1.1 Introduction to package dgeom

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. But the package provides more general tools for computing metrics after a coordinate transformation.

The `dgeom` package computes the frame field connections for an embedded surface using frames defined by the attitude matrix.

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. Then one can call the divergence or Laplace functions `dg_diverg()` or `dg_laplac()` if desired. If the frame field connections are desired, call `dg_ffc()`.

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. See the function `set_ctensor_vars` for more information.

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`.

```
(%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();
                                 2           2    2
                         ds2 = del (rho) - rho  del (omega)

(%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_cords (*coordinate system*) [Function]

The function `dg_cords` takes a predefined coordinate system name and sets the variables `cord_in`, `cord_tr`, `cord,ot`. To see the available predefined coordinate systems use `dg_cords(all)`.

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 ( )` [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_metric` computes the metric in terms of the output coordinates `cords_ot`. This function assumes the input coordinates are flat. This function 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();
                   2    2              2       2    2            2
          ds2 = r  del (theta) + del (r) + r  sin (theta) del (phi)

(%o2)                                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
                 gra = ------------ + -------------- + -- e_r
                       r sin(theta)         r           dr

(%o2)                                done
```

`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 arument `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();
                          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();
                          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)   ]
```

`set_ctensor_vars ( )` [Function]

This command is used to set up the calculation of the Christoffel symbols using the `ctensor` 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);
(%i3) set_ctensor_vars();
                                   2    2              2
                            ds2 = r  del (theta) + del (r)

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

(%o3)                                done
(%i4) christof(mcs);
                                                   1
(%t4)                              mcs          = -
                                      1, 2, 2     r

(%t5)                             mcs          = - r
                                     2, 2, 1

(%o5)                                done
```

`get_ctensor_vars ( )` [Function]

This command sets the 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.

`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(xy_to_polar);
(%o1)                                done
(%i2) dg_metric();
                                   2    2              2
                            ds2 = r  del (theta) + del (r)

(%o2)                                done
(%i3) dg_kill(none);
(%o3)                                done
(%i4) killeq;
        [           d                   d          2     d           ]
        [        2 (-- (xi ))          (-- (xi )) r  + ------ (xi )  ]
        [           dr    1             dr    2        dtheta    1   ]
(%o4)   [                                                            ]
        [  d          2     d                d            2          ]
        [ (-- (xi )) r  + ------ (xi )  2 (------ (xi )) r  + 2 xi  r ]
        [  dr    2        dtheta    1      dtheta    2             1   ]
```

Appendix A Function and Variable index

D

C

G

S

D