1.1 Introduction to package dgeom

Note: This package does not contain support for differential forms. The 'cartan' package in core Maxima provides some basic support for forms. This package should be considered an add-on to the 'ctensor' package in core Maxima, providing further functionality for some tensor calculations, as well as some standard differential geometry tools.

The `dgeom` package provides some basic functions useful for exploring metrics and manifolds. In particular coordinate transformations that allow one to define embedded manifolds in flat space. The package also provides more general tools for computing metrics after a coordinate transformation. This package was designed to be compatible with the `ctensor` package included with core Maxima.

Major capabilities of `dgeom` include:

- The `dgeom` package computes the frame field connections for an embedded surface using frames defined by the attitude matrix. See the function `dg_ffc()`.

- Computation a new metric from an old metric where the old metric is not δ (identity) or η (Minkowski). This is especially useful for stringing together several coordinate transformations when studying a metric.

- Computation of the Killing vector equations from the metric using `dg_kill()`.

The basic use of the package begins by defining the coordinates in the variables `cords_in`, `cords_tr`, and `cords_ot`. This can be done manually or by using the built-in function `dg_cords`.

In a typical session, one should first call `dg_cords(all)` to see the available coordinate systems and then choose one, or set up the coordinates by hand. Second, call `dg_metric()` to compute the metric in the new coordinates. `dgeom` can calculate the divergence, gradient, curl (`d=3` only), and Laplacian in any coordinate system you can define. It also defines them as functions you can then call if you want to use them in a computation. These computations are available in the functions `dg_diverg()`, `dg_grad()`, `dg_curl()`, and `dg_laplac()`.

Note: It helps to simplify the resulting formulas by restricting coordinate ranges. For example, if studying the sphere one should use `assume(theta>0,theta<%pi)` and `assume(r>0)`. These are set automatically in the package, but for coordinate systems input by hand one should consider setting the ranges manually.

The `dgeom` package is compatible with the `ctensor` package. Coordinates defined in `dgeom` can be used for computations in the `ctensor` package such as the Christoffel symbols, the Riemann tensor and so forth. See the function `set_ctensor_vars` for more information.

version: 0.1

Eric Majzoub, Jan 2026

1.2 Functions and Variables for dgeom

`cord_in`   [Variable]

Input coordinates are given as a list in the variable `cord_in`.

`cord_ot`   [Variable]

Output coordinates are given as a list in the variable `cord_ot`.

`cord_tr` [Variable]

Coordinate transformations are given as a list in the variable `cord_tr`. `cord_tr:[r*sin(theta)*cos(phi),r*sin(theta)*sin(phi),r*cos(theta)]`.

`dg_minkowski` [Variable]

The variable `dg_minkowski` (default value: 1) determines whether the metric in the output coordinates `cords_ot` is computed with a Minkowski signature. This variable is set automatically for the predefined coordinate transformations 'rindler'. If defining coordinates in Minkowski signature without using the function `dg_cords`, the timelike coordinate must be the first element in the list for both `cords_in` and `cords_ot`.

`dg_cords (coordinate system)` [Function]

The function `dg_cords` takes a predefined coordinate system name and sets the variables `cords_in`, `cords_tr`, `cords_ot`. `cords_in` are the input coordinates, `cords_tr` are the transformation functions to the new or output coordinates, and `cords_ot` are the output coordinates. To see the available predefined coordinate systems use `dg_cords(all)`.

```
(%i1) dg_cords(rindler);
(%o1)                              done
(%i2) show_cords();
```

$$cords\_in = [t, x]$$

$$cords\_tr = [sinh(omega)\ rho, cosh(omega)\ rho]$$

$$cords\_ot = [omega, rho]$$

$$dg\_minkowski = -1$$

```
(%o2)                              done
(%i3) dg_metric(init);
```

$$ds2\_in = del\ (x)^2 - del\ (t)^2$$

$$ds2 = del\ (rho)^2 - rho^2\ del\ (omega)^2$$

```
(%o3)                              done
```

`show_cords ( )` [Function]

The function `show_cords` takes no arguments. It displays the variables `cord_in`, `cord_tr`, `cord_ot`, and `dg_minkowski`.

`dg_derivs ( )` [Function]

This function takes no arguments, but the input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg‑cords], page 2. The function `dg_derivs` computes the partial derivative functions of the output coordinates in terms of the input coordinates using the transformation functions. For example, for input coordinates `[x,y,z]`, and output

coordinates [r,phi,z], and using the predefined cylindrical coordinate transformation xyz_to_cyl, one obtains the partial derivatives with respect to *(x,y,z)* in terms of the partial derivatives with respect to the variables *(r,phi,z)*.

The resulting partial derivative functions are named according to the input variable names, in the following notation. The partial derivative with respect to *x* is denoted d_dx, and likewise for the other input coordinates. The functions are defined with argument names generated by gensym.

```
(%i1) dg_cords(xyz_to_cyl);
(%o1)                                  done
(%i2) dg_derivs();
(%o2)                                  done
(%i3) fundef(d_dx);
                              dF              dF
(%o3)              d_dx(F) := ---- sin(phi) + -- cos(phi)
                              dphi            dr
```

dg_metric ([*init,cnvrt*])   [Function]
    This function takes as argument *init* or *cnvrt*. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2. The function dg_metric computes the metric in terms of the output coordinates cords_ot. If dg_metric is called with the argument *init* then the function assumes the input metric is diagonal with unit entries, although dg_minkowski may be set to -1 for the time component. If dg_metric is called with the argument *cnvrt*, then the function assumes there is an initial metric given in the matrix lg_in with the ordering of the entries in accordance with the ordering of the variables in cords_in. The value of dg_minkowski is ignored for this case. dg_metric returns the line element ds^2 in the variable ds2, and the metric in the matrix g.

```
(%i1) dg_cords(xyz_to_spher);
(%o1)                                  done
(%i2) dg_metric(init);
                             2        2        2
                 ds2_in = del (z) + del (y) + del (x)

             2    2            2        2   2              2
       ds2 = r  del (theta) + del (r) + r  sin (theta) del (phi)

(%o2)                                  done
```

The following example illustrates the conversion of the Rindler metric to new coordinates that mimic the Schwarzschild metric.

```
(%i1) cords_in:[omega,rho];
(%o1)                              [omega, rho]
(%i2) cords_ot:[t,xi];
(%o2)                                [t, xi]
(%i3) cords_tr:[t/2,2*sqrt(xi)];
                                      t
(%o3)                              [-, 2 sqrt(xi)]
                                      2
(%i4) lg_in:matrix([-rho^2,0],[0,1]);
                                   [      2     ]
(%o4)                              [ - rho    0 ]
                                   [            ]
                                   [   0      1 ]
(%i5) dg_metric(cnvrt);
                         2            2    2
             ds2_in = del (rho) - rho  del (omega)


                              2
                          del (xi)          2
                  ds2 = -------- - xi del (t)
                            xi


(%o5)                                done
```

dg_grad ( )   [Function]

This function takes no arguments. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2. The function `dg_grad` returns an expression for the gradient of a function F in the output coordinates `cord_ot`.

```
(%i1) dg_cords(xyz_to_spher);
(%o1)                               done
(%i2) dg_grad();
                           dF              dF
                           ---- e_phi    ------ e_theta
                           dphi          dtheta            dF
(%o2)         dg_gra(F) := ----------- + -------------- + -- e_r
                           r sin(theta)        r          dr
```

dg_curl ( )   [Function]

This function takes no arguments. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2. The curl function is limited to `d=3` dimensions.

The function `dg_curl` returns a function `dg_cur(W)` that takes vector argument W and returns the curl in terms of the output coordinates `cord_ot`.

```
(%i1) dg_cords(cart3d);
(%o1)                              done
(%i2) dg_curl();
```

$$ds2\_in = del^2(z) + del^2(y) + del^2(x)$$

$$ds2 = del^2(z) + del^2(y) + del^2(x)$$

$$\text{(\%o2)} \quad dg\_cur(W\_x, W\_y, W\_z) := \left[\frac{dW\_z}{dy} - \frac{dW\_y}{dz}, \frac{dW\_x}{dz} - \frac{dW\_z}{dx}, \frac{dW\_y}{dx} - \frac{dW\_x}{dy}\right]$$

```
(%i3) dg_cords(xyz_to_cyl);
(%o3)                              done
(%i4) dg_curl();
```

$$ds2\_in = del^2(z) + del^2(y) + del^2(x)$$

$$ds2 = del^2(z) + del^2(r) + r^2 \, del^2(phi)$$

$$\text{(\%o4)} \quad dg\_cur(W\_r, W\_phi, W\_z) := \left[\frac{\frac{dW\_z}{dphi}}{r} - \frac{dW\_phi}{dz}, \frac{dW\_r}{dz} - \frac{dW\_z}{dr},\right.$$

$$\left. - \frac{\frac{dW\_r}{dphi}}{r} + \frac{W\_phi}{r} + \frac{dW\_phi}{dr}\right]$$

dg_diverg ( ) [Function]

 This function takes no arguments. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2.

 The function `dg_diverg` returns a function `dg_div(W)` that takes argument W and returns the divergence in terms of the output coordinates `cord_ot`. The function `dg_metric` must be called before calling `dg_diverg`.

```
(%i1) dg_cords(xyz_to_cyl);
(%o1)                                done
(%i2) dg_metric(init);
                           2       2       2
              ds2_in = del (z) + del (y) + del (x)


                     2       2       2   2
              ds2 = del (z) + del (r) + r  del (phi)


(%o2)                                done
(%i3) dg_diverg();
                                    W_r   dW_z   dW_r   dW_phi
(%o3)        dg_div(W_r, W_phi, W_z) := --- + ---- + ---- + ------
                                     r     dz     dr     dphi
```

dg_laplac ( )   [Function]

This function takes no arguments. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2.

The function `dg_laplac` returns a function `dg_lap(W)` that takes arument `W` and returns the Laplacian in terms of the output coordinates `cord_ot`. The function `dg_metric` must be called before calling `dg_laplac`.

```
(%i1) dg_cords(xyz_to_cyl);
(%o1)                                done
(%i2) dg_metric(init);
                           2       2       2
              ds2_in = del (z) + del (y) + del (x)


                     2       2       2   2
              ds2 = del (z) + del (r) + r  del (phi)


(%o2)                                done
(%i3) dg_laplac();
                              2     2                2
                             d F   d F   2   dF      d F
                             (--- + ---) r  + -- r + -----
                              2     2         dr       2
                             dz    dr                dphi
(%o3)              dg_lap(F) := ----------------------------
                                            2
                                           r
```

dg_ffc ([*constraint equations*])   [Function]

Compute the frame field connections. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2.

The function `dg_ffc` computes the attitude matrix `A`, the connection coefficients in the matrix `Omega`, and the dual 1-forms in the array `thi`. Any constraint equations given are imposed after calculating the attitude matrix.

The structural equations are computed by first computing the matrices `TT[i,j]`: `thi[i] . thi[j]`, and `DD[i,j]: diff(Omega[i,j])`, and then computing `KK[i,j]` by solving *domega(i,j) = K th(i) * thi(j)*.

```
(%i1) dg_cords(xyz_to_cyl);
(%o1)                                   done
(%i2) dg_ffc([r = 1]);
(%o2)                                   done
(%i3) A;
                          [  cos(phi)    sin(phi)  0 ]
                          [                          ]
(%o3)                     [ - sin(phi)   cos(phi)  0 ]
                          [                          ]
                          [     0           0      1 ]
(%i4) Omega;
                          [    0         del(phi)  0 ]
                          [                          ]
(%o4)                     [ - del(phi)      0      0 ]
                          [                          ]
                          [    0            0      0 ]
(%i5) thi;
                               [   del(r)    ]
                               [             ]
(%o5)                          [ r del(phi)  ]
                               [             ]
                               [   del(z)    ]
```

`dg_kill` (*[none,show]*)   [Function]
    The function `dg_kill` computes the Killing equations. The metric must be computed with `dg_metric` before this function is called. If called with `dis=show` then the killing equations will be shown, otherwise they can be accessed in the array `killeq`.

```
(%i1) dg_cords(mink2d);
(%o1)                                done
(%i2) dg_metric(init);
```

$$ds2\_in = del^2(x) - del^2(t)$$

$$ds2 = del^2(x) - del^2(t)$$

```
(%o2)                                done
(%i3) dg_kill(show);
```

$$killeq_{1,\,1} = -\,2\left(\frac{d}{dt}(xi^t)\right)$$

$$killeq_{1,\,2} = \frac{d}{dt}(xi^x) - \frac{d}{dx}(xi^t)$$

$$killeq_{2,\,2} = 2\left(\frac{d}{dx}(xi^x)\right)$$

```
(%o3)                                done
```

get_ctensor_vars ( )   [Function]

This command is used to set the dgeom coordinate variables from those defined in the ctensor package. This command sets the dgeom coordinate variable cord_ot to the ctensor variable ct_cords. Then sets the lower indexed metric lg to the metric g. The ctensor package must be loaded and the metric computed for this command to work.

set_ctensor_vars ([*init*,*cnvrt*])   [Function]

This function takes as argument *init* or *cnvrt*. The input and output coordinates and the transformation functions must be defined beforehand for this function to work. See the function [dg_cords], page 2. See [dg_metric], page 3, for a description of the options *init* and *cnvrt*. This command is used to set up the calculation of the Christoffel symbols using the ctensor package using the coordinates and transformation functions from the dgeom package. The Christoffel symbols are the connections in the coordinate frame, as opposed to the frame-field connection coefficients computed using the function dg_ffc. The function set_ctensor_vars sets the following ctensor variables: cframe_flage:false, dim, and calls ct_coordsys(). The ctensor package must be loaded for this command to work.

Example: Compute the Christoffel symbols for flat space in polar coordinates.

```
(%i1) load(ctensor)$
(%i2) dg_cords(xy_to_polar);
(%o2)                                done
(%i3) set_ctensor_vars(init);
```

$$ds2\_in = del^2(y) + del^2(x)$$

$$ds2 = r^2\ del^2(theta) + del^2(r)$$

$$ctlist = [r\ cos(theta),\ r\ sin(theta),\ [r,\ theta]]$$

```
(%o3)                                done
(%i4) christof(mcs);
```

$$(\%t4) \qquad\qquad\qquad mcs^{\ \ \ 2}_{1,\ 2} = -\frac{1}{r}$$

$$(\%t5) \qquad\qquad\qquad mcs^{\ \ \ 1}_{2,\ 2} = -\ r$$

```
(%o5)                                done
```

Appendix A Function and Variable index