scalar Deviation 2-form OMDU.AB} = \overset{\text{d}}{\Omega}{}_{AB} \tag{3.183}$$

are given by

$$\overset{\text{w}}{\Omega}{}_{AB} = C_{ABCD}S^{CD} \tag{3.184}$$

$$\overset{\text{c}}{\Omega}{}_{AB} = \pm\frac{1}{2}C_{AB\dot{C}\dot{D}}S^{\dot{C}\dot{D}} \tag{3.185}$$

$$\overset{\text{r}}{\Omega}{}_{AB} = \frac{1}{12}S_{AB}R \tag{3.186}$$

$$\overset{\text{a}}{\Omega}{}_{AB} = \pm A_{(A}{}^{M}S_{M|B)} \tag{3.187}$$

$$\overset{\text{b}}{\Omega}{}_{AB} = \mp\frac{i}{2}B_{AB\dot{C}\dot{D}}S^{\dot{C}\dot{D}} \tag{3.188}$$

$$\overset{\text{d}}{\Omega}{}_{AB} = \frac{i}{12}S_{AB}D \tag{3.189}$$

3.12 Torsion Decomposition

The torsion tensor

$$Q_{abc} = Q_{a[bc]}, \qquad \Theta^a = \frac{1}{2} Q^a{}_{bc}\, S^{bc} \tag{3.190}$$

consists of three irreducible pieces

$$Q_{abc} = \overset{\mathrm{c}}{Q}_{abc} + \overset{\mathrm{t}}{Q}_{abc} + \overset{\mathrm{a}}{Q}_{abc} \tag{3.191}$$

object	exists if	and has n components
Q_{abc}		$\frac{d^2(d-1)}{2}$
$\overset{\mathrm{c}}{Q}_{abc}$	$d \geq 3$	$\frac{d(d^2-4)}{3}$
$\overset{\mathrm{t}}{Q}_{abc}$		d
$\overset{\mathrm{a}}{Q}_{abc}$	$d \geq 3$	$\frac{d(d-1)(d-2)}{6}$

The corresponding union of three objects `Torsion 2-forms` is

$$\texttt{Traceless Torsion 2-form THQC'a} = \overset{\mathrm{c}}{\Theta}{}^a = \frac{1}{2}\, \overset{\mathrm{c}}{Q}{}^a{}_{bc}\, S^{bc} \tag{3.192}$$

$$\texttt{Torsion Trace 2-form THQT'a} = \overset{\mathrm{t}}{\Theta}{}^a = \frac{1}{2}\, \overset{\mathrm{t}}{Q}{}^a{}_{bc}\, S^{bc} \tag{3.193}$$

$$\texttt{Antisymmetric Torsion 2-form THQA'a} = \overset{\mathrm{a}}{\Theta}{}^a = \frac{1}{2}\, \overset{\mathrm{a}}{Q}{}^a{}_{bc}\, S^{bc} \tag{3.194}$$

And the auxiliary quantities

$$\texttt{Torsion Trace QT'a} = \; Q^a \tag{3.195}$$

$$\texttt{Torsion Trace 1-form QQ} = \; Q = -\partial_a \lrcorner \Theta^a \tag{3.196}$$

$$\texttt{Antisymmetric Torsion 3-form QQA} = \overset{\mathrm{a}}{Q} = \theta_a \wedge \Theta^a \tag{3.197}$$

The torsion trace $Q^a = Q^m{}_{am}$ can be obtained `From torsion trace 1-form`

$$Q^a = \partial^a \lrcorner Q \tag{3.198}$$

The `Standard way` for the irreducible torsion 2-forms is

$$\overset{\mathrm{t}}{\Theta}{}^a = -\frac{1}{(d-1)} \theta^a \wedge Q \tag{3.199}$$

$$\overset{\mathrm{t}}{\Theta}{}^a = \frac{1}{3}\partial^a \lrcorner \overset{\mathrm{a}}{Q} \tag{3.200}$$

$$\overset{\mathrm{c}}{\Theta}{}^a = \Theta^a - \overset{\mathrm{t}}{\Theta}{}^a - \overset{\mathrm{a}}{\Theta}{}^a \tag{3.201}$$

The rest of this section is valid in dimension 4 only.

In this case one can introduce the torsion pseudo trace

$$\texttt{Torsion Pseudo Trace QP'a} = \ P^a = \overset{*}{Q}{}^{ma}{}_m, \ \overset{*}{Q}{}^a{}_{bc} = \frac{1}{2}\mathcal{E}_{bc}{}^{pq} Q^a{}_{pq} \tag{3.202}$$

which can be computed `From antisymmetric torsion 3-form`

$$P^a = \partial^a \lrcorner * \overset{\mathrm{a}}{Q} \tag{3.203}$$

Finally let us consider the spinorial representation of the torsion. Below the upper sign corresponds to the signature $(-,+,+,+)$ and lower one to the signature $(+,-,-,-)$.

See page ?? *or* ??.

First we introduce the spinorial analog of the torsion tensor

$$Q_{abc} \longleftrightarrow Q_{A\dot{A}BC}\epsilon_{\dot{B}\dot{C}} + Q_{A\dot{A}\dot{B}\dot{C}}\epsilon_{BC} \tag{3.204}$$

where

$$Q_{A\dot{A}BC} = -i * (\Theta_{A\dot{A}} \wedge S_{BC}), \qquad Q_{A\dot{A}\dot{B}\dot{C}} = i * (\Theta_{A\dot{A}} \wedge S_{\dot{B}\dot{C}}) \tag{3.205}$$

These spinors are reducible but the

$$\texttt{Traceless Torsion Spinor QC.ABC.D}\tilde{} = \ C_{ABC\dot{D}} \tag{3.206}$$

$$\overset{\mathrm{c}}{Q}_{abc} \longleftrightarrow C_{ABC\dot{A}}\epsilon_{\dot{B}\dot{C}} + Q_{\dot{A}\dot{B}\dot{C}A}\epsilon_{BC}, \quad C_{\dot{A}\dot{B}\dot{C}A} = \overline{C_{ABC\dot{A}}}$$

is irreducible (symmetric in ABC). And it can be computed `From torsion` by the relation

$$C_{ABC\dot{A}} = Q_{(A|\dot{A}|BC)} \tag{3.207}$$

The torsion trace can be calculated `From torsion using spinors`

$$Q^a \longleftrightarrow Q^{A\dot{A}}, \quad Q_{A\dot{B}} = \mp(Q^M{}_{\dot{B}MA} + Q_A{}^{\dot{M}}{}_{\dot{M}\dot{B}}) \tag{3.208}$$

And similarly the torsion pseudo-trace can be found `From torsion using spinors`

$$P^a \longleftrightarrow P^{A\dot{A}}, \quad P_{A\dot{B}} = \mp i(Q^M{}_{\dot{B}MA} - Q_A{}^{\dot{M}}{}_{\dot{M}\dot{B}}) \tag{3.209}$$

Finally we introduce the `Undotted trace 2-forms` which are selfdual parts of the irreducible torsion 2-forms

$$\texttt{Undotted Traceless Torsion 2-form THQCU'a} = \overset{\text{c}}{\vartheta}{}^{a} \tag{3.210}$$

$$\texttt{Undotted Torsion Trace 2-form THQTU'a} = \overset{\text{t}}{\vartheta}{}^{a} \tag{3.211}$$

$$\texttt{Undotted Antisymmetric Torsion 2-form THQAU'a} = \overset{\text{a}}{\vartheta}{}^{a} \tag{3.212}$$

These quantities will be used in the gravitational equations. *See page* ??.

This complex 2-forms can be obtained by the equations (`Standard way`):

$$\overset{\text{c}}{\vartheta}{}^{a} \longleftrightarrow \overset{\text{c}}{\vartheta}{}^{A\dot{A}} = C^{A}{}_{BC}{}^{\dot{A}} S^{BC} \tag{3.213}$$

$$\overset{\text{t}}{\vartheta}{}^{a} \longleftrightarrow \overset{\text{t}}{\vartheta}{}^{A\dot{A}} = \mp\frac{1}{3} Q_{M}{}^{\dot{A}} S^{AM} \tag{3.214}$$

$$\overset{\text{a}}{\vartheta}{}^{a} \longleftrightarrow \overset{\text{a}}{\vartheta}{}^{A\dot{A}} = \pm\frac{i}{3} P_{M}{}^{\dot{A}} S^{AM} \tag{3.215}$$

3.13 Nonmetricity Decomposition

In general the nonmetricity tensor

$$N_{abc} = N_{(ab)c}, \qquad N_{ab} = N_{abc}\theta^{c} \tag{3.216}$$

consist of 4 irreducible pieces

$$N_{abcd} = \overset{\text{c}}{N}_{abc} + \overset{\text{a}}{N}_{abc} + \overset{\text{t}}{N}_{abc} + \overset{\text{w}}{N}_{abc} \tag{3.217}$$

object	exists if	and has n components
N_{abc}		$\frac{d^2(d+1)}{2}$
$\overset{\text{c}}{N}_{abc}$	$d \geq 3$	$\frac{d(d-1)(d+4)}{6}$
$\overset{\text{a}}{N}_{abc}$		$\frac{d(d^2-4)}{3}$
$\overset{\text{t}}{N}_{abc}$		d
$\overset{\text{w}}{N}_{abc}$		d

The corresponding union of objects `Nonmetricity 1-forms` consist of

$$\texttt{Symmetric Nonmetricity 1-form NC.a.b} = \overset{\text{c}}{N}_{ab} = \overset{\text{c}}{N}_{abc}\,\theta^{c} \tag{3.218}$$

$$\texttt{Antisymmetric Nonmetricity 1-form NA.a.b} = \overset{\text{a}}{N}_{ab}{=}\overset{\text{a}}{N}_{abc}\,\theta^c \tag{3.219}$$

$$\texttt{Nonmetricity Trace 1-form NT.a.b} = \overset{\text{t}}{N}_{ab}{=}\overset{\text{t}}{N}_{abc}\,\theta^c \tag{3.220}$$

$$\texttt{Weyl Nonmetricity 1-form NW.a.b} = \overset{\text{w}}{N}_{ab}{=}\overset{\text{w}}{N}_{abc}\,\theta^c \tag{3.221}$$

We have also two auxiliary 1-forms

$$\texttt{Weyl Vector NNW} = \overset{\text{w}}{N} \tag{3.222}$$

$$\texttt{Nonmetricity Trace NNT} = \overset{\text{t}}{N} \tag{3.223}$$

They are computed according to the following formulas

$$\overset{\text{w}}{N}{=}\, N^a{}_a \tag{3.224}$$

$$\overset{\text{t}}{N}{=}\,\theta^a\,\partial^b\lrcorner N_{ab} - \frac{1}{d}\,\overset{\text{w}}{N} \tag{3.225}$$

$$\overset{\text{w}}{N}_{ab}{=}\,\frac{1}{d}g_{ab}\,\overset{\text{w}}{N} \tag{3.226}$$

$$\overset{\text{t}}{N}_{ab}{=}\,\frac{d}{(d-1)(d+2)}\left[\theta_b\partial_a\lrcorner\,\overset{\text{t}}{N} + \theta_a\partial_b\lrcorner\,\overset{\text{t}}{N} - \frac{2}{d}g_{ab}\,\overset{\text{t}}{N}\right] \tag{3.227}$$

$$\overset{\text{a}}{N}_{ab}{=}\,\frac{1}{3}\left[\partial_a\lrcorner(\theta^m\wedge\,\overset{0}{N}_{bm}) + \partial_b\lrcorner(\theta^m\wedge\,\overset{0}{N}_{am})\right] \tag{3.228}$$

where

$$\overset{0}{N}_{ab}{=}\,N_{abc}{-}\,\overset{\text{t}}{N}_{abc} - \overset{\text{w}}{N}_{abc}$$

And finally

$$\overset{\text{c}}{N}_{ab}{=}\,N_{abc}{-}\,\overset{\text{a}}{N}_{abc} - \overset{\text{t}}{N}_{abc} - \overset{\text{w}}{N}_{abc} \tag{3.229}$$

3.14 Newman-Penrose Formalism

The method of spinorial differential forms described in the previous sections are essentially equivalent to the well known Newman-Penrose formalism but for the sake of convenience GRG has complete set of macro objects which allows to write the Newman-Penrose equations in traditional notation. All these objects refer (up to some sign and 1/2 factors) to other GRG built-in objects.

In this section upper sign corresponds to the signature (−,+,+,+) and lower one to the signature (+,−,−,−). The frame must be null as explained in section **??**.

See page **??**.

For the Newman-Penrose formalism we use notation and conventions of the book *Exact Solutions of the Einstein Field Equations* by D. Kramer, H. Stephani, M. MacCallum and E. Herlt, ed. E. Schmutzer (Berlin, 1980). We denote this book as ESEFE.

We chose the relationships between NP null tetrad and GRG null frame as follows

$$l^\mu = h_0^\mu, \quad k^\mu = h_1^\mu, \quad \overline{m}^\mu = h_2^\mu, \quad m^\mu = h_3^\mu \tag{3.230}$$

The NP vector operators are just the components of the vector frame ∂_a

$$\texttt{DD} = D = \partial_1 \tag{3.231}$$
$$\texttt{DT} = \Delta = \partial_0 \tag{3.232}$$
$$\texttt{du} = \delta = \partial_3 \tag{3.233}$$
$$\texttt{dd} = \overline{\delta} = \partial_2 \tag{3.234}$$

The spin coefficient are the components of the connection 1-form

$$\texttt{SPCOEF.AB.c} = \ \omega_{AB\,c} = \partial_c \lrcorner \omega_{AB} \tag{3.235}$$

or in the NP notation

$$\texttt{alphanp} \ = \alpha = \pm\omega_{(1)2} \tag{3.236}$$
$$\texttt{betanp} \ = \beta = \pm\omega_{(1)3} \tag{3.237}$$
$$\texttt{gammanp} \ = \gamma = \pm\omega_{(1)0} \tag{3.238}$$
$$\texttt{epsilonnp} \ = \epsilon = \pm\omega_{(1)1} \tag{3.239}$$
$$\texttt{kappanp} \ = \kappa = \pm\omega_{(0)1} \tag{3.240}$$
$$\texttt{rhonp} \ = \rho = \pm\omega_{(0)2} \tag{3.241}$$
$$\texttt{sigmanp} \ = \sigma = \pm\omega_{(0)3} \tag{3.242}$$
$$\texttt{taunp} \ = \tau = \pm\omega_{(0)0} \tag{3.243}$$
$$\texttt{munp} \ = \mu = \pm\omega_{(2)3} \tag{3.244}$$
$$\texttt{nunp} \ = \nu = \pm\omega_{(2)0} \tag{3.245}$$
$$\texttt{lambdanp} \ = \lambda = \pm\omega_{(2)2} \tag{3.246}$$
$$\texttt{pinp} \ = \pi = \pm\omega_{(2)1} \tag{3.247}$$
$$\tag{3.248}$$

where the first index of the quantity $\omega_{(AB)c}$ is included inn parentheses to remind that it is summed spinorial index.

Finally for the curvature we have

$$\texttt{PHINP.AB.CD}\tilde{} = \ \Phi_{AB\dot{C}\dot{D}} = \pm\frac{1}{2} C_{AB\dot{C}\dot{D}} \tag{3.249}$$

$$\texttt{PSINP.ABCD} \;=\; \Psi_{ABCD} = C_{ABCD} \tag{3.250}$$

the conventions for the scalar curvature R in ESEFE and in GRG are the same.

For the signature (−,+,+,+) the Newman-Penrose equations for the quantities introduced above can be found in section 7.1 of ESEFE. For other signature (+,−,−,−) one must alter the sign of Ψ_{ABCD}, $\Phi_{AB\dot{C}\dot{D}}$ and R in Eqs. (7.28)–(7.45).

3.15 Electromagnetic Field

Formulas in this section are valid only in spaces with the signature (−,+,...,+) and (+,−,...,−). The sign factor σ in the expressions below is $\sigma = -\mathrm{diag}_0$ (+1 for the first signature and −1 for the second).

Let us introduce the

$$\texttt{EM Potential A} = \;\; A = A_\mu dx^\mu \tag{3.251}$$

and the

$$\texttt{Current 1-form J} = \;\; J = j_\mu dx^\mu \tag{3.252}$$

The EM strength tensor $F_{\alpha\beta} = \partial_\alpha A_\beta - \partial_\beta A_\alpha$

$$\texttt{EM Tensor FT.a.b} = \;\; F_{ab} = \partial_b \lrcorner \partial_a \lrcorner F \tag{3.253}$$

where F is the

$$\texttt{EM 2-form FF} = \;\; F \tag{3.254}$$

which can be found From EM potential

$$F = dA \tag{3.255}$$

or From EM tensor

$$F = \frac{1}{2} F_{ab}\, S^{ab} \tag{3.256}$$

The EM action d-form

$$\texttt{EM Action EMACT} = \;\; L_{\mathrm{EM}} = -\frac{1}{8\pi}\, F \wedge *F \tag{3.257}$$

The Maxwell Equations

$$\texttt{First Maxwell Equation MWFq} = \;\; d * F = -4\pi\sigma\,(-1)^d * J \tag{3.258}$$

$$\texttt{Second Maxwell Equation MWSq} = \;\; dF = 0 \tag{3.259}$$

The current must satisfy the

$$\texttt{Continuity Equation COq} = \ d * J = 0 \tag{3.260}$$

The

$$\texttt{EM Energy-Momentum Tensor TEM.a.b} = \ T^{\rm EM}_{ab} \tag{3.261}$$

is given by the equation

$$T^{\rm EM}_{ab} = \frac{\sigma}{4\pi} F_{am} F_b{}^m + s\sigma\, g_{ab} \, * L_{\rm EM} \tag{3.262}$$

The rest of the section is valid in the dimension 4 only.

In 4 dimensions the tensor F_{ab} and its dual $\overset{*}{F}_{ab} = \frac{1}{2}\mathcal{E}_{ab}{}^{mn} F_{mn}$ are expressed via usual 3-dimensional vectors $\vec{E}$ and $\vec{H}$

$$F_{ab} = -\sigma \begin{pmatrix} E_1 & E_2 & E_3 \\ & -H_3 & H_2 \\ & & -H_1 \end{pmatrix} \tag{3.263}$$

$$\overset{*}{F}_{ab} = \sigma \begin{pmatrix} H_1 & H_2 & H_3 \\ & E_3 & -E_2 \\ & & E_1 \end{pmatrix} \tag{3.264}$$

Similarly for the current we have

$$J = \sigma(-\rho dt + \vec{j}\, d\vec{x}) \tag{3.265}$$

The `EM scalars`

$$\texttt{First EM Scalar SCF} = \ I_1 = \frac{1}{2} F_{ab} F^{ab} = \vec{H}^2 - \vec{E}^2 \tag{3.266}$$

$$\texttt{Second EM Scalar SCS} = \ I_2 = \frac{1}{2}\ \overset{*}{F}_{ab}\ F^{ab} = 2\vec{E}\cdot\vec{H} \tag{3.267}$$

can be obtained as follows by `Standard way`

$$I_1 = -*(F \wedge *F) \tag{3.268}$$

$$I_2 = *(F \wedge F) \tag{3.269}$$

The

$$\texttt{Complex EM 2-form FFU} = \ \Phi \tag{3.270}$$

can be found `From EM 2-form`

$$\Phi = F - i * F \tag{3.271}$$

or From EM Spinor

$$\Phi = 2\Phi_{AB}\, S^{AB} \tag{3.272}$$

The 2-form Φ must obey the

$$\texttt{Selfduality Equation SDq.AB}\tilde{}\texttt{ } = \Phi \wedge S_{\dot{A}\dot{B}} \tag{3.273}$$

and gives rise to the

$$\texttt{Complex Maxwell Equation MWUq} = d\Phi = -4i\sigma\pi * J \tag{3.274}$$

The EM 2-form F can be restored From Complex EM 2-form

$$F = \frac{1}{2}(\Phi + \overline{\Phi}) \tag{3.275}$$

The symmetric

$$\texttt{Undotted EM Spinor FIU.AB} = \Phi_{AB} \tag{3.276}$$

is the spinorial analog of the tensor F_{ab}

$$F_{ab} \longleftrightarrow \epsilon_{AB}\Phi_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}}\Phi_{AB} \tag{3.277}$$

It can be obtained either From complex EM 2-form

$$\Phi_{AB} = -\frac{i}{2} * (\Phi \wedge S_{AB}) \tag{3.278}$$

of From EM 2-form

$$\Phi_{AB} = -i * (F \wedge S_{AB}) \tag{3.279}$$

The

$$\texttt{Complex EM Scalar SCU} = \iota = I_1 - iI_2 \tag{3.280}$$

can be found From EM Spinor

$$\iota = 2\Phi_{AB}\Phi^{AB} \tag{3.281}$$

or From Complex EM 2-form

$$\iota = -\frac{i}{2} * (\Phi \wedge \Phi) \tag{3.282}$$

Finally we have the

$$\texttt{EM Energy-Momentum Spinor TEMS.AB.CD}\tilde{}\texttt{ } = T^{\mathrm{EM}}_{AB\dot{A}\dot{B}} = \frac{1}{2\pi}\Phi_{AB}\Phi_{\dot{A}\dot{B}} \tag{3.283}$$

3.16 Dirac Field

In this section upper sign corresponds to the signature (−,+,+,+) and lower one to the signature (+,−,−,−).

The four component Dirac spinor consists of two 1-index spinors

$$\psi = \begin{pmatrix} \phi^A \\ \chi_{\dot{A}} \end{pmatrix}, \quad \overline{\psi} = \begin{pmatrix} \chi_A & \phi^{\dot{A}} \end{pmatrix} \tag{3.284}$$

Thus we have the `Dirac spinor` as the union of two objects

$$\texttt{Phi Spinor PHI.A} = \phi_A \tag{3.285}$$

$$\texttt{Chi Spinor CHI.B} = \chi_B \tag{3.286}$$

The gamma-matrices are expressed via sigma-matrices as follows

$$\gamma^m = \sqrt{2} \begin{pmatrix} 0 & \sigma^{mA\dot{B}} \\ \sigma^m{}_{B\dot{A}} & 0 \end{pmatrix} \tag{3.287}$$

Dirac field action 4-form

$$\begin{aligned} &\texttt{Dirac Action 4-form DACT} = L_{\mathrm{D}} = \\ &= \left[\frac{i}{2} (\overline{\psi}\gamma^a(\nabla_a + ieA_a)\psi - (\nabla_a - ieA_a)\overline{\psi}\gamma^a\psi) - m_{\mathrm{D}}\overline{\psi}\psi \right] \upsilon \end{aligned} \tag{3.288}$$

The `Standard way` to compute this quantity is

$$\begin{aligned} L_{\mathrm{D}} = -\frac{i}{\sqrt{2}} \left[\phi_{\dot{A}} \theta^{A\dot{A}} \wedge *(D + ieA)\phi_A - \text{c.c.} - \chi_{\dot{A}} \theta^{A\dot{A}} \wedge *(D - ieA)\chi_A - \text{c.c.} \right] - \\ - m_{\mathrm{D}} \left(\phi^A \chi_A + \text{c.c.} \right) \upsilon \end{aligned} \tag{3.289}$$

The `Dirac equation` is

$$\texttt{Phi Dirac Equation DPq.A}\tilde{} = i\sqrt{2}\partial_{B\dot{A}} \lrcorner (D + ieA - \frac{1}{2}Q)\phi^B - m_{\mathrm{D}}\chi_{\dot{A}} = 0 \tag{3.290}$$

$$\texttt{Chi Dirac Equation DCq.A}\tilde{} = i\sqrt{2}\partial_{B\dot{A}} \lrcorner (D - ieA - \frac{1}{2}Q)\chi^B - m_{\mathrm{D}}\phi_{\dot{A}} = 0 \tag{3.291}$$

where Q is the torsion trace 1-form. Notice that terms with the electromagnetic field eA are included in equations iff the value of A is defined. The unit charge e is given by the constant `ECONST`.

The current 1-form can be computed `From Dirac Spinor`

$$J = \mp\sqrt{2}e(\phi_A\phi_{\dot{A}} + \chi_A\chi_{\dot{A}})\theta^{A\dot{A}} \tag{3.292}$$

The symmetrized

$$\texttt{Dirac Energy-Momentum Tensor TDI.a.b} = \ T^{\mathrm{D}}_{ab} \tag{3.293}$$

can be obtained as follows

$$\begin{aligned} T^{\mathrm{D}}_{ab} &= *(\theta_{(a} \wedge T^{\mathrm{D}}_{b)}) \\ T^{\mathrm{D}}_{a} &= \mp\frac{i}{\sqrt{2}}\Big[* \theta^{A\dot{A}}\partial_a \lrcorner (D + ieA)\phi_A\phi_{\dot{A}} - \text{c.c.} \\ &\qquad - * \theta^{A\dot{A}}\partial_a \lrcorner (D - ieA)\chi_A\chi_{\dot{A}} - \text{c.c.}\Big] \pm \partial_a \lrcorner L_{\mathrm{D}} \end{aligned} \tag{3.294}$$

The

$$\texttt{Undotted Dirac Spin 3-Form SPDIU.AB} = \ s^{\mathrm{D}}_{AB} \tag{3.295}$$

$$s^{\mathrm{D}}_{AB} = \frac{i}{2\sqrt{2}}\left(*\theta_{(A|\dot{A}}\phi_{B)}\phi^{\dot{A}} - *\theta_{(A|\dot{A}}\chi_{B)}\chi^{\dot{A}} \right) \tag{3.296}$$

The Dirac field mass m_{D} is given by the constant `DMASS`.

3.17 Scalar Field

Formulas in this section are valid in any dimension with the signature (−,+,...,+) and (+,−,...,−). The sign factor σ is $\sigma = -\mathrm{diag}_0$ (+1 for the first signature and −1 for the second).

The scalar field

$$\texttt{Scalar Field FI} = \ \phi \tag{3.297}$$

The minimal scalar field action d-form

$$\texttt{Minimal Scalar Action SACTMIN} = \ L_{\mathrm{Smin}} = -\frac{1}{2}\left[\sigma(\partial_\alpha\phi)^2 + m_{\mathrm{s}}^2\phi^2\right]\upsilon \tag{3.298}$$

The nonminimal scalar field action

$$\texttt{Scalar Action SACT} = \ L_{\mathrm{S}} = -\frac{1}{2}\left[\sigma(\partial_\alpha\phi)^2 + (m_{\mathrm{s}}^2 + a_0R)\phi^2\right]\upsilon \tag{3.299}$$

The scalar field equation

$$\texttt{Scalar Equation SCq} = \ s\sigma(-1)^d * d * d\phi - (m_{\mathrm{s}}^2 + a_0R)\phi = 0 \tag{3.300}$$

which gives

$$-\sigma \overset{\{\}}{\nabla}{}^{\pi} \overset{\{\}}{\nabla}{}_{\pi}\phi - (m_{\rm s}^2 + a_0 R)\phi = 0$$

The minimal energy-momentum tensor is

$$\texttt{Minimal Scalar Energy-Momentum Tensor TSCLMIN.a.b} = T_{ab}^{\rm Smin} = \\ = \partial_a\phi\partial_b\phi + s\sigma\, g_{ab} * L_{\rm Smin} \tag{3.301}$$

The nonminimal part of the scalar field energy-momentum tensor can be taken into account in the left-hand side of gravitational equations.

See pages **??** *and* **??**.

The scalar field mass $m_{\rm s}$ are given by the constant `SMASS`. The nonminimal interaction terms are included iff the switch `NONMIN` is turned on and the value of nonminimal interaction constant a_0 is determined by the object

$$\texttt{A-Constants ACONST.i2} = a_i \tag{3.302}$$

The default value of a_0 is the constant `AC0`.

3.18 Yang-Mills Field

Formulas in this section are valid in any dimension with the signature $(-,+,...,+)$ and $(+,-,...,-)$. The sign factor σ in the expressions below is $\sigma = -\mathrm{diag}_0$ ($+1$ for the first signature and -1 for the second). The indices $_{i,j,k,l,m,n}$ are the internal space Yang-Mills indices and we a assume that the internal Yang-Mills metric is δ_{ij}.

The Yang-Mills potential 1-form

$$\texttt{YM Potential AYM.i9} = A^i = A^i_\mu dx^\mu \tag{3.303}$$

The structural constants

$$\texttt{Structural Constants SCONST.i9.j9.k9} = c^i{}_{jk} = c^i{}_{[jk]} \tag{3.304}$$

The Yang-Mills strength 2-form

$$\texttt{YM 2-form FFYM.i9} = F^i \tag{3.305}$$

and strength tensor

$$\texttt{YM Tensor FTYM.i9.a.b} = F^i{}_{ab} \tag{3.306}$$

The F^i can be computed `From YM potential`

$$F^i = dA^i + \frac{1}{2} c^i{}_{jk}\, A^j \wedge A^k \tag{3.307}$$

or `From YM tensor`

$$F^i = \frac{1}{2} F^i{}_{ab}\, S^{ab} \tag{3.308}$$

The `Standard way` to find Yang-Mills strength tensor is

$$F^i{}_{ab} = \partial_b \lrcorner \partial_a \lrcorner F^i \tag{3.309}$$

The Yang-Mills action d-form

$$\texttt{YM Action YMACT} = \; L_{\rm YM} = -\frac{1}{8\pi} F^i \wedge *F_i \tag{3.310}$$

The `YM Equations`

$$\texttt{First YM Equation YMFq.i9} = \; d * F^i + c^i{}_{jk}\, A^j \wedge *F^k = 0 \tag{3.311}$$

$$\texttt{Second YM Equation YMSq.i9} = \; dF^i + c^i{}_{jk}\, A^j \wedge F^k = 0 \tag{3.312}$$

The energy-momentum tensor

$$\texttt{YM Energy-Momentum Tensor TYM.a.b} = \; \frac{\sigma}{4\pi} F^i{}_{am} F^i{}_b{}^m + s\sigma\, g_{ab} * L_{\rm YM} \tag{3.313}$$

3.19 Geodesics

The geodesic equation

$$\texttt{Geodesic Equation GEOq\^{}m} = \; \frac{d^2 x^\mu}{dt^2} + \{{}^{\mu}_{\pi\tau}\} \frac{dx^\pi}{dt} \frac{dx^\tau}{dt} = 0 \tag{3.314}$$

See page ??. Here the parameter t must be declared by the `Affine parameter` declaration.

3.20 Null Congruence and Optical Scalars

Let us consider the congruence defined by the vector field k^α

$$\texttt{Congruence KV} = \; k = k^\mu \partial_\mu \tag{3.315}$$

This congruence is null iff

$$\texttt{Null Congruence Condition NCo} = \; k \cdot k = 0 \tag{3.316}$$

holds.

The congruence is geodesic iff the condition

$$\texttt{Geodesics Congruence Condition GCo'a} = \; k^\mu \overset{\{\}}{\nabla}_\mu k^a = 0 \tag{3.317}$$

is fulfilled.

For the null geodesic congruence one can calculate the `Optical scalars`

$$\texttt{Congruence Expansion thetaO} = \; \theta = \frac{1}{2} \overset{\{\}}{\nabla}^\pi k_\pi \tag{3.318}$$

$$\texttt{Congruence Squared Rotation omegaSQO} = \; \omega^2 = \frac{1}{2} (\overset{\{\}}{\nabla}_{[\alpha} k_{\beta]})^2 \tag{3.319}$$

$$\texttt{Congruence Squared Shear sigmaSQO} = \; \sigma\overline{\sigma} = \frac{1}{2} \left[(\overset{\{\}}{\nabla}_{(\alpha} k_{\beta)})^2 - 2\theta^2 \right] \tag{3.320}$$

3.21 Timelike Congruences and Kinematics

Let us consider the congruence determined by the velocity vector u^α

$$\texttt{Velocity UU'a} = \; u^a \tag{3.321}$$

$$\texttt{Velocity Vector UV} = \; u = u^a \partial_a \tag{3.322}$$

The velocity vector must be normalized and the quantity

$$\texttt{Velocity Square USQ} = \; u^2 = u \cdot u \tag{3.323}$$

must be constant but nonzero.

If the frame metric coincides with its default diagonal value $g_{ab} = \mathrm{diag}(-1, \ldots)$ *See page* **??**.
then `By default` we have for the velocity

$$u^a = (1, 0, \ldots, 0) \tag{3.324}$$

which means that the congruence is comoving in the given frame.

In general case the velocity can be obtained `From velocity vector`

$$u^a = u \lrcorner \theta^a \tag{3.325}$$

We introduce the auxiliary object

$$\texttt{Projector PR'a.b} = \; P^a{}_b = \delta^a_b - \frac{1}{u^2} u^a n_b \tag{3.326}$$

The following four quantities called `Kinematics` comprise the complete set of the congruence characteristics

$$\texttt{Acceleration accU'a} = \; A^a = \overset{\{\}}{\nabla}_u u^a \tag{3.327}$$

$$\texttt{Vorticity omegaU.a.b} = \; \omega_{ab} = P^m{}_a P^n{}_b \overset{\{\}}{\nabla}_{[m} u_{n]} \tag{3.328}$$

$$\texttt{Volume Expansion thetaU} = \; \Theta = \overset{\{\}}{\nabla}_a u^a \tag{3.329}$$

$$\texttt{Shear sigmaU.a.b} = \; P^m{}_a P^n{}_b \overset{\{\}}{\nabla}_{(m} u_{n)} - \frac{1}{(d-1)} P_{ab} \Theta \tag{3.330}$$

3.22 Ideal And Spin Fluid

The ideal fluid is characterized by the

$$\texttt{Pressure PRES} = \; p \tag{3.331}$$

and

$$\texttt{Energy Density ENER} = \; \varepsilon \tag{3.332}$$

The ideal fluid energy-momentum tensor is

$$\begin{aligned} \texttt{Ideal Fluid Energy-Momentum Tensor TIFL.a.b} = T^{\mathrm{IF}}_{ab} = \\ = (\varepsilon + p) u_a u_b - u^2 p g_{ab} \end{aligned} \tag{3.333}$$

The rest of the section requires the nonmetricity be zero (`NONMETR` is off).
In addition spin-fluid is characterized by

$$\texttt{Spin Density SPFLT.a.b} \; = \; S^{\mathrm{SF}}_{ab} = S^{\mathrm{SF}}_{[ab]} \tag{3.334}$$

or equivalently by

$$\texttt{Spin Density 2-form SPFL} \; = \; S^{\mathrm{SF}} \tag{3.335}$$

The spin 2-form can be obtained `From spin density`

$$S^{\mathrm{SF}} = \frac{1}{2} S^{\mathrm{SF}}_{ab} \theta^a \wedge \theta^a \tag{3.336}$$

and s_{ab} is determined `From spin density 2-form`

$$S^{\rm SF}_{ab} = \partial_b \lrcorner \partial_a \lrcorner S^{\rm SF} \tag{3.337}$$

The spin density must satisfy the Frenkel condition

$$\texttt{Frenkel Condition FCo} = \; u \lrcorner S^{\rm SF} = 0 \tag{3.338}$$

The spin fluid energy-momentum tensor is

$$\begin{aligned}\texttt{Spin Fluid Energy-Momentum Tensor TSFL.a.b} = T^{\rm SF}_{ab} = \\ = (\varepsilon + p) u_a u_b - u^2 p g_{ab} + \Delta_{(ab)}\end{aligned} \tag{3.339}$$

where

$$\Delta_{ab} = -2(g^{cd} + u^{-2}\, u^c u^d) \nabla_c S^{\rm SF}_{(ab)d} \tag{3.340}$$

$$s^{\rm SF}_{abc} = u_a \, S^{\rm SF}_{bc} \tag{3.341}$$

if torsion is zero (`TORSION` off) and

$$\Delta_{ab} = 2u^{-2}\, u_a u^d \, \nabla_u S^{\rm SF}_{bd} \tag{3.342}$$

if torsion is nonzero (`TORSION` on).

Notice that the energy-momentum tensor $T^{\rm SF}_{ab}$ is symmetrized. *See page* **??**.

Finally yet another representation for the spin is the undotted spin 3-form

$$\texttt{Undotted Fluid Spin 3-form SPFLU.AB} \; = \; s^{\rm SF}_{AB} \tag{3.343}$$

which is given by the standard spinor $\longleftrightarrow$ tensor correspondence rules

$$s^{\rm SF}_{mab} \, * \theta^m \longleftrightarrow \epsilon_{AB} s^{\rm SF}_{\dot A \dot B} + \epsilon_{\dot A \dot B} s^{\rm SF}_{AB} \tag{3.344}$$

according to Eq. (**??**). This quantity is used in the right-hand side of gravitational equations. *See page* **??**.

3.23 Total Energy-Momentum And Spin

The total energy-momentum tensor

$$\texttt{Total Energy-Momentum Tensor TENMOM.a.b} = \; T_{ab} \tag{3.345}$$

and the total undotted spin 3-form *See pages* **??** *and* **??**.

$$\texttt{Total Undotted Spin 3-form SPINU.AB} = \; s_{AB} \tag{3.346}$$

play the role of sources in the right-hand side of the gravitational equations.

The expression for these quantities read

$$T_{ab} = T^{\rm D}_{ab} + T^{\rm EM}_{ab} + T^{\rm YM}_{ab} + T^{\rm Smin}_{ab} + T^{\rm IF}_{ab} + T^{\rm SF}_{ab} \tag{3.347}$$

$$s_{AB} = s^{\rm D}_{AB} + s^{\rm SF}_{AB} \tag{3.348}$$

When T_{ab} and s_{AB} are calculated GRG does not tries to find value of all objects in the right-hand side of Eqs. (**??**), (**??**) instead it adds only the quantities whose value are currently defined. In particular if none of above tensors and spinors are defined then $T_{ab} = s_{AB} = 0$.

Notice that T_{ab} and all tensors in the right-hand side of Eq. (**??**) are symmetric.
See page **??**. They are the symmetric parts of the canonical energy-momentum tensors.

In addition we introduce the

$$\texttt{Total Energy-Momentum Trace TENMOMT} = \ T = T^{a}{}_{a} \tag{3.349}$$

and the spinor

$$\texttt{Total Energy-Momentum Spinor TENMOMS.AB.CD\~{}} = \ T_{AB\dot{C}\dot{D}} \tag{3.350}$$

is a spinorial equivalent of the traceless part of T_{ab}

$$T_{ab} - \frac{1}{4} g_{ab} T \longleftrightarrow T_{AB\dot{A}\dot{B}} \tag{3.351}$$

3.24 Einstein Equations

The Einstein equation

$$\texttt{Einstein Equation EEq.a.b} = \ R_{ab} - \frac{1}{2} g_{ab} R + \Lambda R = 8\pi G\, T_{ab} \tag{3.352}$$

And the `Spinor Einstein equations`

$$\texttt{Traceless Einstein Equation CEEq.AB.CD\~{}} = \ C_{AB\dot{C}\dot{D}} = 8\pi G\, T_{AB\dot{C}\dot{D}} \tag{3.353}$$

$$\texttt{Trace of Einstein Equation TEEq} = \ R - 4\Lambda = -8\pi G\, T \tag{3.354}$$

The cosmological constant is included in these equations iff the switch `CCONST` is turned on and its value is given by the constant `CCONST`. The gravitational constant G is given by the constant `GCONST`.

3.25 Gravitational Equations in Space With Torsion

Equations in this section are valid in dimension $d = 4$ with the signature $(-,+,+,+)$ and $(+,-,-,-)$ only. The $\sigma = 1$ for the first signature and $\sigma = -1$ for the second. The nonmetricity must be zero and the switch `NONMETR` turned off.

Let us consider the action

$$S = \int \Big[\frac{\sigma}{16\pi G} L_{\rm g} + L_{\rm m}\Big] \tag{3.355}$$

where

$$\texttt{Action LACT} = \; L_{\rm g} = \upsilon\, \mathcal{L}_{\rm g} \tag{3.356}$$

is the gravitational action 4-form and

$$L_{\rm m} = \upsilon\, \mathcal{L}_{\rm m} \tag{3.357}$$

is the matter action 4-form.

Let us define the following variational derivatives

$$Z^{\mu}{}_{a} = \frac{1}{\sqrt{-g}} \frac{\delta \sqrt{-g}\mathcal{L}_{\rm g}}{\delta h^{a}_{\mu}}, \qquad t^{\mu}{}_{a} = \frac{\sigma}{\sqrt{-g}} \frac{\delta \sqrt{-g}\mathcal{L}_{\rm m}}{\delta h^{a}_{\mu}} \tag{3.358}$$

$$V^{\mu}{}_{ab} = \frac{1}{\sqrt{-g}} \frac{\delta \sqrt{-g}\mathcal{L}_{\rm g}}{\delta \omega^{ab}{}_{\mu}}, \qquad s^{\mu}{}_{ab} = \frac{\sigma}{\sqrt{-g}} \frac{\delta \sqrt{-g}\mathcal{L}_{\rm m}}{\delta \omega^{ab}{}_{\mu}} \tag{3.359}$$

Then the gravitational equations reads

$$Z^{\mu}{}_{a} = -16\pi G\, t^{\mu}{}_{a} \tag{3.360}$$

$$V^{\mu}{}_{ab} = -16\pi G\, s^{\mu}{}_{ab} \tag{3.361}$$

Here the first equation is an analog of Einstein equation and has the canonical nonsymmetric energy-momentum tensor $t^{\mu}{}_{a}$ as a source. The source in the second equation is the spin tensor $s^{\mu}{}_{ab}$.

Now we rewrite these equation in other equivalent form. First let us define the following 3-forms

$$Z_a = Z^{m}{}_{a} * \theta_m, \qquad t_a = t^{m}{}_{a} * \theta_m \tag{3.362}$$

$$V_{ab} = V^{m}{}_{ab} * \theta_m, \qquad s_{ab} = s^{m}{}_{ab} * \theta_m \tag{3.363}$$

Notice that Eq. (**??**) is not symmetric but the antisymmetric part of this equation is expressed via second Eq. (**??**) due to Bianchi identity. Therefore

only the symmetric part of Eq. (**??**) is essential. Eq. (**??**) is antisymmetric and we can consider its spinorial analog using the standard relations

$$V_{ab} \longleftrightarrow V_{A\dot{A}B\dot{B}} = \epsilon_{AB} V_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}} V_{AB} \tag{3.364}$$

$$s_{ab} \longleftrightarrow s_{A\dot{A}B\dot{B}} = \epsilon_{AB} s_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}} s_{AB} \tag{3.365}$$

See page **??**.

Finally we define the `Gravitational equations` in the form

$$\texttt{Metric Equation METRq.a.b} = -\frac{1}{2} Z_{(ab)} = 8\pi G\, T_{ab} \tag{3.366}$$

$$\texttt{Torsion Equation TORSq.AB} = V_{AB} = -16\pi G\, s_{AB} \tag{3.367}$$

See page **??**. where the currents in the right-hand side of equations are

$$\texttt{Total Energy-Momentum Tensor TENMOM.a.b} = T_{ab} = t_{(ab)} \tag{3.368}$$

$$\texttt{Total Undotted Spin 3-form SPINU.AB} = s_{AB} \tag{3.369}$$

Now let us consider the equations which are used in GRG to compute the left-hand side of the gravitational equations $Z_{(ab)}$ and V_{AB}. We have to emphasize

See page **??**. that we use spinors and all restrictions imposed by the spinorial formalism must be fulfilled.

We consider the Lagrangian which is an arbitrary algebraic function of the curvature and torsion tensors

$$\mathcal{L}_{\rm g} = \mathcal{L}_{\rm g}(R_{abcd}, Q_{abc}) \tag{3.370}$$

No derivatives of the torsion or curvature are permitted. For such a Lagrangian we define so called curvature and torsion momentums

$$\widetilde{R}^{abcd} = 2\frac{\partial \mathcal{L}_{\rm g}(R,Q)}{\partial R_{abcd}}, \qquad \widetilde{Q}^{abc} = 2\frac{\partial \mathcal{L}_{\rm g}(R,Q)}{\partial Q_{abc}}, \tag{3.371}$$

The corresponding objects are

$$\texttt{Undotted Curvature Momentum POMEGAU.AB} = \widetilde{\Omega}_{AB} \tag{3.372}$$

$$\texttt{Torsion Momentum PTHETA'a} = \widetilde{\Theta}^{a} \tag{3.373}$$

where

$$\widetilde{\Omega}_{ab} = \frac{1}{2} \widetilde{R}_{abcd}\, S^{cd} \tag{3.374}$$

$$\widetilde{\Theta}^{a} = \frac{1}{2} \widetilde{Q}^{a}{}_{cd}\, S^{cd} \tag{3.375}$$

and

$$\widetilde{\Omega}_{ab} \longleftrightarrow \widetilde{\Omega}_{A\dot{A}B\dot{B}} = \epsilon_{AB}\widetilde{\Omega}_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}}\widetilde{\Omega}_{AB} \tag{3.376}$$

If value of three objects $L_{\rm g}$ (`Action`), $\widetilde{\Omega}_{AB}$ (`Undotted curvature momentum`) and $\widetilde{\Theta}^a$ are specified then the `Gravitational equations` can be calculated using equations (`Standard way`)

$$\begin{aligned} Z_{(ab)} &= *(\theta_{(a} \wedge Z_{b)}), \\ Z_a &= D\widetilde{\Theta}_a + (\partial_a \lrcorner \Theta^b) \wedge \widetilde{\Theta}_b + 2(\partial_a \lrcorner \Omega^{MN}) \wedge \widetilde{\Omega}_{MN} \\ &\quad + \text{c.c.} - \partial_a L_{\rm g} \end{aligned} \tag{3.377}$$

$$\begin{aligned} V_{AB} &= -D\widetilde{\Omega}_{AB} - \widetilde{\Theta}_{AB}, \\ \theta_{[a} \wedge \widetilde{\Theta}_{b]} &\longleftrightarrow \epsilon_{AB}\widetilde{\Theta}_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}}\widetilde{\Theta}_{AB} \end{aligned} \tag{3.378}$$

Since gravitational equations are computed in the spinorial formalism with the standard null frame the metric equation is complex and components 02, 12, 22 are conjugated to 03. 13, 33. Since these components are not independent For the sake of efficiency by default GRG computes only the 00, 01, 02, 11, 12, 22 and 23 components of $Z_{(ab)}$ only. If you want to have all components the switch `FULL` must be turned on.

See pages **??** *and* **??**.

These equations allows one to compute field equations for gravity theory with an arbitrary Lagrangian. But the value of three quantities $L_{\rm g}$, $\widetilde{\Omega}_{AB}$ and $\widetilde{\Theta}^a$ must be specified by the user. In addition GRG has built-in formulas for the most general quadratic in torsion and curvature Lagrangian. The `Standard way` for $L_{\rm g}$, $\widetilde{\Omega}_{AB}$ and $\widetilde{\Theta}^a$ is

$$\widetilde{\Theta}^a = i\mu_1(\overset{\rm c}{\vartheta}{}^a - \text{c.c.}) + i\mu_2(\overset{\rm t}{\vartheta}{}^a - \text{c.c.}) + i\mu_3(\overset{\rm a}{\vartheta}{}^a - \text{c.c.}), \tag{3.379}$$

$$\begin{aligned} \widetilde{\Omega}_{AB} = {} & i(\lambda_0 - \sigma\, 8\pi G\, a_0\phi^2)\, S_{AB} \\ & + i\lambda_1 \overset{\rm w}{\Omega}_{AB} - i\lambda_2 \overset{\rm c}{\Omega}_{AB} + i\lambda_3 \overset{\rm r}{\Omega}_{AB} \\ & + i\lambda_4 \overset{\rm a}{\Omega}_{AB} - i\lambda_5 \overset{\rm b}{\Omega}_{AB} + i\lambda_6 \overset{\rm d}{\Omega}_{AB}, \end{aligned} \tag{3.380}$$

$$\begin{aligned} L_{\rm g} = {} & (-2\Lambda + \frac{1}{2}\lambda_0 R - \sigma\, 4\pi G a_0 \phi^2 R)\upsilon + \Omega^{AB} \wedge \widetilde{\Omega}_{AB} + \text{c.c.} \\ & + \frac{1}{2}\Theta^a \wedge \widetilde{\Theta}_a \end{aligned} \tag{3.381}$$

The cosmological term Λ is included into equations iff the switch `CCONST` is turned on and the value of Λ is given by the constant `CCONST`. The term with

the scalar field ϕ is included into equations iff the switch NONMIN is on. The gravitational constant G is given by the constant GCONST. The parameters of the quadratic Lagrangian are given by the objects

$$\texttt{L-Constants LCONST.i6} = \lambda_i \tag{3.382}$$

$$\texttt{M-Constants MCONST.i3} = \mu_i \tag{3.383}$$

$$\texttt{A-Constants ACONST.i2} = a_i \tag{3.384}$$

The default value of these objects (Standard way) is

$$\lambda_i = (\texttt{LC0}, \texttt{LC1}, \texttt{LC2}, \texttt{LC3}, \texttt{LC4}, \texttt{LC5}, \texttt{LC6}), \tag{3.385}$$

$$\mu_i = (0, \texttt{MC1}, \texttt{MC2}, \texttt{MC32}), \tag{3.386}$$

$$a_i = (\texttt{AC0}, 0, 0) \tag{3.387}$$

3.26 Gravitational Equations in Riemann Space

Equations in this section are valid in dimension $d = 4$ with the signature (−,+,+,+) and (+,−,−,−) only. The $\sigma = 1$ for the first signature and $\sigma = -1$ for the second. The nonmetricity and torsion must be zero and the switches NONMETR and TORSION must be turned off.

Let us consider the action

$$S = \int \Big[\frac{\sigma}{16\pi G} L_{\rm g} + L_{\rm m}\Big] \tag{3.388}$$

where

$$\texttt{Action LACT} = L_{\rm g} = \upsilon\, \mathcal{L}_{\rm g} \tag{3.389}$$

is the gravitational action 4-form and

$$L_{\rm m} = \upsilon\, \mathcal{L}_{\rm m} \tag{3.390}$$

is the matter action 4-form.

Let us define the following variational derivatives

$$Z^{\mu}{}_{a} = \frac{1}{\sqrt{-g}} \frac{\delta\sqrt{-g}\mathcal{L}_{\rm g}}{\delta h^a_\mu}, \qquad T^{\mu}{}_{a} = \frac{\sigma}{\sqrt{-g}} \frac{\delta\sqrt{-g}\mathcal{L}_{\rm m}}{\delta h^a_\mu} \tag{3.391}$$

Then the Metric equation is

$$\texttt{Metric Equation METRq.a.b} = -\frac{1}{2} Z_{ab} = 8\pi G\, T_{ab} \tag{3.392}$$

Notice that Z_{ab} and T_{ab} are automatically symmetric.

Let us define 3-form

$$Z_a = Z^m{}_a * \theta_m, \qquad t_a = t^m{}_a * \theta_m \tag{3.393}$$

Now we consider the equations which are used in GRG to compute the left-hand side of the metric equation Z_{ab}. We have to emphasize that we use spinors and all restrictions imposed by the spinorial formalism must be fulfilled. *See pages* **??** *or* **??**.

We consider the Lagrangian which is an arbitrary algebraic function of the curvature tensor

$$\mathcal{L}_{\mathrm{g}} = \mathcal{L}_{\mathrm{g}}(R_{abcd}) \tag{3.394}$$

No derivatives of the curvature are permitted. For such a Lagrangian we define so called curvature momentum

$$\widetilde{R}^{abcd} = 2\frac{\partial \mathcal{L}_{\mathrm{g}}(R)}{\partial R_{abcd}} \tag{3.395}$$

The corresponding GRG built-in object is

$$\texttt{Undotted Curvature Momentum POMEGAU.AB} = \widetilde{\Omega}_{AB} \tag{3.396}$$

where

$$\widetilde{\Omega}_{ab} = \frac{1}{2}\widetilde{R}_{abcd}\, S^{cd} \tag{3.397}$$

$$\tag{3.398}$$

and

$$\widetilde{\Omega}_{ab} \longleftrightarrow \widetilde{\Omega}_{A\dot{A}B\dot{B}} = \epsilon_{AB}\widetilde{\Omega}_{\dot{A}\dot{B}} + \epsilon_{\dot{A}\dot{B}}\widetilde{\Omega}_{AB} \tag{3.399}$$

If value of the objects L_{g} (`Action`) and $\widetilde{\Omega}_{AB}$ (`Undotted curvature momentum`) is specified then the `Metric equation` can be calculated using equations (`Standard way`)

$$\begin{aligned} Z_{ab} &= *(\theta_{(a} \wedge Z_{b)}), \\ Z_a &= D[2\partial_m \lrcorner D\widetilde{\Omega}_a{}^m - \frac{1}{2}\theta_a \wedge (\partial_m \lrcorner \partial_n \lrcorner D\widetilde{\Omega}^{mn})] \\ &\quad +2(\partial_a \lrcorner \Omega^{MN}) \wedge \widetilde{\Omega}_{MN} + \text{c.c.} - \partial_a L_{\mathrm{g}} \end{aligned} \tag{3.400}$$

Since gravitational equations are computed in the spinorial formalism with the standard null frame the metric equation is complex and components 02, 12, 22 are conjugated to 03, 13, 33. For the sake of efficiency by default GRG computes *See page* **??** *or page* **??**.

only the components 00, 01, 02, 11, 12, 22 and 23 only. If you want to have all components the switch `FULL` must be turned on.

These equations allows one to compute field equations for gravity theory with an arbitrary Lagrangian. But the value of three quantities $L_{\rm g}$ and $\widetilde{\Omega}_{AB}$ must be specified by user. In addition GRG has built-in formulas for the most general quadratic in the curvature Lagrangian. The `Standard way` for $L_{\rm g}$ and $\widetilde{\Omega}_{AB}$ is

$$\begin{aligned}\widetilde{\Omega}_{AB} &= i(\lambda_0 - \sigma 8\pi G\, a_0\phi^2)\, S_{AB} \\ &\quad + i\lambda_1 \overset{\rm w}{\Omega}_{AB} - i\lambda_2 \overset{\rm c}{\Omega}_{AB} + i\lambda_3 \overset{\rm r}{\Omega}_{AB}, \end{aligned} \tag{3.401}$$

$$L_{\rm g} = (-2\Lambda + \frac{1}{2}\lambda_0 R - \sigma 4\pi G a_0\phi^2 R)\upsilon + \Omega^{AB} \wedge \widetilde{\Omega}_{AB} + \text{c.c.} \tag{3.402}$$

The cosmological term is included into equations iff the switch `CCONST` is on and the value of Λ is given by the constant `CCONST`. The term with the scalar field ϕ is included into equations iff the switch `NONMIN` is on. The gravitational constant G is given by the constant `GCONST`. The parameters of the quadratic lagrangian are given by the object

$$\texttt{L-Constants LCONST.i6} = \lambda_i \tag{3.403}$$

$$\texttt{A-Constants ACONST.i2} = a_i \tag{3.404}$$

The default value of these objects (`Standard way`) is

$$\lambda_i = (\texttt{LC0}, \texttt{LC1}, \texttt{LC2}, \texttt{LC3}, \texttt{LC4}, \texttt{LC5}, \texttt{LC6}), \tag{3.405}$$

$$a_i = (\texttt{AC0}, 0, 0) \tag{3.406}$$

APPENDIX ***A***

GRG **Switches**

Switch	Default State	Description	See page
AEVAL	Off	Use AEVAL instead of REVAL.	??
WRS	On	Re-simplify object before printing.	??
WMATR	Off	Write 2-index objects in matrix form.	??
TORSION	Off	Torsion.	??
NONMETR	Off	Nonmetricity.	??
UNLCORD	On	Save coordinates in Unload.	??
AUTO	On	Automatic object calculation in expressions.	??
TRACE	On	Trace the calculation process.	??
SHOWCOMMANDS	Off	Show compound command expansion.	??
EXPANDSYM	Off	Enable Sy Asy Cy in expressions	??
DFPCOMMUTE	On	Commutativity of DFP derivatives.	??
NONMIN	Off	Nonminimal interaction for scalar field.	??
NOFREEVARS	Off	Prohibit free variables in Print.	??
CCONST	Off	Include cosmological constant in equations.	??
FULL	Off	Number of components in Metric Equation.	??
LATEX	Off	LaTeX output mode.	??
GRG	Off	GRG output mode.	??
REDUCE	Off	Reduce output mode.	??
MAPLE	Off	Maple output mode.	??
MATH	Off	Mathematica output mode.	??
MACSYMA	Off	Macsyma output mode.	??
DFINDEXED	Off	Print DF in index notation.	??
BATCH	Off	Batch mode.	??
HOLONOMIC	On	Keep frame holonomic.	??
SHOWEXPR	Off	Print expressions during algebraic classification.	??

APPENDIX ***B***

Macro Objects

Macro objects can be used in expression, in `Write` and `Show` commands but not in the `Find` command. The notation for indices is the same as in the `New Object` declaration (see page ??).

B.1 Dimension and Signature

`dim`	Dimension d
`sdiag.idim`	`sdiag(n)` is the n'th element of the signature $\mathrm{diag}(-1, +1 \ldots)$
`sign` `sgnt`	Product of the signature specification elements $\prod_{n=0}^{d-1}$ `sdiag(`n`)`
`mpsgn` `pmsgn`	`sdiag(0)` `-sdiag(0)`

B.2 Metric and Frame

`x^m` `X^m`	m'th coordinate
`h'a_m` `hi.a^m`	Frame coefficients
`g_m_n` `gi^m^n`	Holonomic metric

B.3 Delta and Epsilon Symbols

`del'a.b` `delh^m_n`	Delta symbols

`eps.a.b.c.d`
`epsi'a'b'c'd`
`epsh_m_n_p_q`
`epsih^m^n^p^q`
	Totally antisymmetric symbols
	(number of indices depend on d)

B.4 Spinors

`DEL'A.B`	Delta symbol
`EPS.A.B` `EPSI'A'B`	Spinorial metric
`sigma'a.A.B~` `sigmai.a'A'B~`	Sigma matrices
`cci.i3`	Frame index conjugation in standard null frame `cci(0)=0 cci(1)=1 cci(2)=3 cci(3)=2`

B.5 Connection Coefficients

`CHR^m_n_p`
`CHRF_m_n_p`
`CHRT_m`
	Christoffel symbols $\{^{\mu}_{\nu\pi}\}$
	and $[_{\mu,\nu\pi}]$
	Christoffel symbol trace $\{^{\pi}_{\pi\mu}\}$

`SPCOEF.AB.c`	Spin coefficients $\omega_{AB\,c}$

B.6 NP Formalism

PHINP.AB.CD PSINP.ABCD	$\Phi_{AB\dot{c}\dot{D}}$ Ψ_{ABCD}
alphanp	α
betanp	β
gammanp	γ
epsilonnp	ϵ
kappanp	κ
rhonp	ρ
sigmanp	σ
taunp	τ
munp	μ
nunp	ν
lambdanp	λ
pinp	π
DD	D
DT	Δ
du	δ
dd	$\overline{\delta}$

APPENDIX **C**

Objects

Here we present the complete list of built-in objects with names and identifiers. The notation for indices is the same as in the `New Object` declaration (see page **??**). Some names (group names) refer to a set of objects. For example the group name `Spinorial S - forms` below denotes `SU.AB` and `SD.AB~`

C.1 Metric, Frame, Basis, Volume ...

Frame	T'a
Vector Frame	D.a
Metric	G.a.b
Inverse Metric	GI'a'b
Det of Metric	detG
Det of Holonomic Metric	detg
Sqrt Det of Metric	sdetG
Volume	VOL
Basis	b'idim
Vector Basis	e.idim
S-forms	S'a'b
Spinorial S-forms	
Undotted S-forms	SU.AB
Dotted S-forms	SD.AB~

C.2 Rotation Matrices

Frame Transformation	L'a.b
Spinorial Transformation	LS.A'B

C.3 Connection and related objects

Frame Connection	omega'a.b
Holonomic Connection	GAMMA^m_n
Spinorial Connection	
Undotted Connection	omegau.AB
Dotted Connection	omegad.AB~
Riemann Frame Connection	romega'a.b
Riemann Holonomic Connection	RGAMMA^m_n
Riemann Spinorial Connection	
Riemann Undotted Connection	romegau.AB
Riemann Dotted Connection	romegad.AB~
Connection Defect	K'a.b

C.4 Torsion

Torsion	THETA'a
Contorsion	KQ'a.b
Torsion Trace 1-form	QQ
Antisymmetric Torsion 3-form	QQA
Spinorial Contorsion	
Undotted Contorsion	KU.AB
Dotted Contorsion	KD.AB~
Torsion Spinors Torsion Components	
Torsion Trace	QT'a
Torsion Pseudo Trace	QP'a
Traceless Torsion Spinor	QC.ABC.D~
Torsion 2-forms	
Traceless Torsion 2-form	THQC'a
Torsion Trace 2-form	THQT'a
Antisymmetric Torsion 2-form	THQA'a
Undotted Torsion 2-forms	
Undotted Torsion Trace 2-form	THQTU'a
Undotted Antisymmetric Torsion 2-form	THQAU'a
Undotted Traceless Torsion 2-form	THQCU'a

C.5 Curvature

Curvature	OMEGA'a.b
Spinorial Curvature	
Undotted Curvature	OMEGAU.AB
Dotted Curvature	OMEGAD.AB~
Riemann Tensor	RIM'a.b.c.d
Ricci Tensor	RIC.a.b
A-Ricci Tensor	RICA.a.b
S-Ricci Tensor	RICS.a.b
Homothetic Curvature	OMEGAH
Einstein Tensor	GT.a.b
Curvature Spinors Curvature Components	
Weyl Spinor	RW.ABCD
Traceless Ricci Spinor	RC.AB.CD~
Scalar Curvature	RR
Ricanti Spinor	RA.AB
Traceless Deviation Spinor	RB.AB.CD~
Scalar Deviation	RD
Undotted Curvature 2-forms	
Undotted Weyl 2-form	OMWU.AB
Undotted Traceless Ricci 2-form	OMCU.AB
Undotted Scalar Curvature 2-form	OMRU.AB
Undotted Ricanti 2-form	OMAU.AB
Undotted Traceless Deviation 2-form	OMBU.AB
Undotted Scalar Deviation 2-form	OMDU.AB
Curvature 2-forms	
Weyl 2-form	OMW.a.b
Traceless Ricci 2-form	OMC.a.b
Scalar Curvature 2-form	OMR.a.b
Ricanti 2-form	OMA.a.b
Traceless Deviation 2-form	OMB.a.b
Antisymmetric Curvature 2-form	OMD.a.b
Homothetic Curvature 2-form	OSH.a.b
Antisymmetric S-Ricci 2-form	OSA.a.b
Traceless S-Ricci 2-form	OSC.a.b
Antisymmetric S-Curvature 2-form	OSV.a.b
Symmetric S-Curvature 2-form	OSU.a.b

C.6 Nonmetricity

Nonmetricity	N.a.b
Nonmetricity Defect	KN'a.b
Weyl Vector	NNW
Nonmetricity Trace	NNT
Nonmetricity 1-forms	
Symmetric Nonmetricity 1-form	NC.a.b
Antisymmetric Nonmetricity 1-form	NA.a.b
Nonmetricity Trace 1-form	NT.a.b
Weyl Nonmetricity 1-form	NW.a.b

C.7 EM field

EM Potential	A
Current 1-form	J
EM Action	EMACT
EM 2-form	FF
EM Tensor	FT.a.b
Maxwell Equations	
First Maxwell Equation	MWFq
Second Maxwell Equation	MWSq
Continuity Equation	COq
EM Energy-Momentum Tensor	TEM.a.b
EM Scalars	
First EM Scalar	SCF
Second EM Scalar	SCS
Selfduality Equation	SDq.AB~
Complex EM 2-form	FFU
Complex Maxwell Equation	MWUq
Undotted EM Spinor	FIU.AB
Complex EM Scalar	SCU
EM Energy-Momentum Spinor	TEMS.AB.CD~

C.8 Scalar field

Scalar Equation	SCq
Scalar Field	FI
Scalar Action	SACT
Minimal Scalar Action	SACTMIN
Minimal Scalar Energy-Momentum Tensor	TSCLMIN.a.b

C.9 YM field

YM Potential	AYM.i9
Structural Constants	SCONST.i9.j9.k9
YM Action	YMACT
YM 2-form	FFYM.i9
YM Tensor	FTYM.i9.a.b
YM Equations	
First YM Equation	YMFq.i9
Second YM Equation	YMSq.i9
YM Energy-Momentum Tensor	TYM.a.b

C.10 Dirac field

Dirac Spinor	
Phi Spinor	PHI.A
Chi Spinor	CHI.B
Dirac Action 4-form	DACT
Undotted Dirac Spin 3-Form	SPDIU.AB
Dirac Energy-Momentum Tensor	TDI.a.b
Dirac Equation	
Phi Dirac Equation	DPq.A~
Chi Dirac Equation	DCq.A~

C.11 Geodesics

Geodesic Equation	GEOq^m

C.12 Null Congruence

Congruence	KV
Null Congruence Condition	NCo
Geodesics Congruence Condition	GCo'a
Optical Scalars	
Congruence Expansion	thetaO
Congruence Squared Rotation	omegaSQO
Congruence Squared Shear	sigmaSQO

C.13 Kinematics

Velocity Vector	UV
Velocity	UU'a
Velocity Square	USQ
Projector	PR'a.b
Kinematics	
Acceleration	accU'a
Vorticity	omegaU.a.b
Volume Expansion	thetaU
Shear	sigmaU.a.b

C.14 Ideal and Spin Fluid

Pressure	PRES
Energy Density	ENER
Ideal Fluid Energy-Momentum Tensor	TIFL.a.b
Spin Fluid Energy-Momentum Tensor	TSFL.a.b
Spin Density	SPFLT.a.b
Spin Density 2-form	SPFL
Undotted Fluid Spin 3-form	SPFLU.AB
Frenkel Condition	FCo

C.15 Total Energy-Momentum and Spin

Total Energy-Momentum Tensor	TENMOM.a.b
Total Energy-Momentum Spinor	TENMOMS.AB.CD~
Total Energy-Momentum Trace	TENMOMT
Total Undotted Spin 3-form	SPINU.AB

C.16 Einstein Equations

Einstein Equation	EEq.a.b
Spinor Einstein Equations	
Traceless Einstein Equation	CEEq.AB.CD~
Trace of Einstein Equation	TEEq

C.17 Constants

A-Constants	ACONST.i2
L-Constants	LCONST.i6
M-Constants	MCONST.i3

C.18 Gravitational Equations

Action	LACT
Undotted Curvature Momentum	POMEGAU.AB
Torsion Momentum	PTHETA'a
Gravitational Equations	
Metric Equation	METRq.a.b
Torsion Equation	TORSq.AB

APPENDIX ***D***

Standard Synonymy

Below we present the default synonymy as it is defined in the global configuration file. See section **??** to find out how to change the default synonymy or define a new one.

```
Affine Aff
Anholonomic Nonholonomic AMode ABasis
Antisymmetric Asy
Change Transform
Classify Class
Components Comp
Connection Con
Constants Const Constant
Coordinates Cord
Curvature Cur
Dimension Dim
Dotted Do
Equation Equations Eq
Erase Delete Del
Evaluate Eval Simplify
Find F Calculate Calc
Form Forms
Functions Fun Function
Generic Gen
Gravitational Gravity Gravitation Grav
Holonomic HMode HBasis
Inverse Inv
Load Restore
Next N
Normalize Normal
Object Obj
Output Out
```

```
Parameter Par
Rotation Rot
Scalar Scal
Show ?
Signature Sig
Solutions Solution Sol
Spinor Spin Spinorial Sp
standardlisp lisp
Switch Sw
Symmetries Sym Symmetric
Tensor Tensors Tens
Torsion Tors
Transformation Trans
Undotted Un
Unload Save
Vector Vec
Write W
Zero Nullify
```