Symbolic tensor calculus on manifolds: a SageMath implementation

CHAPTER 1

Introduction

Contents

1. What is tensor calculus on manifolds?

We shall provide precise definitions in Chaps. 2 and 3. Here, let us state briefly that *tensor calculus on manifolds* stands for calculus on vector fields, and more generally tensor fields, on differentiable manifolds, involving the following operations [12]:

- arithmetics of tensor fields;
- tensor product, contraction;
- (anti)symmetrization;
- Lie derivation along vector fields;
- pullback and pushforward associated with smooth manifold maps;
- exterior (Cartan) calculus on differential forms;
- covariant derivation with respect to a given affine connection;
- evaluating the torsion and the curvature of an affine connection.

Moreover, on pseudo-Riemannian manifolds, i.e. differentiable manifolds endowed with a metric tensor, we may add the following operations [11, 17]:

- musical isomorphisms (i.e. raising and lowering indices with the metric tensor);
- determining the Levi-Civita connection;
- evaluating the curvature tensor of the Levi-Civita connection (Riemann tensor);
- Hodge duality;
- computing geodesics.

2. A few words of history

Symbolic tensor calculus has a long history, which started almost as soon as computer algebra itself in the 1960s. Probably, the first tensor calculus program was GEOM, written by J.G. Fletcher in 1965 [7]. Its main capability was to compute the Riemann tensor of a given metric. In 1969, R.A. d'Inverno developed ALAM (for *Atlas Lisp Algebraic Manipulator*) and used it to compute the Riemann and Ricci tensors of the Bondi metric. According to [22], the original calculations took Bondi and collaborators 6 months to finish, while the computation with ALAM took 4 minutes and yielded the discovery of 6 errors in the original paper by Bondi et al. Since then, numerous packages have been developed; the reader is referred to [14] for a recent review of computer algebra systems for general relativity (see also [13] for a review up to 2002), and to [10, 4] for more recent reviews focused on tensor calculus. It is also worth to point out the extensive list of tensor calculus packages maintained by J. M. Martin-Garcia at `http://www.xact.es/links.html`.

3. Software for differential geometry

Software packages for differential geometry and tensor calculus can be classified in two categories:

(1) Applications atop some general purpose computer algebra system. Notable examples[1] are the xAct suite [15] and Ricci [20], both running atop Mathematica, DifferentialGeometry [1] integrated into Maple, GRTensorIII [8] atop Maple, Atlas 2 [2] for Mathematica and Maple, ctensor and itensor for Maxima [26] and SageManifolds [21] integrated in SageMath.
(2) Standalone applications. Recent examples are Cadabra [19] (field theory), SnapPy [6] (topology and geometry of 3-manifolds) and Redberry [5] (tensors); older examples can be found in Refs. [13, 14].

All applications listed in the second category are free software. In the first category, xAct and Ricci are also free software, but they require a proprietary product, the source code of which is closed (Mathematica).

As far as tensor calculus is concerned, the above packages can be distinguished by the type of computation that they perform: abstract calculus (xAct/xTensor, Ricci, itensor, Cadabra, Redberry), or component calculus (xAct/xCoba, DifferentialGeometry, GRTensorIII, Atlas 2, ctensor, SageManifolds). In the first category, tensor operations such as contraction or covariant differentiation are performed by manipulating the indices themselves rather than the components to which they correspond. In the second category, vector frames are explicitly introduced on the manifold and tensor operations are carried out on the components in a given frame.

4. A brief overview of SageMath

Since the tensor calculus method presented here is implemented in SageMath, we give first a brief overview of the latter.

SageMath[2] is a free, open-source mathematics software system, which is based on the Python programming language. It makes use of over 90 open-source packages, among which are Maxima, Pynac and SymPy (symbolic calculations), GAP (group theory), PARI/GP (number theory), Singular (polynomial computations), matplotlib (high quality 2D figures), and Jupyter (graphical interface). SageMath provides a uniform Python interface to all these packages; however, SageMath is much more than a mere interface: it contains a large and increasing part of original code (more than 750,000 lines of Python and Cython, involving 5344 classes). SageMath was created in 2005 by William Stein [24] and since then its development has been sustained by more than a hundred researchers (mostly mathematicians). In particular, a strong impulse is currently being provided by the European Horizon 2020 project OpenDreamKit [18]. Very good introductory textbooks about SageMath are [9, 27, 28, 3].

Apart from the syntax, which is based on a popular programming language (Python) and not on some custom script language, a difference between SageMath and, e.g., Maple or Mathematica is the use of the *parent/element pattern*. This pattern closely reflects actual mathematics. For instance, in Mathematica, all objects are trees of symbols and the program is essentially a set of sophisticated rules to manipulate symbols. On the contrary, in SageMath each object has a given type (i.e. is an instance of a given Python class[3]), and one distinguishes ***parent*** types, which model mathematical sets with some structure (e.g. algebraic structure), from ***element*** types, which model set elements. Moreover, each parent belongs to some dynamically generated class that encodes information about its *category*, in the mathematical sense of the word.[4] Automatic conversion rules, called ***coercions***, prior to a binary operation, e.g. $x + y$ with x and y having different parents, are implemented.

5. The purpose of this lecture

This lecture aims at presenting a *symbolic tensor calculus method* that

- runs on fully specified smooth manifolds (described by an atlas);

[1] See `https://en.wikipedia.org/wiki/Tensor_software` for more examples.

[2] `http://www.sagemath.org`

[3] Let us recall that within an object-oriented programming language (as Python), a ***class*** is a structure to declare and store the properties common to a set of objects. These properties are data (called ***attributes*** or ***state variables***) and functions acting on the data (called ***methods***). A specific realization of an object within a given class is called an ***instance*** of that class.

[4] See `http://doc.sagemath.org/html/en/reference/categories/sage/categories/primer.html` for a discussion of SageMath's category framework

- is not limited to a single coordinate chart or vector frame;
- runs even on non-parallelizable manifolds (i.e. manifolds that cannot be covered by a single vector frame);
- is independent of the symbolic backend (e.g., Pynac/Maxima, SymPy, ...) used to perform calculus at the level of coordinate expressions.

The aim is to present not only the main ideas of the method, but also some details of its implementation in SageMath. This implementation has been performed via the SageManifolds project:

`https://sagemanifolds.obspm.fr`,

the contributors to which are listed at

`https://sagemanifolds.obspm.fr/authors.html`.

CHAPTER 2

Differentiable manifolds

Contents

1. Introduction

Starting from basic mathematical definitions, we present the implementation of manifolds and coordinate charts in SageMath (Sec. 2). We then focus on the algebra of scalar fields on a manifold (3). As we shall see in Chap. 3, this algebra plays a central role in the implementation of vector fields, the latter being considered as forming a module over it.

2. Differentiable manifolds

2.1. **Topological manifolds.** Let $\mathbb{K}$ be a topological field. In most applications $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. Given an integer $n \geq 1$, a ***topological manifold of dimension*** n ***over*** $\mathbb{K}$ is a topological space M obeying the following properties:

(1) M is a ***separated space*** (also called ***Hausdorff space***): any two distinct points of M admit disjoint open neighbourhoods.
(2) M has a ***countable base***:[5] there exists a countable family $(U_k)_{k\in\mathbb{N}}$ of open sets of M such that any open set of M can be written as the union (possibly infinite) of some members of this family.
(3) Around each point of M, there exists a neighbourhood which is homeomorphic to an open subset of $\mathbb{K}^n$.

Property 1 excludes manifolds with "forks". Property 2 excludes "too large" manifolds; in particular it permits setting up the theory of integration on manifolds. In the case $\mathbb{K} = \mathbb{R}$, it also allows for a smooth manifold of dimension n to be embedded smoothly into the Euclidean space $\mathbb{R}^{2n}$ (Whitney theorem). Property 3 expresses the essence of a manifold: it means that, locally, M "resembles" $\mathbb{K}^n$.

Let us start to discuss the implementation of manifolds in SageMath. We shall do it on a concrete example, exposed in a Jupyter notebook which can be downloaded from the page devoted to these lectures:

`https://sagemanifolds.obspm.fr/jncf2018/`

As for all SageMath, the syntax used in this notebook is Python one. However, no a priori knowledge of Python is required, since we shall explain the main notations as they appear. The first cell of the Jupyter notebook is to have all outputs rendered with LaTeX:

```
In [1]: %display latex
```

In SageMath, manifolds are constructed by means of the global function `Manifold`:

```
In [2]: M = Manifold(2, 'M')
        print(M)
```

```
2-dimensional differentiable manifold M
```

[5] In the language of topology, one says that M is a *second-countable space*.

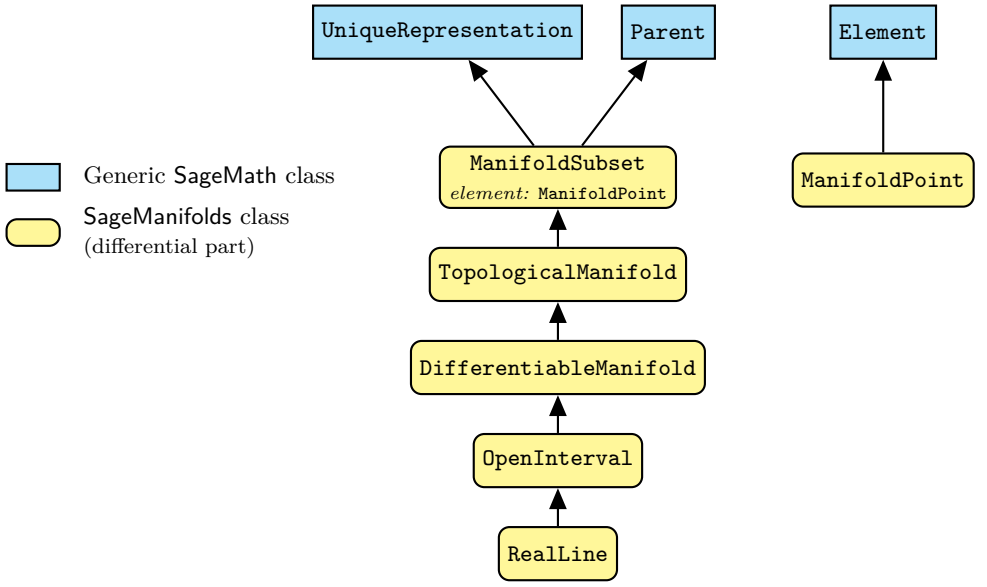


Figure 2.1: Python classes for topological manifolds, differentiable manifolds, subsets of them and points on them (`ManifoldPoint`).

By default, the function `Manifold` returns a manifold over $\mathbb{K} = \mathbb{R}$:

```
In [3]: M.base_field()
```

Out[3]: $\mathbf{R}$

Note the use of the standard object-oriented notation (ubiquitous in Python): the method `base_field()` is called on the object `M`; since this method does not require any extra argument (all the information lies in `M`), its argument list is empty, hence the final `()`. Base fields different from $\mathbb{R}$ must be specified with the optional keyword `field`, for instance

```
M = Manifold(2, 'M', field='complex')
```

to construct a complex manifold[6]. We may check that M is a topological space:

```
In [4]: M in Sets().Topological()
```

```
Out[4]: True
```

Actually, M belongs to the following categories:

```
In [5]: M.categories()
```

Out[5]: $[\mathbf{Smooth_R}, \mathbf{Differentiable_R}, \mathbf{Manifolds_R}, \mathbf{TopologicalSpaces}(\mathbf{Sets}), \mathbf{Sets}, \mathbf{SetsWithPartialMaps}, \mathbf{Objects}]$

As we can see from the first category in the above list, `Manifold` constructs a smooth manifold by default. If one would like to stick to the topological level, one should add the keyword argument `structure='topological'` to `Manifold`, i.e.

```
M = Manifold(2, 'M', structure='topological')
```

Then M would have been a topological manifold without any further structure.

Manifolds are implemented by the Python classes `TopologicalManifold` and `DifferentiableManifold` (see Fig. 2.1), actually by dynamically generated subclasses of those, via SageMath category framework:[7]

[6] Note however that the functionalities regarding complex manifolds are pretty limited at the moment. Volunteers are welcome to implement them! See `https://sagemanifolds.obspm.fr/contrib.html`.

[7] See `http://doc.sagemath.org/html/en/reference/categories/sage/categories/primer.html` for details.

```
In [6]: type(M)
```

```
Out[6]: <class 'sage.manifolds.differentiable.manifold.
                DifferentiableManifold_with_category'>
```

Let us check that the actual class of `M`, i.e. `DifferentiableManifold-with-category`, is a subclass of `DifferentiableManifold`:

```
In [7]: isinstance(M,
                   sage.manifolds.differentiable.manifold.DifferentiableManifold)
```

```
Out[7]: True
```

and hence of `TopologicalManifold` according to the inheritance diagram of Fig. 2.1:

```
In [8]: isinstance(M, sage.manifolds.manifold.TopologicalManifold)
```

```
Out[8]: True
```

Notice from Fig. 2.1 that `TopologicalManifold` itself is a subclass of `ManifoldSubset` (the class for generic subsets of a manifold), which reflects the fact that $M \subset M$.

2.2. **Coordinate charts.** Property 3 in the definition of a topological manifold (Sec. 2.1) means that one can label the points of M in a continuous way by n numbers $(x^\alpha)_{\alpha\in\{0,\ldots,n-1\}} \in \mathbb{K}^n$, which are called ***coordinates***. More precisely, given an open subset $U \subset M$, a ***coordinate chart*** (or simply a ***chart***) on U is a homeomorphism[8]

$$\begin{array}{rccc} X: & U\subset M & \longrightarrow & X(U)\subset \mathbb{K}^n \\ & p & \longmapsto & (x^0,\ldots,x^{n-1}). \end{array} \tag{2.1}$$

We declare a chart, along with the symbols used to denote the coordinates (here $x = x^0$ and $y = x^1$) by

```
In [9]: U = M.open_subset('U')
        XU.<x,y> = U.chart()
        XU
```

```
Out[9]: (U, (x, y))
```

Open subsets of a differentiable manifold are implemented by a (dynamically generated) subclass of `DifferentiableManifold`, since they are differentiable manifolds in their own:

```
In [10]: isinstance(U,
                    sage.manifolds.differentiable.manifold.DifferentiableManifold)
```

```
Out[10]: True
```

Points on M are created from their coordinates in a given chart:

```
In [11]: p = U((1,2), chart=XU, name='p')
         print(p)
```

```
Point p on the 2-dimensional differentiable manifold M
```

The syntax `U(...)` used to create p as an element of U reflects the parent/element pattern employed in SageMath; indeed U is the parent of p:

```
In [12]: p.parent()
```

[8] Let us recall that a ***homeomorphism*** between two topological spaces (here U and $X(U)$) is a bijective map X such that both X and X^{-1} are continuous.

```
Out[12]: U
```

Points are implemented by a dynamically generated subclass of `ManifoldPoint` (cf. Fig. 2.1). The principal attribute of this class is the one storing the point's coordinates in various charts; it is implemented as a Python dictionary,[9] whose keys are the charts:

```
In [13]: p._coordinates
```

```
Out[13]: {(U, (x, y)) : (1, 2)}
```

The leading underscore in the name `_coordinates` is a notation convention to specify that this attribute is a ***private*** one: the dictionary `_coordinates` should not be manipulated by the end user or involved in some code outside of the class `ManifoldPoint`. It belongs to the internal implementation, which may be changed while the user interface of the class `ManifoldPoint` is kept fixed. We show this private attribute here because we are precisely interested in implementation features. The public way to recover the point's coordinates is to let the chart act on the point (reflecting thereby the definition (2.1) of a chart):

```
In [14]: XU(p)
```

```
Out[14]: (1, 2)
```

Usually, one needs more than a single coordinate system to cover M. An ***atlas*** on M is a set of pairs $(U_i, X_i)_{i\in I}$, where I is a set, U_i an open set of M and X_i a chart on U_i, such that the union of all U_i's covers M:

$$\bigcup_{i\in I} U_i = M. \tag{2.2}$$

Here we introduce a second chart on M:

```
In [15]: V = M.open_subset('V')
         XV.<xp,yp> = V.chart("xp:x' yp:y'")
         XV
```

```
Out[15]: (V, (x', y'))
```

and declare that M is covered by only two charts, i.e. that $M = U \cup V$:

```
In [16]: M.declare_union(U, V)
```

```
In [17]: M.atlas()
```

```
Out[17]: [(U, (x, y)), (V, (x', y'))]
```

2.3. **Smooth manifolds.** For manifolds, the concept of differentiability is defined from the smooth structure of $\mathbb{K}^n$, via an atlas: a ***smooth manifold***, is a topological manifold M equipped with an atlas $(U_i, X_i)_{i\in I}$ such that for any non-empty intersection $U_i \cap U_j$, the map

$$X_i \circ X_j^{-1} : X_j(U_i \cap U_j) \subset \mathbb{K}^n \longrightarrow X_i(U_i \cap U_j) \subset \mathbb{K}^n \tag{2.3}$$

is smooth (i.e. C^∞). Note that the above map is from an open set of $\mathbb{K}^n$ to an open set of $\mathbb{K}^n$, so that the invoked differentiability is nothing but that of $\mathbb{K}^n$. Such a map is called a ***change***

[9] A ***dictionary***, also known as ***associative array***, is a data structure that generalizes the concept of array in the sense that the key to access to an element is not restricted to an integer or a tuple of integers.

of coordinates or, in the mathematical literature, a ***transition map***. The atlas $(U_i, X_i)_{i\in I}$ is called a ***smooth atlas***.

Remark 1: Strictly speaking a smooth manifold is a pair $(M, \mathcal{A})$ where $\mathcal{A}$ is a (maximal) smooth atlas on M. Indeed a given topological manifold M can have non-equivalent differentiable structures, as shown by Milnor (1956) [16] in the specific case of the unit sphere of dimension 7, $\mathbb{S}^7$: there exist smooth manifolds, the so-called *exotic spheres*, that are homeomorphic to $\mathbb{S}^7$ but not diffeomorphic to $\mathbb{S}^7$. On the other side, for $n \leq 6$, there is a unique smooth structure for the sphere $\mathbb{S}^n$. Moreover, any manifold of dimension $n \leq 3$ admits a unique smooth structure. Amazingly, in the case of $\mathbb{R}^n$, there exists a unique smooth structure (the standard one) for any $n \neq 4$, but for $n = 4$ (the spacetime case!) there exist uncountably many non-equivalent smooth structures, the so-called *exotic* $\mathbb{R}^4$ [25].

For the manifold M under consideration, we define the transition map `XU` $\to$ `XV` on $W = U \cap V$ as follows:

In [18]:
```
XU_to_XV = XU.transition_map(XV,
                             (x/(x^2+y^2), y/(x^2+y^2)),
                             intersection_name='W',
                             restrictions1= x^2+y^2!=0,
                             restrictions2= xp^2+yp^2!=0)
XU_to_XV.display()
```

Out[18]:
$$\begin{cases} x' = \frac{x}{x^2+y^2} \\ y' = \frac{y}{x^2+y^2} \end{cases}$$

The argument `restrictions1` means that $W = U \setminus \{S\}$, where S is the point of coordinates $(x, y) = (0, 0)$, while the argument `restrictions2` means that $W = V \setminus \{N\}$, where N is the point of coordinates $(x', y') = (0, 0)$. Since $M = U \cup V$, we have then

$$U = M \setminus \{N\}, \qquad V = M \setminus \{S\}, \quad \text{and} \quad W = M \setminus \{N, S\}. \tag{2.4}$$

The transition map `XV` $\to$ `XU` is obtained by computing the inverse of the one defined above:

In [19]:
```
XU_to_XV.inverse().display()
```

Out[19]:
$$\begin{cases} x = \frac{x'}{x'^2+y'^2} \\ y = \frac{y'}{x'^2+y'^2} \end{cases}$$

At this stage, the smooth manifold M is fully specified, being covered by one atlas with all transition maps specified. The reader may have recognized that M is nothing but the 2-dimensional sphere:

$$M = \mathbb{S}^2, \tag{2.5}$$

with `XU` (resp. `XV`) being the chart of ***stereographic coordinates*** from the North pole N (resp. the South pole S).

Since the transition maps have been defined, we can ask for the coordinates (x', y') of the point p, whose (x, y) coordinates were $(1, 2)$:

In [20]:
```
XV(p)
```

Out[20]:
$$\left(\frac{1}{5}, \frac{2}{5}\right)$$

This operation has updated the internal dictionary `_coordinates` (compare with `Out [13]`):

In [21]:
```
p._coordinates
```

Out[21]:
$$\left\{(U, (x, y)) : (1, 2), (V, (x', y')) : \left(\frac{1}{5}, \frac{2}{5}\right)\right\}$$

2.4. **Smooth maps.** Given two smooth manifolds, M and M', of respective dimensions n and n', we say that a map $\Phi : M \to M'$ is ***smooth map*** if and only if in some (and hence all, thanks to the smoothness of (2.3)) coordinate systems of M and M' belonging to the smooth atlases of M and M', the coordinates of the image $\Phi(p)$ of any point $p \in M$ are smooth functions $\mathbb{K}^n \to \mathbb{K}^{n'}$ of the coordinates of p. The map Φ is said to be a ***diffeomorphism*** iff it is bijective and both Φ and Φ^{-1} are smooth. This implies $n = n'$.

Back to our example manifold, a natural smooth map is the embedding of $\mathbb{S}^2$ in $\mathbb{R}^3$. To define it, we start by declaring $\mathbb{R}^3$ as a 3-dimensional smooth manifold, canonically endowed with a single chart, that of Cartesian coordinates (X, Y, Z):

In [22]:
```
R3 = Manifold(3, 'R^3', r'\mathbb{R}^3')
XR3.<X,Y,Z> = R3.chart()
XR3
```

Out[22]: $\left(\mathbb{R}^3, (X, Y, Z)\right)$

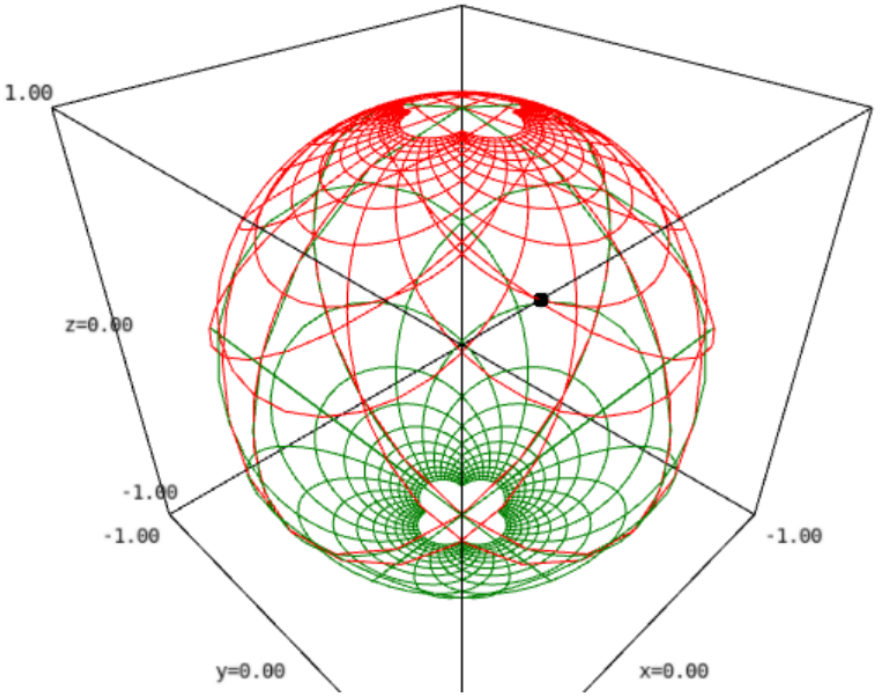
The embedding $\Phi : \mathbb{S}^2 \to \mathbb{R}^3$ is then defined in terms of its coordinate expression in the two charts covering $M = \mathbb{S}^2$:

In [23]:
```
Phi = M.diff_map(R3, {(XU, XR3):
                        [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2),
                         (x^2+y^2-1)/(1+x^2+y^2)],
                      (XV, XR3):
                        [2*xp/(1+xp^2+yp^2), 2*yp/(1+xp^2+yp^2),
                         (1-xp^2-yp^2)/(1+xp^2+yp^2)]},
                     name='Phi', latex_name=r'\Phi')
Phi.display()
```

Out[23]:
$$\begin{array}{lccl} \Phi: & M & \longrightarrow & \mathbb{R}^3 \\ \text{on } U: & (x, y) & \longmapsto & (X, Y, Z) = \left(\frac{2\,x}{x^2+y^2+1}, \frac{2\,y}{x^2+y^2+1}, \frac{x^2+y^2-1}{x^2+y^2+1}\right) \\ \text{on } V: & (x', y') & \longmapsto & (X, Y, Z) = \left(\frac{2\,x'}{x'^2+y'^2+1}, \frac{2\,y'}{x'^2+y'^2+1}, -\frac{x'^2+y'^2-1}{x'^2+y'^2+1}\right) \end{array}$$

We may use Φ for graphical purposes, for instance to display the grids of the stereographic charts `XU` (in red) and `XV` (in green), with the point p atop:

In [24]:
```
graph = XU.plot(chart=XR3, mapping=Phi, number_values=25,
                label_axes=False) + \
        XV.plot(chart=XR3, mapping=Phi, number_values=25,
                color='green', label_axes=False) + \
        p.plot(chart=XR3, mapping=Phi, label_offset=0.05)
show(graph, viewer='threejs', online=True)
```



3. Scalar fields and their algebra

3.1. **Definition and implementation.** Given a smooth manifold M over a topological field $\mathbb{K}$, a ***scalar field*** (also called a ***scalar-valued function***) on M is a smooth map

$$\begin{array}{rccc} f: & M & \longrightarrow & \mathbb{K} \\ & p & \longmapsto & f(p). \end{array} \tag{3.1}$$

A scalar field has different coordinate representations F, $\hat{F}$, etc. in different charts X, $\hat{X}$, etc. defined on M:

$$f(p) = F(\underbrace{x^1, \ldots, x^n}_{\substack{\text{coord. of } p \\ \text{in chart } X}}) = \hat{F}(\underbrace{\hat{x}^1, \ldots, \hat{x}^n}_{\substack{\text{coord. of } p \\ \text{in chart } \hat{X}}}) = \ldots \tag{3.2}$$

In SageMath, scalar fields are implemented by the class `DiffScalarField`[10] and the various representations (3.2) are stored in the private attribute `_express` of this class, which is a Python dictionary whose keys are the various charts defined on M:

$$f.\texttt{\_express} = \left\{X : F,\ \hat{X} : \hat{F}, \ldots\right\}. \tag{3.3}$$

Each representation F is an instance of the class `ChartFunction`, devoted to functions of coordinates, allowing for different internal representations: SageMath symbolic expression, SymPy expression, etc.

For instance, let us define a scalar field on our example manifold $M = \mathbb{S}^2$:

In [25]:
```
f = M.scalar_field({XU: 1/(1+x^2+y^2), XV: (xp^2+yp^2)/(1+xp^2+yp^2)},
                   name='f')
f.display()
```

Out[25]:
$$\begin{array}{rccc} f: & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x, y) & \longmapsto & \frac{1}{x^2+y^2+1} \\ \text{on } V: & (x', y') & \longmapsto & \frac{x'^2+y'^2}{x'^2+y'^2+1} \end{array}$$

The internal dictionary `_express` is then

[10] http://doc.sagemath.org/html/en/reference/manifolds/sage/manifolds/differentiable/scalarfield.html

In [26]:
```
f._express
```

Out[26]: $\left\{(U, (x, y)) : \dfrac{1}{x^2+y^2+1}, (V, (x', y')) : \dfrac{x'^2+y'^2}{x'^2+y'^2+1}\right\}$

The reader may wonder about the compatibility of the two coordinate expressions provided in the definition of f. Actually, to ensure the compatibility, it is possible to declare the scalar field in a single chart, `XU` say, and then to obtain its expression in chart `XV` by analytic continuation from the expression in $W = U \cap V$, where both expressions are known, thanks to the transition map `XV` $\to$ `XU`:

In [27]:
```
f0 = M.scalar_field({XU: 1/(1+x^2+y^2)})
f0.add_expr_by_continuation(XV, U.intersection(V))
f == f0
```

Out[27]: `True`

The representation of the scalar field in a given chart, i.e. the public access to the private directory `_express`, is obtained via the method `coord_function()`:

In [28]:
```
fU = f.coord_function(XU)
fU.display()
```

Out[28]: $(x, y) \mapsto \dfrac{1}{x^2+y^2+1}$

In [29]:
```
fV = f.coord_function(XV)
fV.display()
```

Out[29]: $(x', y') \mapsto \dfrac{x'^2+y'^2}{x'^2+y'^2+1}$

As mentioned above, each chart representation is an instance of the class `ChartFunction`:

In [30]:
```
isinstance(fU, sage.manifolds.chart_func.ChartFunction)
```

Out[30]: `True`

Mathematically, ***chart functions*** are $\mathbb{K}$-valued functions on the codomain of the considered chart. They map coordinates to elements of the base field $\mathbb{K}$:

In [31]:
```
fU(1,2)
```

Out[31]: $\dfrac{1}{6}$

In [32]:
```
fU(*XU(p))
```

Out[32]: $\dfrac{1}{6}$

Note the use of Python's star operator in `*XU(p)` to unpack the tuple of coordinates returned by `XU(p)` (in the present case: `(1,2)`) to positional arguments for the function `fU` (in the present case: `1, 2`). On their side, scalar fields map *manifold points*, not coordinates, to $\mathbb{K}$:

In [33]:
```
f(p)
```

Out[33]: $\dfrac{1}{6}$

Note that the equality between `Out[32]` and `Out[33]` reflects the identity $f = F \circ X$, where

F is the chart function (denoted `fU` above) representing the scalar field f on the chart X (cf. Eq. (3.2)).

Internally, each chart function stores coordinate expressions with respect to various computational backends:

- SageMath symbolic engine, based on the Pynac[11] backend, with Maxima used for some simplifications or computation of integrals;
- SymPy[12] (Python library for symbolic mathematics);
- in the future, more symbolic or numerical backends will be implemented.

The coordinate expressions are stored in the private dictionary `_express`[13] of the class `ChartFunction`, whose keys are strings identifying the computational backends. By default only SageMath symbolic expressions, i.e. expressions pertaining to the so-called SageMath's Symbolic Ring (`SR`), are stored:

```
In [34]: fU._express
```

Out[34]: $\left\{\mathtt{SR} : \dfrac{1}{x^2+y^2+1}\right\}$

The public access to the private dictionary `_express` is performed via the method `expr()`:

```
In [35]: fU.expr()
```

Out[35]: $\dfrac{1}{x^2+y^2+1}$

```
In [36]: type(fU.expr())
```

```
Out[36]: <type 'sage.symbolic.expression.Expression'>
```

Actually, `fU.expr()` is a shortcut for `fU.expr('SR')` since `SR` is the default symbolic backend. Note that the class `Expression` is that devoted to SageMath symbolic expressions. The method `expr()` can also be invoked to get the expression in another symbolic backend, for instance SymPy:

```
In [37]: fU.expr('sympy')
```

```
Out[37]: 1/(x**2 + y**2 + 1)
```

```
In [38]: type(fU.expr('sympy'))
```

```
Out[38]: <class 'sympy.core.power.Pow'>
```

This operation has updated the internal dictionary `_express` (compare with `Out [34]`):

```
In [39]: fU._express
```

Out[39]: $\left\{\mathtt{SR} : \dfrac{1}{x^2+y^2+1}, \mathtt{sympy} : \mathtt{1/(x**2 + y**2 + 1)}\right\}$

The default calculus backend for chart functions of chart `XU` can changed thanks to the method `set_calculus_method()`:

[11] `http://pynac.org`
[12] `https://www.sympy.org`
[13] not to be confused with the attribute `_express` of class `DiffScalarField` presented at `In [26]`

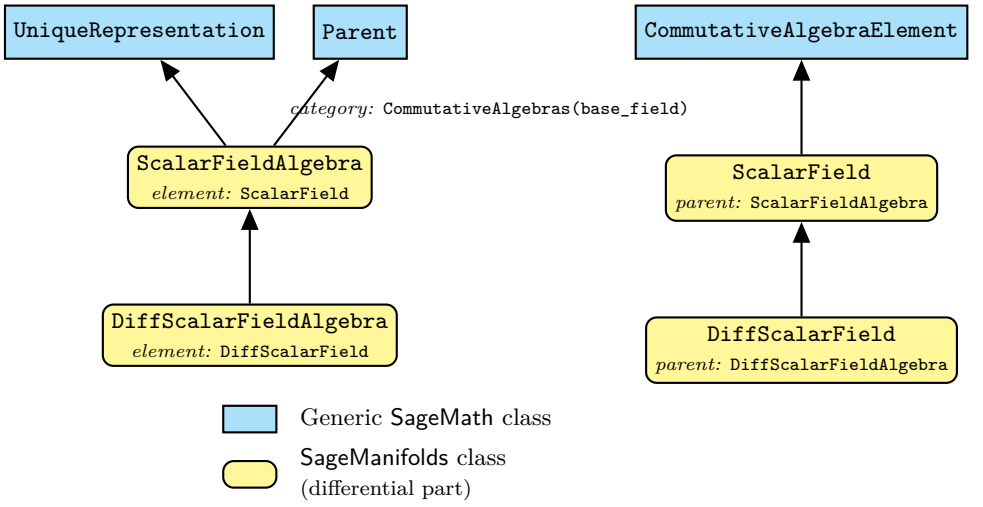


Figure 3.1: SageMath classes for scalar fields on a manifold.

```
In [40]: XU.set_calculus_method('sympy')
         fU.expr()
```

```
Out[40]: 1/(x**2 + y**2 + 1)
```

Reverting to SageMath's symbolic engine:

```
In [41]: XU.set_calculus_method('SR')
         fU.expr()
```

Out[41]: $\dfrac{1}{x^2+y^2+1}$

Symbolic expressions can be accessed directly from the scalar field, `f.expr(XU)` being a shortcut for `f.coord_function(XU).expr()`:

```
In [42]: f.expr(XU)
```

Out[42]: $\dfrac{1}{x^2+y^2+1}$

```
In [43]: f.expr(XV)
```

Out[43]: $\dfrac{x'^2+y'^2}{x'^2+y'^2+1}$

3.2. **Scalar field algebra.** The set $C^\infty(M)$ of all scalar fields on M has naturally the structure of a commutative algebra over $\mathbb{K}$: it is clearly a vector space over $\mathbb{K}$ and it is endowed with a commutative ring structure by pointwise multiplication:

$$\forall f, g \in C^\infty(M), \quad \forall p \in M, \quad (f.g)(p) := f(p)g(p). \tag{3.4}$$

The algebra $C^\infty(M)$ is implemented in SageMath via the parent class `DiffScalarFieldAlgebra`,[14] in the category `CommutativeAlgebras`. The corresponding element class is of course `DiffScalarField` (cf. Fig. 3.1).

The SageMath object representing $C^\infty(M)$ is obtained from `M` via the method `scalar_field_algebra()`:

[14] http://doc.sagemath.org/html/en/reference/manifolds/sage/manifolds/differentiable/scalarfield_algebra.html

In [44]:
```
CM = M.scalar_field_algebra()
CM
```

Out[44]: $C^\infty(M)$

In [45]:
```
CM.category()
```

Out[45]: $\mathbf{CommutativeAlgebras}_{\mathrm{SR}}$

As for the manifold classes, the actual Python class implementing $C^\infty(M)$ is inherited from `DiffScalarFieldAlgebra` via SageMath's category framework (cf. Sec. 2.1), hence it bares the name `DiffScalarFieldAlgebra_with_category`:

In [46]:
```
type(CM)
```

Out[46]:
```
<class 'sage.manifolds.differentiable.scalarfield_algebra.
        DiffScalarFieldAlgebra_with_category'>
```

The class `DiffScalarFieldAlgebra_with_category` is dynamically generated as a subclass of `DiffScalarFieldAlgebra` with extra functionalities, like for instance the method `is_commutative()`:

In [47]:
```
CM.is_commutative()
```

Out[47]: True

To have a look at the corresponding code, we use the double question mark, owing to the fact that SageMath is open-source:

In [48]:
```
CM.is_commutative??
```

```
1  def is_commutative(self):
2      """
3      Return ``True``, since commutative magmas are commutative.
4
5      EXAMPLES::
6
7          sage: Parent(QQ,category=CommutativeRings()).is_commutative()
8          True
9      """
10     return True
11 File: .../local/lib/python2.7/site-packages/sage/categories/magmas.py
```

We see from the `File` field in line 11 that the code belongs to the category part of SageMath, not to the manifold part, where the class `DiffScalarFieldAlgebra` is defined. This shows that the method `is_commutative()` has indeed be added to the methods of the base class `DiffScalarFieldAlgebra`, while dynamically generating the class `DiffScalarFieldAlgebra-with-category`.

Regarding the scalar field `f` introduced in Sec. 3.1, we have of course

In [49]:
```
f in CM
```

Out[49]: True

Actually, in SageMath language, `CM`=$C^\infty(M)$ is the parent of `f`:

In [50]:
```
f.parent() is CM
```

```
Out[50]: True
```

The zero element of the algebra $C^\infty(M)$ is

```
In [51]: CM.zero().display()
```

$$\begin{array}{lccc} 0: & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x,y) & \longmapsto & 0 \\ \text{on } V: & (x',y') & \longmapsto & 0 \end{array}$$

Out[51]:

while its unit element is

```
In [52]: CM.one().display()
```

$$\begin{array}{lccc} 1: & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x,y) & \longmapsto & 1 \\ \text{on } V: & (x',y') & \longmapsto & 1 \end{array}$$

Out[52]:

3.3. **Implementation of algebra operations.** Let us consider some operation in the algebra $C^\infty(M)$:

```
In [53]: h = f + 2*CM.one()
         h.display()
```

$$\begin{array}{lccc} & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x,y) & \longmapsto & \frac{2\,x^2+2\,y^2+3}{x^2+y^2+1} \\ \text{on } V: & (x',y') & \longmapsto & \frac{3\,x'^2+3\,y'^2+2}{x'^2+y'^2+1} \end{array}$$

Out[53]:

```
In [54]: h(p)
```

Out[54]: $\frac{13}{6}$

Let us examine how the addition in `In [53]` is performed. For the Python interpreter `h = f + 2*CM.one()` is equivalent to `h = f.__add__(2*CM.one())`, i.e. the `+` operator amounts to calling the method `__add__()` on its left operand, with the right operand as argument. To have a look at the source code of this method, we use the double question mark:[15]

```
In [55]: f.__add__??
```

```
File: .../src/sage/structure/element.pyx
def __add__(left, right):
    """
    Top-level addition operator for :class:`Element` invoking
    the coercion model.

    See :ref:`element_arithmetic`.
    ...
    """
    cdef int cl = classify_elements(left, right)
    if HAVE_SAME_PARENT(cl):
        return (<Element>left)._add_(right)
    # Left and right are Sage elements => use coercion model
    if BOTH_ARE_ELEMENT(cl):
```

[15] In this transcript of code and in those that follow, some parts have been skipped, being not relevant for the discussion; they are marked by "...".

```
15          return coercion_model.bin_op(left, right, add)
16      ...
```

From lines 1 and 4, we see that the method `__add__()` is implemented at the level of the class `Element` from which `DiffScalarField` inherits, via `CommutativeAlgebraElement` (cf. Fig. 3.1). In the present case, `left` = `f` and `right` = `2*CM.one()` have the same parent, namely the algebra `CM`, so that the actual result is computed in line 12. The latter invokes the method `_add_()` (note the single underscore on each side of `add`). This operator is implemented at the level of `ScalarField`, as checked from the source code (see line 24 below):

In [56]:
```
f._add_??
```

```
def _add_(self, other):
    """
    Scalar field addition.

    INPUT:
    - ``other`` -- a scalar field (in the same algebra as ``self``)

    OUTPUT:
    - the scalar field resulting from the addition of ``self`` and
      ``other``
    ...
    """
    ...
    # Generic case:
    com_charts = self.common_charts(other)
    if com_charts is None:
        raise ValueError("no common chart for the addition")
    result = type(self)(self.parent())
    for chart in com_charts:
        # ChartFunction addition:
        result._express[chart] = self._express[chart] + other._express[chart]
    ...
    return result
File: .../local/lib/python2.7/site-packages/sage/manifolds/scalarfield.py
```

This reflects a general strategy[16] in SageMath: the arithmetic Python operators `__add__()`, `__sub__()`, etc. are implemented at the top-level class `Element`, while specific element subclasses, like `ScalarField` here, implement single-underscore methods `_add_()`, `_sub_()`, etc., which perform the actual computation when both operands have the same parent. Looking at the code (lines 15 to 23), we notice that the first step is to search for the charts in which both operands of the addition operator have a coordinate expression (line 15). This is performed by the method `common_charts()`; in the current example, we get the two stereographic charts defined on M:

In [57]:
```
f.common_charts(2*CM.one())
```

Out[57]: $[(U, (x, y)), (V, (x', y'))]$

In general, `common_charts()` returns the charts for which both operands have already a known coordinate expression or for which a coordinate expression can be computed by a known transition map, as we can see on the source code:

In [58]:
```
f.common_charts??
```

```
def common_charts(self, other):
    """
    Find common charts for the expressions of the scalar field and
    ``other``.

    INPUT:
```

[16] See `http://doc.sagemath.org/html/en/thematic_tutorials/coercion_and_categories.html` for details.

```
    - ``other`` -- a scalar field

    OUTPUT:
    - list of common charts; if no common chart is found, ``None`` is
      returned (instead of an empty list)
    ...
    """
    if not isinstance(other, ScalarField):
        raise TypeError("the second argument must be a scalar field")
    coord_changes = self._manifold._coord_changes
    resu = []
    #
    # 1/ Search for common charts among the existing expressions, i.e.
    #    without performing any expression transformation.
    #    ----------------------------------------------------------------
    for chart1 in self._express:
        if chart1 in other._express:
            resu.append(chart1)
    # Search for a subchart:
    known_expr1 = self._express.copy()
    known_expr2 = other._express.copy()
    for chart1 in known_expr1:
        if chart1 not in resu:
            for chart2 in known_expr2:
                if chart2 not in resu:
                    if chart2 in chart1._subcharts:
                        self.expr(chart2)
                        resu.append(chart2)
                    if chart1 in chart2._subcharts:
                        other.expr(chart1)
                        resu.append(chart1)
    #
    # 2/ Search for common charts via one expression transformation
    #    ------------------------------------------------------------
    for chart1 in known_expr1:
        if chart1 not in resu:
            for chart2 in known_expr2:
                if chart2 not in resu:
                    if (chart1, chart2) in coord_changes:
                        self.coord_function(chart2, from_chart=chart1)
                        resu.append(chart2)
                    if (chart2, chart1) in coord_changes:
                        other.coord_function(chart1, from_chart=chart2)
                        resu.append(chart1)
    if resu == []:
        return None
    else:
        return resu
File: .../local/lib/python2.7/site-packages/sage/manifolds/scalarfield.py
```

Once the list of charts in which both operands have a coordinate expression has been found, the addition is performed at the chart function level (cf. Sec. 3.1), via the loop on the charts in lines 19-21 of the code for `_add_()`. The code for the addition of chart functions defined on the same chart is (recall that `fU` is the chart function representing f in chart `XU`):

```
In [59]: fU._add_??
```

```
def _add_(self, other):
    """
    Addition operator.

    INPUT:
```

```
6       - ``other`` -- a :class:`ChartFunction` or a value
7
8       OUTPUT:
9       - chart function resulting from the addition of ``self``
10        and ``other``
11      ...
12      """
13      curr = self._calc_method._current
14      res = self._simplify(self.expr() + other.expr())
15      if curr =='SR' and res.is_trivial_zero():
16          # NB: "if res == 0" would be too expensive (cf. #22859)
17          return self.parent().zero()
18      else:
19          return type(self)(self.parent(), res)
20  File: .../local/lib/python2.7/site-packages/sage/manifolds/chart_func.py
```

We notice that the addition is performed in line 14 on the symbolic expression with respect to the symbolic backend currently at work (SageMath/Pynac, SymPy, ...), as returned by the method `expr()` (see Sec. 3.1). Let us recall that the user can change the symbolic backend at any time by means of the method `set_calculus_method()`, applied either to a chart or to an open subset (possibly `M` itself). Besides, we notice on line 14 above that the result of the symbolic addition is automatically simplified, by means of the method `_simplify`. The latter invokes a chain of simplifying functions, which depends on the symbolic backend.[17]

Let us now discuss the second case in the `__add__()` method of `Element`, namely the case for which the parents of both operands are different (lines 14-15 in the code listed as a result of `In [55]`, on page 18). This case is treated via SageMath coercion model, which allows one to deal with additions like

```
In [60]: h1 = f + 2
         h1.display()
```

Out[60]:

$$\begin{array}{llcl} & & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x, y) & \longmapsto & \frac{2\,x^2+2\,y^2+3}{x^2+y^2+1} \\ \text{on } V: & (x', y') & \longmapsto & \frac{3\,x'^2+3\,y'^2+2}{x'^2+y'^2+1} \end{array}$$

A priori, `f + 2` is not a well defined operation, since the integer 2 does not belong to the algebra $C^\infty(M)$. However SageMath manages to treat it because 2 can be coerced (i.e. automatically and unambiguously converted) via `CM(2)` into a element of $C^\infty(M)$, namely the constant scalar field whose value is 2:

```
In [61]: CM(2).display()
```

Out[61]:

$$\begin{array}{llcl} & & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x, y) & \longmapsto & 2 \\ \text{on } V: & (x', y') & \longmapsto & 2 \end{array}$$

This happens because there exists a coercion map from the parent of 2, namely the ring of integers $\mathbb{Z}$ (denoted `ZZ` in SageMath), to $C^\infty(M)$:

```
In [62]: 2.parent()
```

Out[62]: **Z**

```
In [63]: CM.has_coerce_map_from(ZZ)
```

[17] See `https://github.com/sagemath/sage/blob/develop/src/sage/manifolds/utilities.py` for details; note that the simplifications regarding the SymPy engine are not fully implemented yet.

CHAPTER 3

Vector fields

Contents

1. Introduction

This chapter is devoted to the most basic objects of tensor calculus: vector fields. We start by defining tangent vectors and tangent spaces on a differentiable manifold (Sec. 2), and then move to vector fields (Sec. 3).

2. Tangent vectors

2.1. **Definitions.** Let M be a smooth manifold of dimension n over the topological field $\mathbb{K}$ and $C^\infty(M)$ the corresponding algebra of scalar fields introduced in Sec. 3.2. For $p \in M$, a ***tangent vector at*** p is a map

$$\boldsymbol{v} : C^\infty(M) \longrightarrow \mathbb{K} \tag{2.1}$$

such that (i) $\boldsymbol{v}$ is $\mathbb{K}$-linear and (ii) $\boldsymbol{v}$ obeys

$$\forall f, g \in C^\infty(M), \quad \boldsymbol{v}(fg) = \boldsymbol{v}(f)g(p) + f(p)\boldsymbol{v}(g). \tag{2.2}$$

Because of property (2.2), one says that $\boldsymbol{v}$ is a ***derivation at*** p.

The set T_pM of all tangent vectors at p is a vector space of dimension n over $\mathbb{K}$; it is called the ***tangent space to*** M ***at*** p.

2.2. **SageMath implementation.** To illustrate the implementation of tangent vectors in SageMath, we shall consider the same example $M = \mathbb{S}^2$ as in Chap. 2. First of all, we recreate the same objects as in Chap. 2, starting with the manifold M and its two stereographic charts $X_U = (U, (x, y))$ and $X_V = (V, (x', y'))$, with $M = U \cup V$ (the full Jupyter notebook is available at `https://sagemanifolds.obspm.fr/jncf2018/`):

In [1]:
```
%display latex
```

In [2]:
```
M = Manifold(2, 'M')
U = M.open_subset('U')
XU.<x,y> = U.chart()
V = M.open_subset('V')
XV.<xp,yp> = V.chart("xp:x' yp:y'")
M.declare_union(U,V)
XU_to_XV = XU.transition_map(XV,
                             (x/(x^2+y^2), y/(x^2+y^2)),
                             intersection_name='W',
                             restrictions1= x^2+y^2!=0,
                             restrictions2= xp^2+yp^2!=0)
XV_to_XU = XU_to_XV.inverse()
M.atlas()
```

Out[2]: $[(U,(x,y)),(V,(x',y')),(W,(x,y)),(W,(x',y'))]$

Then we introduce the point $p \in U$ of coordinates $(x,y) = (1,2)$:

In [3]:
```
p = U((1,2), chart=XU, name='p')
print(p)
```

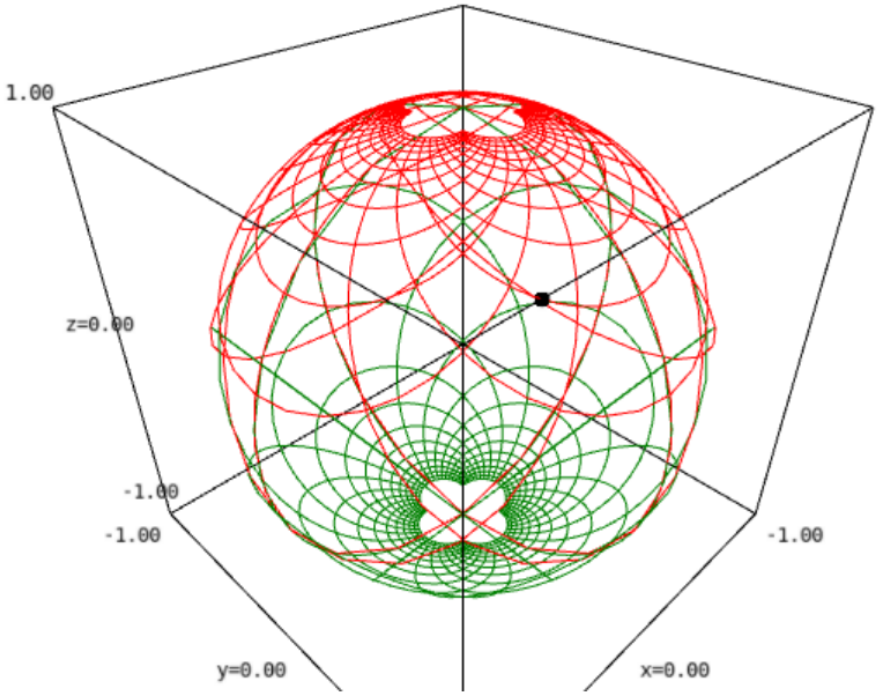
```
Point p on the 2-dimensional differentiable manifold M
```

The canonical embedding of $\mathbb{S}^2$ in $\mathbb{R}^3$ is defined mostly for graphical purposes:

In [4]:
```
R3 = Manifold(3, 'R^3', r'\mathbb{R}^3')
XR3.<X,Y,Z> = R3.chart()
Phi = M.diff_map(R3, {(XU, XR3):
                        [2*x/(1+x^2+y^2), 2*y/(1+x^2+y^2),
                        (x^2+y^2-1)/(1+x^2+y^2)],
                      (XV, XR3):
                        [2*xp/(1+xp^2+yp^2), 2*yp/(1+xp^2+yp^2),
                        (1-xp^2-yp^2)/(1+xp^2+yp^2)]},
                    name='Phi', latex_name=r'\Phi')
Phi.display()
```

Out[4]:
$$\begin{array}{llcl} \Phi: & M & \longrightarrow & \mathbb{R}^3 \\ \text{on } U: & (x,y) & \longmapsto & (X,Y,Z) = \left(\frac{2\,x}{x^2+y^2+1}, \frac{2\,y}{x^2+y^2+1}, \frac{x^2+y^2-1}{x^2+y^2+1}\right) \\ \text{on } V: & (x',y') & \longmapsto & (X,Y,Z) = \left(\frac{2\,x'}{x'^2+y'^2+1}, \frac{2\,y'}{x'^2+y'^2+1}, -\frac{x'^2+y'^2-1}{x'^2+y'^2+1}\right) \end{array}$$

In [5]:
```
graph = XU.plot(chart=XR3, mapping=Phi, number_values=25,
                label_axes=False) + \
        XV.plot(chart=XR3, mapping=Phi, number_values=25,
                color='green', label_axes=False) + \
        p.plot(chart=XR3, mapping=Phi, label_offset=0.05)
show(graph, viewer='threejs', online=True)
```



Finally, the last objects defined in Chap. 2 are the scalar field f:

In [6]:
```
f = M.scalar_field({XU: 1/(1+x^2+y^2), XV: (xp^2+yp^2)/(1+xp^2+yp^2)},
                   name='f')
f.display()
```

Out[6]:
$$\begin{array}{llcl} f: & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x, y) & \longmapsto & \frac{1}{x^2+y^2+1} \\ \text{on } V: & (x', y') & \longmapsto & \frac{x'^2+y'^2}{x'^2+y'^2+1} \end{array}$$

and its parent, namely the commutative algebra $C^\infty(M)$ of smooth maps $M \to \mathbb{R}$:

In [7]:
```
CM = M.scalar_field_algebra()
CM
```

Out[7]: $C^\infty(M)$

The tangent space at the point p introduced in `In [3]` is generated by

In [8]:
```
Tp = M.tangent_space(p)
Tp
```

Out[8]: $T_p M$

It is a vector space over $\mathbb{K}$ (here $\mathbb{K} = \mathbb{R}$, which is represented by SageMath's Symbolic Ring `SR`):

In [9]:
```
print(Tp.category())
```

```
Category of finite dimensional vector spaces over Symbolic Ring
```

The dimension of the vector space T_pM equals that of the manifold M:

In [10]:
```
dim(Tp)
```

Out[10]: 2

Tangent spaces are implemented as a class inherited from `TangentSpace` via the category framework:

In [11]:
```
type(Tp)
```

Out[11]:
```
<class
 'sage.manifolds.differentiable.tangent_space.TangentSpace_with_category'>
```

`FiniteRankFreeModule`,[18], which, in SageMath is devoted to free modules of finite rank without any distinguished basis:

In [12]:
```
isinstance(Tp, FiniteRankFreeModule)
```

Out[12]: `True`

Remark 1: In SageMath, free modules with a distinguished basis are created with the command `FreeModule` or `VectorSpace` and belong to classes different from `FiniteRankFreeModule`. The differences are illustrated at
`http://doc.sagemath.org/html/en/reference/modules/sage/tensor/modules/finite_rank_free_module.html#diff-freemodule`.

Two bases of T_pM are already available: those generated by the derivations at p along the coordinates of charts `XU` and `XV` respectively:

[18] `http://doc.sagemath.org/html/en/reference/tensor_free_modules/sage/tensor/modules/finite_rank_free_module.html`

In [13]:
```
Tp.bases()
```

Out[13]: $\left[\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right), \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right]$

None of these bases is distinguished, but one if the default one, which simply means that it is the basis to be considered if the basis argument is skipped in some methods:

In [14]:
```
Tp.default_basis()
```

Out[14]: $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$

A tangent vector is created as an element of the tangent space by the standard SageMath procedure `new_element = parent(...)`, where `...` stands for some material sufficient to construct the element:

In [15]:
```
vp = Tp((-3, 2), name='v')
print(vp)
```

```
Tangent vector v at Point p on the 2-dimensional differentiable manifold M
```

Since the basis is not specified, the pair $(-3, 2)$ refers to components with respect to the default basis:

In [16]:
```
vp.display()
```

Out[16]: $v = -3\frac{\partial}{\partial x} + 2\frac{\partial}{\partial y}$

We have of course

In [17]:
```
vp.parent()
```

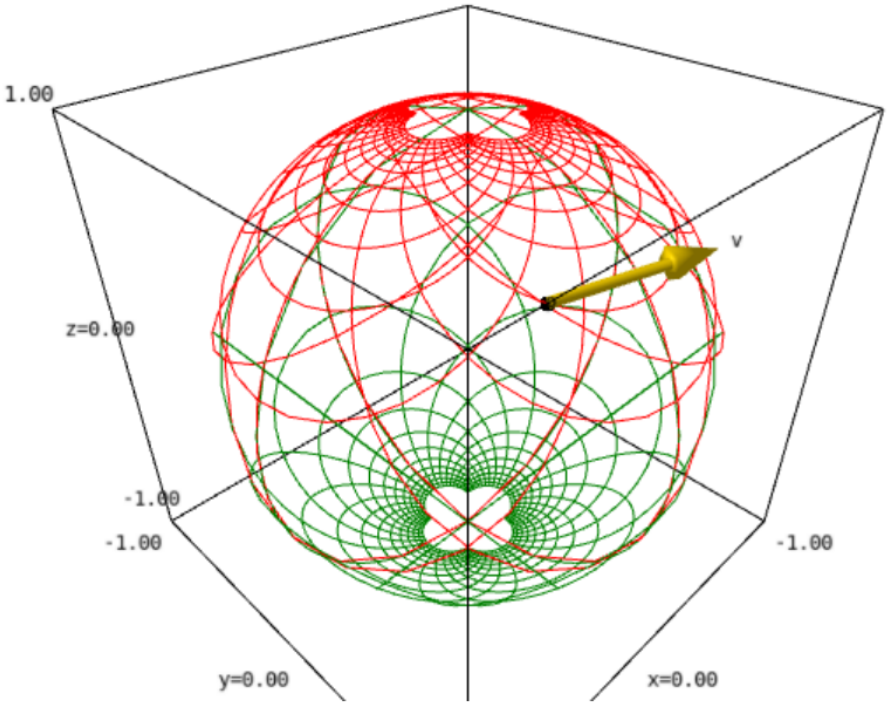
Out[17]: $T_p\,M$

In [18]:
```
vp in Tp
```

Out[18]: True

As other manifold objects, tangent vectors have some plotting capabilities:

In [19]:
```
graph += vp.plot(chart=XR3, mapping=Phi, scale=0.5, color='gold')
show(graph, viewer='threejs', online=True)
```



The main attribute of the object `vp` representing the vector $\boldsymbol{v}$ is the private dictionary `_components`, which stores the components of $\boldsymbol{v}$ in various bases of T_pM:

```
In [20]: vp._components
```

Out[20]: $\left\{ \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) :$
1-index components w.r.t. Basis (d/dx,d/dy) on the Tangent space at Point p on the 2-dimensional differentiable manifold M $\right\}$

The keys of the dictionary `_components` are the bases of T_pM, while the values belong to the class `Components`[19] devoted to store ring elements indexed by integers or tuples of integers:

```
In [21]: vpc = vp._components[Tp.default_basis()]
         vpc
```

```
Out[21]: 1-index components w.r.t. Basis (d/dx,d/dy) on the Tangent space at Point p
         on the 2-dimensional differentiable manifold M
```

```
In [22]: type(vpc)
```

```
Out[22]: <class 'sage.tensor.modules.comp.Components'>
```

The components themselves are stored in the private dictionary `_comp` of the `Components` object, with the indices as keys:

```
In [23]: vpc._comp
```

Out[23]: $\{(0) : -3, (1) : 2\}$

Hence the components are not stored via a sequence data type (list or tuple), as one might have expected, but via a mapping type (dictionary). This is a general feature of the class `Components`

[19] `http://doc.sagemath.org/html/en/reference/tensor_free_modules/sage/tensor/modules/comp.html`

and all its subclasses, which permits to not store vanishing components and, in case of symmetries (for multi-index objects like tensors), to store only non-redundant components.

3. Vector fields

3.1. **Definition.** The ***tangent bundle*** of M is the disjoint union of the tangent spaces at all points of M:

$$TM = \coprod_{p\in M} T_pM. \tag{3.1}$$

Elements of TM are usually denoted by $(p, \boldsymbol{u})$, with $\boldsymbol{u} \in T_pM$. The tangent bundle is canonically endowed with the ***projection map***:

$$\begin{array}{rccc} \pi: & TM & \longrightarrow & M \\ & (p, \boldsymbol{u}) & \longmapsto & p. \end{array} \tag{3.2}$$

The tangent bundle inherits some manifold structure from M: TM is a smooth manifold of dimension $2n$ over $\mathbb{K}$ (n being the dimension of M).

A ***vector field*** on M is a continuous right-inverse of the projection map, i.e. it is a map

$$\begin{array}{rccc} \boldsymbol{v}: & M & \longrightarrow & TM \\ & p & \longmapsto & \boldsymbol{v}|_p \end{array} \tag{3.3}$$

such that $\pi \circ \boldsymbol{v} = \mathrm{Id}_M$, i.e. such that

$$\forall p \in M, \quad \boldsymbol{v}|_p \in T_pM. \tag{3.4}$$

3.2. **Module of vector fields.** The set $\mathfrak{X}(M)$ of all vector fields on M is naturally endowed with two algebraic structures:

(1) $\mathfrak{X}(M)$ is a (infinite dimensional) vector space over $\mathbb{K}$ — the base field of M —, the scalar multiplication $\mathbb{K} \times \mathfrak{X}(M) \to \mathfrak{X}(M)$, $(\lambda, \boldsymbol{v}) \mapsto \lambda\boldsymbol{v}$ being defined by

$$\forall p \in M, \quad (\lambda\boldsymbol{v})|_p = \lambda \boldsymbol{v}|_p\,, \tag{3.5}$$

where the right-hand side involves the scalar multiplication in the vector space T_pM;

(2) $\mathfrak{X}(M)$ is a module over $C^\infty(M)$ — the commutative algebra of scalar fields —, the scalar multiplication $C^\infty(M) \times \mathfrak{X}(M) \to \mathfrak{X}(M)$, $(f, \boldsymbol{v}) \mapsto f\boldsymbol{v}$ being defined by

$$\forall p \in M, \quad (f\boldsymbol{v})|_p = f(p)\boldsymbol{v}|_p\,, \tag{3.6}$$

where the right-hand side involves the scalar multiplication by $f(p) \in \mathbb{K}$ in the vector space T_pM.

An important subcase of 2 is when $\mathfrak{X}(M)$ is a ***free module*** over $C^\infty(M)$, i.e. when $\mathfrak{X}(M)$ admits a basis (a generating set consisting of linearly independent elements). If this occurs, then $\mathfrak{X}(M)$ is actually a ***free module of finite rank*** over $C^\infty(M)$ and its rank is n – the dimension of M over $\mathbb{K}$, which means that all bases share the same cardinality, namely n. One says that M is a ***parallelizable*** manifold. A basis $(\boldsymbol{e}_a)_{1\le a\le n}$ of $\mathfrak{X}(M)$ is called a ***vector frame***; for any $p \in M$, $(\boldsymbol{e}_a|_p)_{1\le a\le n}$ is then a basis of the tangent vector space T_pM. Any vector field has a unique decomposition with respect to the vector frame[20] $(\boldsymbol{e}_a)_{1\le a\le n}$:

$$\forall \boldsymbol{v} \in \mathfrak{X}(M), \quad \boldsymbol{v} = v^a \boldsymbol{e}_a, \quad \text{with } v^a \in C^\infty(M). \tag{3.7}$$

At each point $p \in M$, Eq. (3.7) gives birth to an identity in the tangent space T_pM:

$$\boldsymbol{v}|_p = v^a(p)\, \boldsymbol{e}_a|_p\,, \quad \text{with } v^a(p) \in \mathbb{K}, \tag{3.8}$$

which is nothing but the expansion of the tangent vector $\boldsymbol{v}|_p$ on the basis $(\boldsymbol{e}_a|_p)_{1\le a\le n}$ of the vector space T_pM.

Note that if M is covered by a chart X, i.e. M is the domain of the chart X, then M is parallelizable and a vector frame is $(\partial/\partial x^a)_{1\le a\le n}$, where the x^a's are the coordinates of chart X. Such a vector frame is called a ***coordinate frame*** or ***natural basis***. More generally, examples of parallelizable manifolds are [12]

[20] Einstein's convention for summation on repeated indices is assumed.

- the Cartesian space $\mathbb{R}^n$ for $n = 1, 2, \ldots$,
- the circle $\mathbb{S}^1$,
- the torus $\mathbb{T}^2 = \mathbb{S}^1 \times \mathbb{S}^1$,
- the sphere $\mathbb{S}^3 \simeq \mathrm{SU}(2)$, as any Lie group,
- the sphere $\mathbb{S}^7$,
- any orientable 3-manifold (Steenrod theorem [23]).

On the other hand, examples of non-parallelizable manifolds are

- the sphere $\mathbb{S}^2$ (as a consequence of the hairy ball theorem), as well as any sphere $\mathbb{S}^n$ with $n \notin \{1, 3, 7\}$,
- the real projective plane $\mathbb{RP}^2$.

Actually, "most" manifolds are non-parallelizable. As noticed above, if a manifold is covered by a single chart, it is parallelizable (the prototype being $\mathbb{R}^n$). But the reverse is not true: $\mathbb{S}^1$ and $\mathbb{T}^2$ are parallelizable and require at least two charts to cover them.

3.3. **SageMath implementation.** Among the two algebraic structures for $\mathfrak{X}(M)$ discussed in Sec. 3.2, we select the second one, i.e. we consider $\mathfrak{X}(M)$ as a $C^\infty(M)$-module. With respect to the infinite-dimensional $\mathbb{K}$-vector space point of view, the advantage for the implementation is the reduction to finite-dimensional structures: free modules of rank n on parallelizable open subsets of M. Indeed, if U is such an open subset, i.e. if $\mathfrak{X}(U)$ is a free $C^\infty(U)$-module of rank n, the generic class `FiniteRankFreeModule` discussed in Sec. 2.2 can be used to implement $\mathfrak{X}(U)$. The great benefit is that all calculus implemented on the free module elements, like the addition or the scalar multiplication, can be used as such for vector fields. This implies that vector fields will be described by their (scalar-field) components on vector frames, as defined by Eq. (3.7), on parallelizable open subsets of M.

If the manifold M is not parallelizable, we assume that it can be covered by a finite number m of parallelizable open subsets U_i $(1 \leq i \leq m)$:

$$M = \bigcup_{i=1}^{m} U_i, \qquad \text{with} \quad U_i \text{ parallelizable} \tag{3.9}$$

In particular, this holds if M is compact, for any compact manifold admits a finite atlas.

For each $i \in \{1, \ldots, m\}$, $\mathfrak{X}(U_i)$ is a free module of rank $n = \dim M$ and is implemented in SageMath as an instance of `VectorFieldFreeModule`, which is a subclass of `FiniteRankFreeModule`. This inheritance is illustrated in Fig. 3.1. On that figure, we note that the class `TangentSpace` discussed in Sec. 2.2 inherits from
`FiniteRankFreeModule` as well.

A vector field $\boldsymbol{v} \in \mathfrak{X}(M)$ is then described by its restrictions $\left(\boldsymbol{v}|_{U_i}\right)_{1\leq i\leq m}$ to each of the U_i's. Assuming that at least one vector frame is introduced in each of the U_i's, $(\boldsymbol{e}_{i,a})_{1\leq a\leq n}$ say, the restriction $\boldsymbol{v}|_{U_i}$ of $\boldsymbol{v}$ to U_i is decribed by its components v_i^a in that frame:

$$\boldsymbol{v}|_{U_i} = v_i^a\, \boldsymbol{e}_{i,a}, \quad \text{with } v_i^a \in C^\infty(U_i). \tag{3.10}$$

Let us illustrate this strategy with the example of $\mathbb{S}^2$. We get $\mathfrak{X}(M)$ by[21]

In [24]:
```
YM = M.vector_field_module()
YM
```

Out[24]: $\mathfrak{X}(M)$

As discussed above, $\mathfrak{X}(M)$ is considered as a module over $C^\infty(M)$:

In [25]:
```
YM.category()
```

Out[25]: $\mathbf{Modules}_{C^\infty(M)}$

Since the algebra $C^\infty(M)$ is denoted `CM`, we have

[21] We are using `YM` to denote $\mathfrak{X}(M)$ and not `XM`, because we reserve the symbol `X` to denote coordinate charts, as `XU`, `XV` or `XR3`.

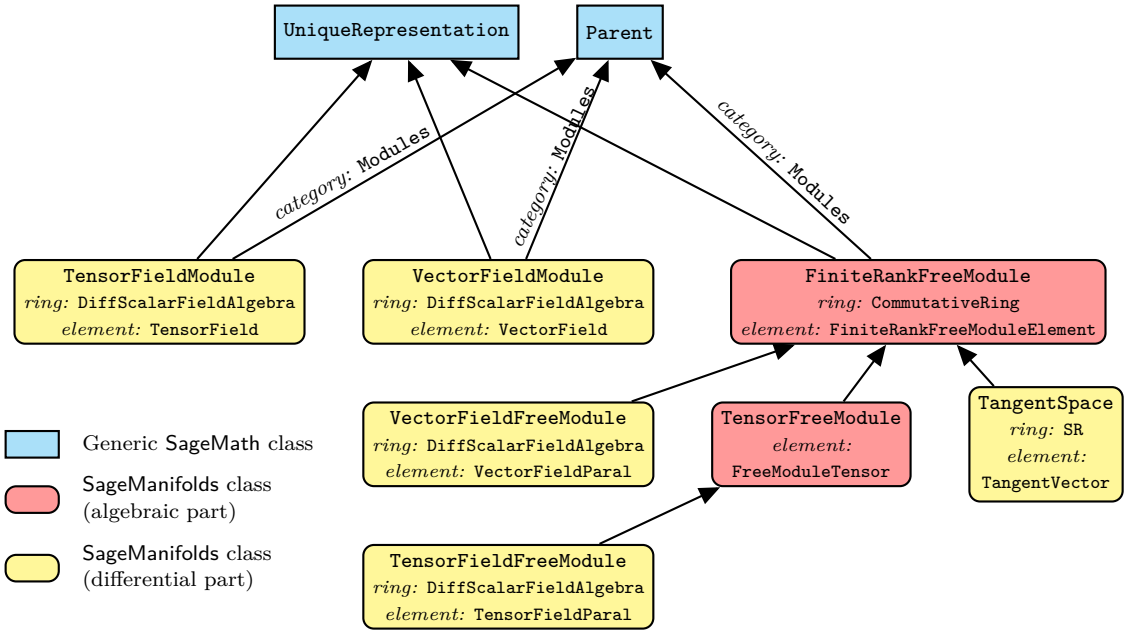


Figure 3.1: SageMath classes for modules involved in differentiable manifolds.

```
In [26]: YM.base_ring() is CM
```

```
Out[26]: True
```

$\mathfrak{X}(M)$ is not a free module; in particular, we can check that its SageMath implementation does not belong to the class `FiniteRankFreeModule`:

```
In [27]: isinstance(YM, FiniteRankFreeModule)
```

```
Out[27]: False
```

This is because $M = \mathbb{S}^2$ is not a parallelizable manifold:

```
In [28]: M.is_manifestly_parallelizable()
```

```
Out[28]: False
```

Via SageMath category framework, the module $\mathfrak{X}(M)$ is implemented by a dynamically-generated subclass of the class `VectorFieldModule`, which is devoted to modules of vector fields on non-parallelizable manifolds:

```
In [29]: type(YM)
```

```
Out[29]: <class 'sage.manifolds.differentiable.vectorfield_module.
                 VectorFieldModule_with_category'>
```

On the contrary, the set $\mathfrak{X}(U)$ of vector fields on U is a free module of finite rank over the algebra $C^\infty(U)$:

```
In [30]: YU = U.vector_field_module()
         isinstance(YU, FiniteRankFreeModule)
```

```
Out[30]: True
```

In [31]:
```
YU.base_ring()
```

Out[31]: $C^\infty(U)$

This is because the open subset U is a parallelizable manifold:

In [32]:
```
U.is_manifestly_parallelizable()
```

Out[32]: True

being the domain of a coordinate chart:

In [33]:
```
U.is_manifestly_coordinate_domain()
```

Out[33]: True

We can check that in U's atlas, at least one chart has U for domain:

In [34]:
```
U.atlas()
```

Out[34]: $[(U,(x,y)),(W,(x,y)),(W,(x',y'))]$

This chart is `XU` $= (U,(x,y))$, i.e. the chart of stereographic coordinates from the North pole. The rank of $\mathfrak{X}(U)$ as a free $C^\infty(U)$-module is the manifold's dimension:

In [35]:
```
rank(YU)
```

Out[35]: 2

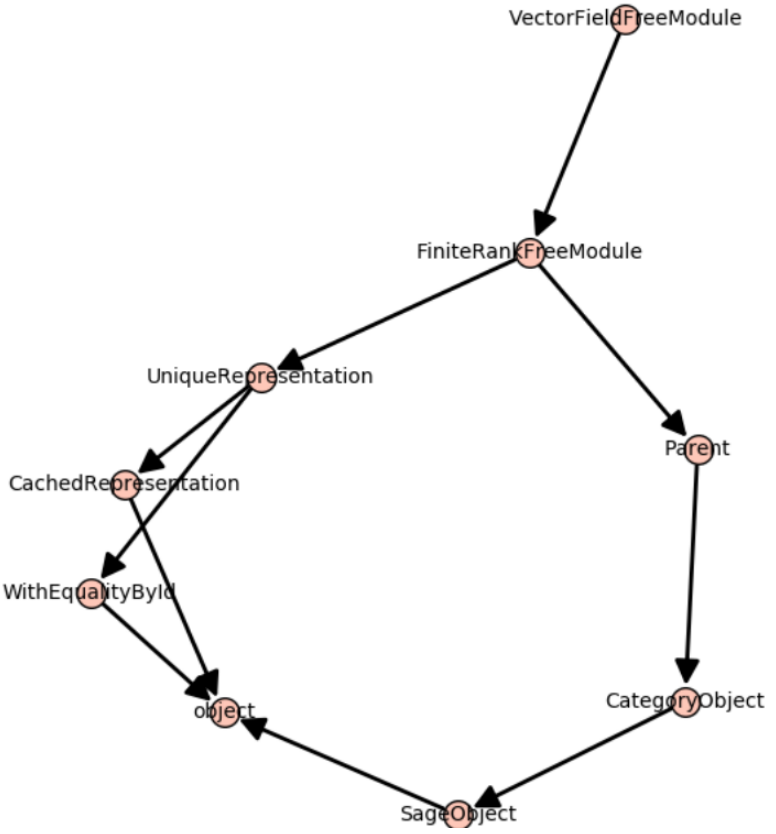
Via the category framework, the free module $\mathfrak{X}(U)$ is implemented by a dynamically-generated subclass of the class `VectorFieldFreeModule`, which is devoted to modules of vector fields on parallelizable manifolds:

In [36]:
```
type(YU)
```

Out[36]:
```
<class 'sage.manifolds.differentiable.vectorfield_module.
        VectorFieldFreeModule_with_category'>
```

The class `VectorFieldFreeModule` is itself a subclass of the generic class `FiniteRankFreeModule`:

In [37]:
```
class_graph(
  sage.manifolds.differentiable.vectorfield_module.VectorFieldFreeModule
).plot()
```



Since U is a chart domain, the free module $\mathfrak{X}(U)$ is automatically endowed with a basis, which is the coordinate frame associated to the chart:

In [38]:
```
YU.bases()
```

Out[38]: $\left[\left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right)\right]$

Let us denote by `eU` this frame. We can set `eU = YU.bases()[0]` or alternatively

In [39]:
```
eU = YU.default_basis()
eU
```

Out[39]: $\left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right)$

Another equivalent instruction would have been `eU = U.default_frame()`.

Similarly, $\mathfrak{X}(V)$ is a free module, endowed with the coordinate frame associated to stereographic coordinates from the South pole, which we denote by `eV`:

In [40]:
```
YV = V.vector_field_module()
YV.bases()
```

Out[40]: $\left[\left(V, \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right)\right]$

In [41]:
```
eV = YV.default_basis()
eV
```

Out[41]: $\left(V, \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right)$

If we consider the intersection $W = U \cap V$, we notice its module of vector fields is endowed with two bases, reflecting the fact that W is covered by two charts: $(W, (x, y))$ and $(W, (x', y'))$:

In [42]:
```
W = U.intersection(V)
YW = W.vector_field_module()
YW.bases()
```

Out[42]: $\left[\left(W, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right), \left(W, \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right)\right]$

Let us denote by `eUW` and `eUV` these two bases, which are actually the restrictions of the vector frames `eU` and `eV` to W:

In [43]:
```
eUW = eU.restrict(W)
eVW = eV.restrict(W)
YW.bases() == [eUW, eVW]
```

Out[43]: `True`

The free module $\mathfrak{X}(W)$ is also automatically endowed with automorphisms connecting the two bases, i.e. change-of-frame operators:

In [44]:
```
W.changes_of_frame()
```

Out[44]:

$$\left\{ \left(\left(W, \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right), \left(W, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right)\right) :$$

Field of tangent-space automorphisms on the Open subset W of the 2-dimensional differentiable manifold M,

$$\left(\left(W, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right), \left(W, \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right)\right) :$$

Field of tangent-space automorphisms on the Open subset W of the 2-dimensional differentiable manifold M $\Big\}$

The first of them is

In [45]:
```
P = W.change_of_frame(eUW, eVW)
P
```

Out[45]:
```
Field of tangent-space automorphisms on the Open subset W of the
2-dimensional differentiable manifold M
```

It belongs to the general linear group of the free module $\mathfrak{X}(W)$:

In [46]:
```
P.parent()
```

Out[46]: $\mathrm{GL}\left(\mathfrak{X}\left(W\right)\right)$

and its matrix is deduced from the Jacobian matrix of the transition map `XV` $\to$ `XU`:

In [47]:
```
P[:]
```

Out[47]: $\begin{pmatrix} -x^2 + y^2 & -2\,xy \\ -2\,xy & x^2 - y^2 \end{pmatrix}$

3.4. **Construction and manipulation of vector fields.** Let us introduce a vector field $\boldsymbol{v}$ on M:

In [48]:
```
v = M.vector_field(name='v')
v[eU, 0] = f.restrict(U)
v[eU, 1] = -2
v.display(eU)
```

Out[48]: $v = \left(\frac{1}{x^2+y^2+1}\right)\frac{\partial}{\partial x} - 2\frac{\partial}{\partial y}$

Notice that, at this stage, we have defined $\boldsymbol{v}$ only on U, by setting its components in the vector frame `eU`, either explicitly as scalar fields, like the component v^0 set to the restriction of f to U or implicitly, like the component v^1: the integer `-2` will be coerced to the constant scalar field of value -2 (cf. Sec. 3.3). We can ask for the scalar-field value of a component via the double-bracket operator; since `eU` is the default frame on M, we do not have to specify it:

In [49]:
```
v[[0]]
```

Out[49]: f

In [50]:
```
v[[0]].display()
```

Out[50]:
$$\begin{array}{lccc} f: & U & \longrightarrow & \mathbb{R} \\ & (x,y) & \longmapsto & \frac{1}{x^2+y^2+1} \\ \text{on } W: & (x',y') & \longmapsto & \frac{x'^2+y'^2}{x'^2+y'^2+1} \end{array}$$

Note that, for convenience, the single bracket operator returns a chart function of the component:

In [51]:
```
v[0]
```

Out[51]: $\frac{1}{x^2+y^2+1}$

The restriction of $\boldsymbol{v}$ to W is of course

In [52]:
```
v.restrict(W).display(eUW)
```

Out[52]: $v = \left(\frac{1}{x^2+y^2+1}\right)\frac{\partial}{\partial x} - 2\frac{\partial}{\partial y}$

Since we have a second vector frame on W, namely `eVW`, and the change-of-frame automorphisms are known, we can ask for the components of $\boldsymbol{v}$ with respect to that frame:

In [53]:
```
v.restrict(W).display(eVW)
```

Out[53]:
$$v = \left(\frac{4\,xy^3 - x^2 + 4\,(x^3+x)y + y^2}{x^6+y^6+(3\,x^2+1)y^4+x^4+(3\,x^4+2\,x^2)y^2}\right)\frac{\partial}{\partial x'} + \left(-\frac{2\,(x^4-y^4+x^2+xy-y^2)}{x^6+y^6+(3\,x^2+1)y^4+x^4+(3\,x^4+2\,x^2)y^2}\right)\frac{\partial}{\partial y'}$$

Notice that the components are expressed in terms of the coordinates (x, y) since they form the default chart on W. To have them expressed in terms of the coordinates (x', y'), we have to add the restriction of the chart $(V, (x', y'))$ to W as the second argument of the method `display()`:

In [54]:
```
v.restrict(W).display(eVW, XV.restrict(W))
```

Out[54]:
$$v = \left(-\frac{x'^4 - 4\,x'y'^3 - y'^4 - 4\left(x'^3 + x'\right)y'}{x'^2 + y'^2 + 1} \right) \frac{\partial}{\partial x'} + \left(-\frac{2\left(x'^4 + x'^3y' + x'y'^3 - y'^4 + x'^2 - y'^2\right)}{x'^2 + y'^2 + 1} \right) \frac{\partial}{\partial y'}$$

We extend the expression of $\boldsymbol{v}$ to the full vector frame `XV` by continuation of this expression:

In [55]:
```
v.add_comp_by_continuation(eV, W, chart=XV)
```

We have then

In [56]:
```
v.display(eV)
```

Out[56]:
$$v = \left(-\frac{x'^4 - 4\,x'y'^3 - y'^4 - 4\left(x'^3 + x'\right)y'}{x'^2 + y'^2 + 1} \right) \frac{\partial}{\partial x'} + \left(-\frac{2\left(x'^4 + x'^3y' + x'y'^3 - y'^4 + x'^2 - y'^2\right)}{x'^2 + y'^2 + 1} \right) \frac{\partial}{\partial y'}$$

At this stage, the vector field $\boldsymbol{v}$ is defined in all M. According to the hairy ball theorem, it has to vanish somewhere. Let us show that this occurs at the North pole, by first introducing the latter, as the point of stereographic coordinates $(x', y') = (0, 0)$:

In [57]:
```
N = M((0,0), chart=XV, name='N')
print(N)
```

```
Point N on the 2-dimensional differentiable manifold M
```

As a check, we verify that the image of N by the canonical embedding $\Phi : \mathbb{S}^2 \to \mathbb{R}^3$ is the point of Cartesian coordinates $(0, 0, 1)$:

In [58]:
```
XR3(Phi(N))
```

Out[58]: $(0, 0, 1)$

The vanishing of $\boldsymbol{v}|_N$:

In [59]:
```
v.at(N).display()
```

Out[59]: $v = 0$

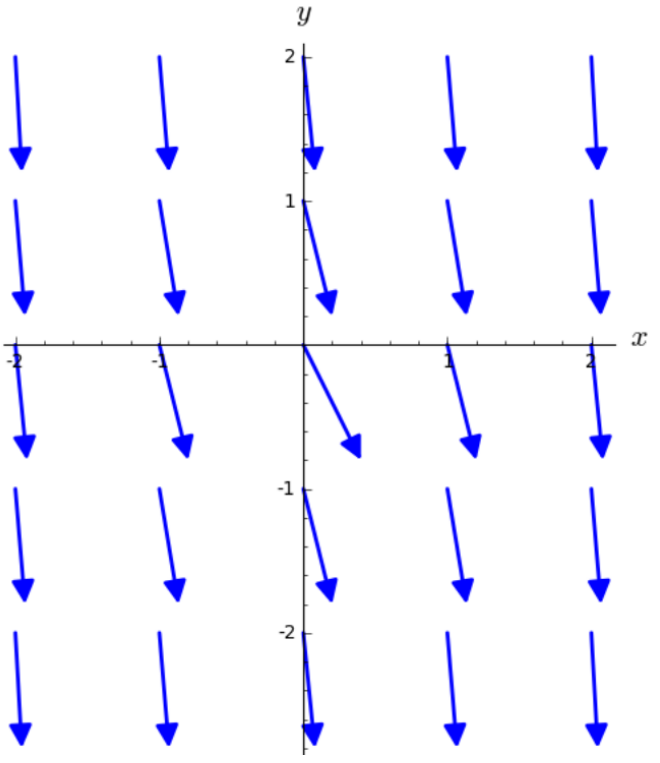
On the other hand, $\boldsymbol{v}$ does not vanish at the point p introduced above:

In [60]:
```
v.at(p).display()
```

Out[60]:
$$v = \frac{1}{6}\frac{\partial}{\partial x} - 2\frac{\partial}{\partial y}$$

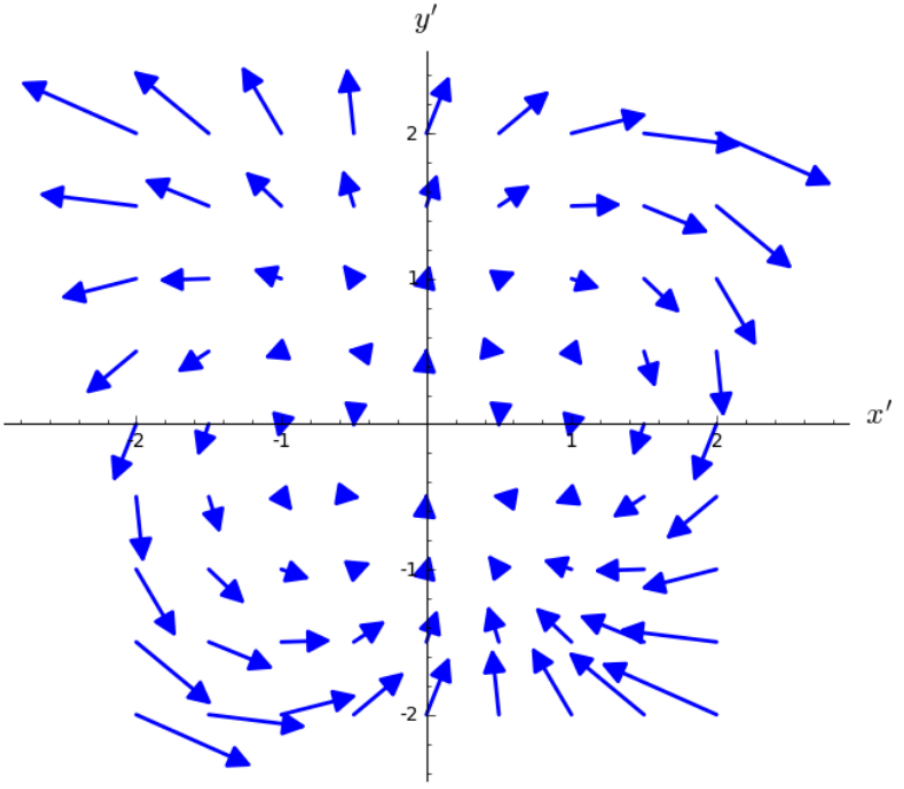
We may plot the vector field $\boldsymbol{v}$ in terms of the stereographic coordinates from the North pole:

In [61]:
```
v.plot(chart=XU, chart_domain=XU, max_range=2,
       number_values=5, scale=0.4, aspect_ratio=1)
```



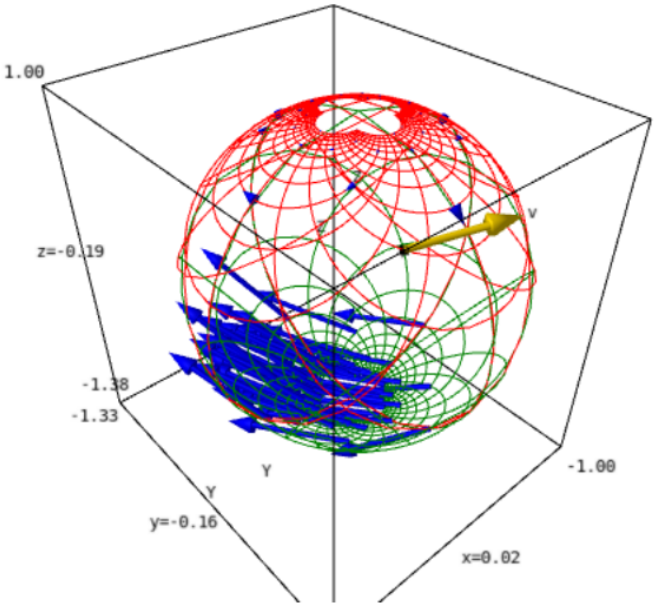
or in term of those from the South pole:

In [62]:
```
v.plot(chart=XV, chart_domain=XV, max_range=2,
       number_values=9, scale=0.05, aspect_ratio=1)
```



Thanks to the embedding Φ, we may also have a 3D plot of the vector field $\boldsymbol{v}$ atop of the 3D plot already obtained:

```
In [63]: graph_v = v.plot(chart=XR3, mapping=Phi, chart_domain=XU,
                          number_values=7, scale=0.2) + \
                 v.plot(chart=XR3, mapping=Phi, chart_domain=XV,
                          number_values=7, scale=0.2)
         show(graph + graph_v, viewer='threejs', online=True)
```



Note that the sampling, performed on the two charts `XU` and `XV` is not uniform on the sphere. A better sampling would be achieved by introducing spherical coordinates.

3.5. **Implementation details regarding vector fields.** Let us now investigate some internals of the implementation of vector fields. Vector fields on M are implemented via the class `VectorField`[22] (actually by a dynamically generated subclass of it, within SageMath category framework):

```
In [64]: isinstance(v, sage.manifolds.differentiable.vectorfield.VectorField)
```

```
Out[64]: True
```

Since M is not parallelizable, the defining data of a vector field $\boldsymbol{v}$ on M are its restrictions $\left(\boldsymbol{v}|_{U_i}\right)_{1\leq i\leq m}$ to parallelizable open subsets U_i, following the scheme presented in Sec. 3.3. These restrictions are stored in the private dictionary `_restrictions`, whose keys are the open subsets:

```
In [65]: v._restrictions
```

```
Out[65]: {V : v, W : v, U : v}
```

Let us consider one of these restrictions, for instance the restriction $\boldsymbol{v}|_U$ to U:

```
In [66]: vU = v._restrictions[U]
         vU is v.restrict(U)
```

```
Out[66]: True
```

Since U is a parallelizable open subset, the object `vU` belongs to the class `VectorFieldParal`, which is devoted to vector fields on parallelizable manifolds:

[22] http://doc.sagemath.org/html/en/reference/manifolds/sage/manifolds/differentiable/vectorfield.html

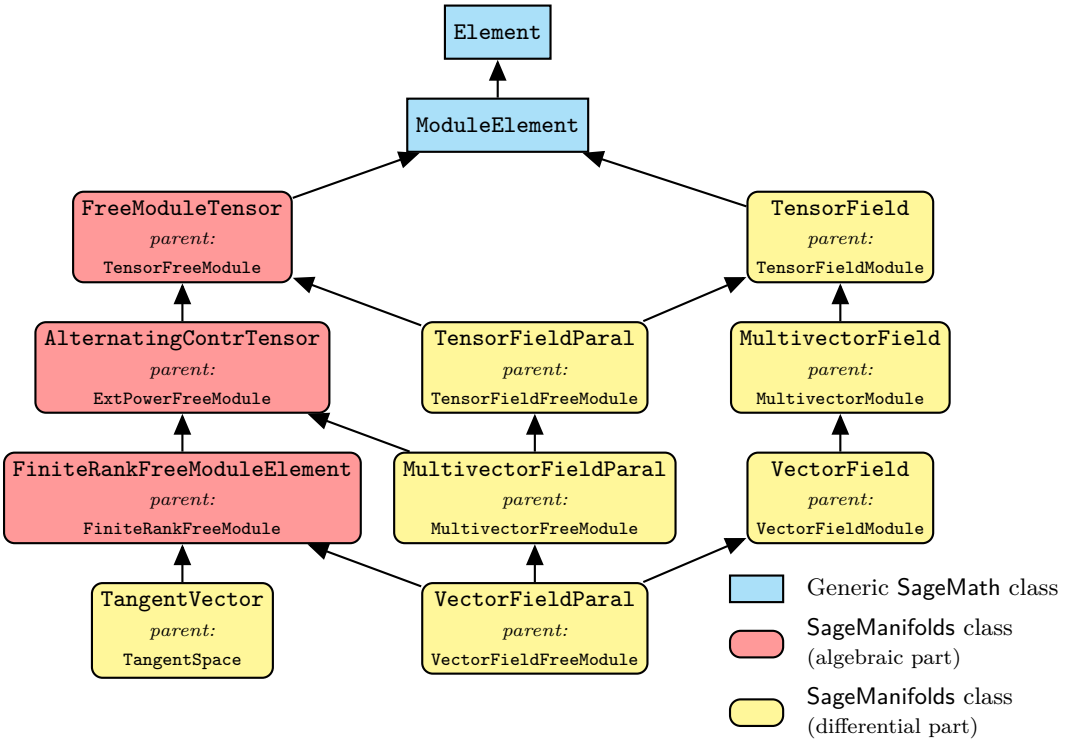


Figure 3.2: SageMath classes for tensor fields involved in differentiable manifolds. There are various multiple inheritances involving diamond diagrams; Python's method resolution order algorithm (MRO) relies on the ordering of the parents in the class declaration and this order can be read from the left to the right in this figure. For instance, the class `VectorFieldParal` is declared as `class VectorFieldParal(FiniteRankFreeModuleElement, MultivectorFieldParal, VectorField)`.

In [67]:
```
isinstance(vU, sage.manifolds.differentiable.vectorfield.VectorFieldParal)
```

Out[67]: `True`

The class `VectorFieldParal` inherits both from `FiniteRankFreeModuleElement` (as `TangentVector`) and from `VectorField` (see Fig. 3.2). The defining data of $\boldsymbol{v}|_U$ are its sets of components with respect to (possibly various) vector frames on U, according to Eq. (3.10). The sets of components are stored in the private dictionary `_components`, whose keys are the vector frames:

In [68]:
```
vU._components
```

Out[68]: $\left\{\left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right) : \texttt{1-index components w.r.t. Coordinate frame (U, (d/dx,d/dy))}\right\}$

Similarly, we have:

In [69]:
```
v._restrictions[W]._components
```

Out[69]:
$$\left\{\left(W, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\right) : \texttt{1-index components w.r.t. Coordinate frame (W, (d/dx,d/dy))},$$
$$\left(W, \left(\frac{\partial}{\partial x'}, \frac{\partial}{\partial y'}\right)\right) : \texttt{1-index components w.r.t. Coordinate frame (W, (d/dxp,d/dyp)}\right\}$$

The values of the dictionary `_components` belong to the same class `Components` as that discussed in Sec. 2.2 for the storage of components of tangent vectors:

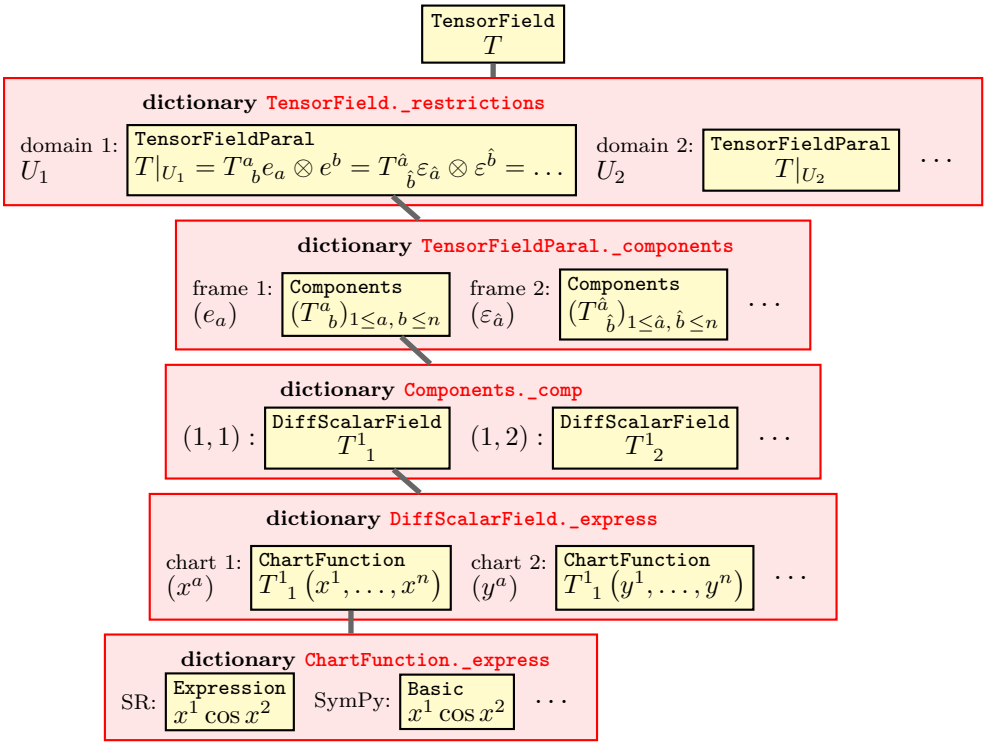


Figure 3.3: Internal storage of tensor fields. Red boxes represent Python dictionaries, yellow boxes are dictionary values, with the corresponding dictionary key located on the left of them. The Python class of each dictionary value is indicated in typewriter font at the top of the yellow box. In the hierarchical tree, only the leftmost branch is indicated by grey connectors. In the special case of vector fields, the classes `TensorField` and `TensorFieldParal` are to be replaced by `VectorField` and `VectorFieldParal` respectively.

```
In [70]: vUc = vU._components[eU]
         vUc
```

```
Out[70]: 1-index components w.r.t. Coordinate frame (U, (d/dx,d/dy))
```

```
In [71]: type(vUc)
```

```
Out[71]: <class 'sage.tensor.modules.comp.Components'>
```

As already mentioned in Sec. 2.2, the components themselves are stored in the private attribute `_comp` of the `Components` object; this is a dictionary whose keys are the indices:

```
In [72]: vUc._comp
```

```
Out[72]: {(0) : f,
          (1) : Scalar field on the Open subset U of the 2-dimensional differentiable manifold M}
```

The difference with the tangent vector case is that the values of that dictionary are now scalar fields, i.e. elements of $C^\infty(U)$ in the present case. This is of course in agreement with the treatment of $\mathfrak{X}(U)$ as a free module over $C^\infty(U)$, as discussed in Sec. 3.3. Taking into account the storage of scalar fields presented in Sec. 3.1, the full storage structure of vector fields is presented in Fig. 3.3 (the latter actually regards tensor fields, of which vector fields constitute a subcase).

Let us perform some algebraic operation on vector fields:

```
In [73]: w = v + f*v
         w
```

Out[73]: Vector field on the 2-dimensional differentiable manifold M

The code for the addition is accessible via

```
In [74]: v.__add__??
```

```
File: .../src/sage/structure/element.pyx
def __add__(left, right):
    """
    Top-level addition operator for :class:`Element` invoking
    the coercion model.

    See :ref:`element_arithmetic`.
    ...
    """
    cdef int cl = classify_elements(left, right)
    if HAVE_SAME_PARENT(cl):
        return (<Element>left)._add_(right)
    # Left and right are Sage elements => use coercion model
    if BOTH_ARE_ELEMENT(cl):
        return coercion_model.bin_op(left, right, add)
    ...
```

This is exactly the same method `__add__()` as that discussed in Sec. 3.3 for the addition of scalar fields (cf. page 18), namely the method `__add__()` of the top-level class `Element`, from which both `VectorField` and `DiffScalarField` inherit, cf. the inheritance diagrams of Figs. 3.2 and 3.1 (taking into account that `CommutativeAlgebraElement` is a subclass of `Element`). In the present case, `left` = `v` and `right` = `f*v` have the same parent, so that the actual result is computed in line 12, via the method `_add_()` (note the single underscore on each side of `add`). This operator is implemented at the level of `TensorField`, as it can be checked from the source code (see lines 3 and 29 below):

```
In [75]: v._add_??
```

```
def _add_(self, other):
    """
    Tensor field addition.

    INPUT:

    - ``other`` -- a tensor field, in the same tensor module as ``self``

    OUTPUT:

    - the tensor field resulting from the addition of ``self``
      and ``other``
    ...
    """
    resu_rst = {}
    for dom in self._common_subdomains(other):
        resu_rst[dom] = self._restrictions[dom] + other._restrictions[dom]
    some_rst = next(itervalues(resu_rst))
    resu_sym = some_rst._sym
    resu_antisym = some_rst._antisym
    resu = self._vmodule.tensor(self._tensor_type, sym=resu_sym,
                                antisym=resu_antisym)
    resu._restrictions = resu_rst
    if self._name is not None and other._name is not None:
        resu._name = self._name + '+' + other._name
```

```
26      if self._latex_name is not None and other._latex_name is not None:
27          resu._latex_name = self._latex_name + '+' + other._latex_name
28      return resu
29  File:  .../site-packages/sage/manifolds/differentiable/tensorfield.py
```

The first step in the addition of two vector fields is to search in the restrictions of both vector fields for common domains: this is performed in line 16, via the method `_common_subdomains`. Then the addition is performed at the level of the restrictions, in line 17. The rest of the code is simply the set up of the vector field object containing the result. Recursively, the addition performed in line 17 will reach a level at which the domains are parallelizable. Then a different method `_add_()`, will be involved, as we can check on `vU`:

```
In [76]: vU._add_??
```

```
1  def _add_(self, other):
2      """
3      Tensor addition.
4
5      INPUT:
6
7      - ``other`` -- a tensor, of the same type as ``self``
8
9      OUTPUT:
10
11     - the tensor resulting from the addition of ``self`` and ``other``
12     ...
13     """
14     # No need for consistency check since self and other are guaranted
15     # to belong to the same tensor module
16     basis = self.common_basis(other)
17     if basis is None:
18         raise ValueError("no common basis for the addition")
19     comp_result = self._components[basis] + other._components[basis]
20     result = self._fmodule.tensor_from_comp(self._tensor_type, comp_result)
21     if self._name is not None and other._name is not None:
22         result._name = self._name + '+' + other._name
23     if self._latex_name is not None and other._latex_name is not None:
24         result._latex_name = self._latex_name + '+' + other._latex_name
25     return result
26 File:  .../site-packages/sage/tensor/modules/free_module_tensor.py
```

From line 26, we see that this method `_add_()` is implemented at the level of tensors on free modules, i.e. in the class `FreeModuleTensor`,[23] from which `VectorFieldParal` inherits (cf. the diagram in Fig. 3.2). Here the free module is clearly $\mathfrak{X}(U)$. The addition amounts to adding the components in a basis of the free module in which both operands have known components. Such a basis is returned by the method `common_basis` invoked in line 16. If necessary, this method can use change-of-basis formulas to compute the components of `self` or `other` in a common basis. The addition of the components in the found basis is performed in line 19. It involves the method `__add__()` of class `Components`; we can examine the corresponding code via the object `vUc` since the latter has been defined above as `vUc = vU._components[eU]`, i.e. `vUc` represents the set of components of the vector field $\boldsymbol{v}|_U$ in the basis `eU` $= (\partial/\partial x, \partial/\partial y)$ of $\mathfrak{X}(U)$:

```
In [77]: vUc.__add__??
```

```
1  def __add__(self, other):
2      """
3      Component addition.
4
5      INPUT:
6
7      - ``other`` -- components of the same number of indices and defined
```

[23] http://doc.sagemath.org/html/en/reference/tensor_free_modules/sage/tensor/modules/free_module_tensor.html

```
8         on the same frame as ``self``
9
10      OUTPUT:
11
12      - components resulting from the addition of ``self`` and ``other``
13      ...
14      """
15      ...
16      result = self.copy()
17      nproc = Parallelism().get('tensor')
18      if nproc != 1 :
19          # Parallel computation
20          ...
21      else:
22          # Sequential computation
23          for ind, val in other._comp.items():
24              result[[ind]] += val
25      return result
26  File:   .../site-packages/sage/tensor/modules/comp.py
```

First of all, we note from line 26 that this is not the method `__add__()` of class `Element`, as it was for `VectorField` and `VectorFieldParal`, but instead the method `__add__()` implemented in class `Components`. This is because `Components` is a *technical* class, as opposed to the *mathematical* classes `VectorField` and `DiffScalarField`; therefore it does not inherits from `Element`, but only from the base class `SageObject`, which does not implement any addition. We note from lines 17-19 that the computation of the components can be done in parallel on more that one CPU core if user has turned on parallelization.[24] Focusing on the sequential code (lines 23-24), we see that the addition is performed component by component. Note that this addition is that of scalar fields, as discussed in Sec. 3.3, since each component being an element of $C^\infty(U)$, the base ring of $\mathfrak{X}(U)$.

3.6. **Action of vector fields on scalar fields.** The action of $\boldsymbol{v}$ on f is defined pointwise by considering $\boldsymbol{v}$ at each point $p \in M$ as a derivation (the very definition of a tangent vector, cf. Sec. 2.1); the result is then a scalar field $\boldsymbol{v}(f)$ on M:

In [78]:
```
vf = v(f)
vf
```

Out[78]: $v\,(f)$

In [79]:
```
vf.display()
```

Out[79]:
$$\begin{array}{lccl} v\,(f): & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x,y) & \longmapsto & \frac{2\left(2\,y^3+2\left(x^2+1\right)y-x\right)}{x^6+y^6+3\,(x^2+1)y^4+3\,x^4+3\,(x^4+2\,x^2+1)y^2+3\,x^2+1} \\ \text{on } V: & (x',y') & \longmapsto & -\frac{2\left(x'^5+2\,x'^3y'^2+x'y'^4-2\,y'^5-2\left(2\,x'^2+1\right)y'^3-2\left(x'^4+x'^2\right)y'\right)}{x'^6+y'^6+3\,(x'^2+1)y'^4+3\,x'^4+3\,(x'^4+2\,x'^2+1)y'^2+3\,x'^2+1} \end{array}$$

[24] This is done with the command `Parallelism().set(nproc=8)` (for 8 threads); many examples of parallelized computations are presented at `https://sagemanifolds.obspm.fr/examples.html`.

CHAPTER 4

Tensor fields

Contents

1. Introduction

Having presented vector fields in Chap. 3, we move now to more general tensor fields. We keep the same example manifold, $M = \mathbb{S}^2$, as in Chap. 2 and 3.

2. Differential forms

Let us continue with the same example notebook as that considered in Chap. 3. There, we had introduced f as a scalar field on the 2-dimensional manifold $M = \mathbb{S}^2$ (cf. Sec. 2.2). The differential of f is a 1-form on M:

In [80]:
```
df = f.differential()
df
```

Out[80]: $\mathrm{d}f$

In [81]:
```
print(df)
```

```
1-form df on the 2-dimensional differentiable manifold M
```

A 1-form is actually a tensor field of type $(0, 1)$:

In [82]:
```
df.tensor_type()
```

Out[82]: $(0, 1)$

while a vector field is a tensor field of type $(1, 0)$:

In [83]:
```
v.tensor_type()
```

Out[83]: $(1, 0)$

Specific 1-forms are those forming the dual basis (coframe) of a given vector frame: for instance for the vector frame `eU` $= (\partial/\partial x, \partial/\partial y)$ on U, considered as a basis of the free $C^\infty(U)$-module $\mathfrak{X}(U)$, we have:

In [84]:
```
eU.dual_basis()
```

Out[84]: $(U, (\mathrm{d}x, \mathrm{d}y))$

In [85]:
```
print(eU.dual_basis()[0])
```

```
1-form dx on the Open subset U of the 2-dimensional differentiable manifold M
```

Since `eU` is the default frame on M, the default display of $\mathrm{d}f$ is performed in terms of `eU`'s coframe:

In [86]:
```
df.display()
```

Out[86]:
$$\mathrm{d}f = \left(-\frac{2\,x}{x^4 + y^4 + 2\,(x^2+1)y^2 + 2\,x^2 + 1}\right)\mathrm{d}x + \left(-\frac{2\,y}{x^4 + y^4 + 2\,(x^2+1)y^2 + 2\,x^2 + 1}\right)\mathrm{d}y$$

We may check that in this basis, the components of $\mathrm{d}f|_U$ are nothing but the partial derivatives of the coordinate expression of f with respect to coordinates (x, y):

In [87]:
```
df[0] == diff(f.expr(), x)
```

Out[87]:
```
True
```

In [88]:
```
df[1] == diff(f.expr(), y)
```

Out[88]:
```
True
```

In the coframe associated with `eV` $= (\partial/\partial x', \partial/\partial y')$:

In [89]:
```
df.display(eV)
```

Out[89]:
$$\mathrm{d}f = \left(\frac{2\,x'}{x'^4 + y'^4 + 2\,(x'^2+1)y'^2 + 2\,x'^2 + 1}\right)\mathrm{d}x' + \left(\frac{2\,y'}{x'^4 + y'^4 + 2\,(x'^2+1)y'^2 + 2\,x'^2 + 1}\right)\mathrm{d}y'$$

Since `eV` is not the default vector frame on M and `XV` $= (V, (x', y'))$ is not the default chart on M, we get the individual components by specifying both `eV` and `XV`, in addition to the index, in the square-bracket operator:

In [90]:
```
df[eV,0,XV]
```

Out[90]:
$$\frac{2\,x'}{x'^4 + y'^4 + 2\,(x'^2+1)y'^2 + 2\,x'^2 + 1}$$

We may then check that the components in the frame `eV` are the partial derivatives with respect to the coordinates `xp` $= x'$ and `yp` $= y'$ of the chart `XV`:

In [91]:
```
df[eV,0,XV] == diff(f.expr(XV), xp)
```

Out[91]:
```
True
```

In [92]:
```
df[eV,1,XV] == diff(f.expr(XV), yp)
```

Out[92]:
```
True
```

The parent of $\mathrm{d}f$ is the set $\Omega^1(M)$ of all 1-forms on M, considered as a $C^\infty(M)$-module:

In [93]:
```
print(df.parent())
df.parent()
```

```
Module Omega^1(M) of 1-forms on the 2-dimensional differentiable manifold M
```

Out[93]: $\Omega^1(M)$

In [94]:
```
df.parent().base_ring()
```

Out[94]: $C^\infty(M)$

This module is actually the dual of the vector-field module $\mathfrak{X}(M)$, which is represented by the Python object `YM` (cf. Sec. 3.3):

In [95]:
```
YM.dual()
```

Out[95]: $\Omega^1(M)$

Consequently, a 1-form acts on vector fields, yielding an element of $C^\infty(M)$, i.e. a scalar field:

In [96]:
```
print(df(v))
```

```
Scalar field df(v) on the 2-dimensional differentiable manifold M
```

This scalar field is nothing but the result of the action of $\boldsymbol{v}$ on f discussed in Sec. 3.6:

In [97]:
```
df(v) == v(f)
```

Out[97]: `True`

3. More general tensor fields

We construct a tensor of type $(1,1)$ by taking the tensor product $\boldsymbol{v} \otimes \mathrm{d}f$:

In [98]:
```
t = v * df
t
```

Out[98]: Tensor field of type (1,1) on the 2-dimensional differentiable manifold M

In [99]:
```
t.display()
```

Out[99]:
$$v \otimes \mathrm{d}f = \left(-\frac{2\,x}{x^6+y^6+3\,(x^2+1)y^4+3\,x^4+3\,(x^4+2\,x^2+1)y^2+3\,x^2+1}\right)\frac{\partial}{\partial x} \otimes \mathrm{d}x + \left(-\frac{2\,y}{x^6+y^6+3\,(x^2+1)y^4+3\,x^4+3\,(x^4+2\,x^2+1)y^2+3\,x^2+1}\right)\frac{\partial}{\partial x} \otimes \mathrm{d}y + \left(\frac{4\,x}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1}\right)\frac{\partial}{\partial y}\otimes\mathrm{d}x + \left(\frac{4\,y}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1}\right)\frac{\partial}{\partial y}\otimes\mathrm{d}y$$

In [100]:
```
t.display(eV)
```

Out[100]:
$$v\otimes \mathrm{d}f = \left(-\frac{2\left(x'^5-4\,x'^2y'^3-x'y'^4-4\left(x'^4+x'^2\right)y'\right)}{x'^6+y'^6+3\left(x'^2+1\right)y'^4+3\,x'^4+3\left(x'^4+2\,x'^2+1\right)y'^2+3\,x'^2+1}\right)\frac{\partial}{\partial x'}\otimes \mathrm{d}x' + \left(-\frac{2\left(x'^4y'-4\,x'y'^4-y'^5-4\left(x'^3+x'\right)y'^2\right)}{x'^6+y'^6+3\left(x'^2+1\right)y'^4+3\,x'^4+3\left(x'^4+2\,x'^2+1\right)y'^2+3\,x'^2+1}\right)\frac{\partial}{\partial x'}\otimes \mathrm{d}y' + \left(-\frac{4\left(x'^5+x'^4y'+x'^2y'^3-x'y'^4+x'^3-x'y'^2\right)}{x'^6+y'^6+3\left(x'^2+1\right)y'^4+3\,x'^4+3\left(x'^4+2\,x'^2+1\right)y'^2+3\,x'^2+1}\right)\frac{\partial}{\partial y'}\otimes \mathrm{d}x' + \left(-\frac{4\left(x'^3y'^2+x'y'^4-y'^5-y'^3+\left(x'^4+x'^2\right)y'\right)}{x'^6+y'^6+3\left(x'^2+1\right)y'^4+3\,x'^4+3\left(x'^4+2\,x'^2+1\right)y'^2+3\,x'^2+1}\right)\frac{\partial}{\partial y'}\otimes \mathrm{d}y'$$

We can use the method `display_comp()` for a display component by component:

In [101]:
```
t.display_comp()
```

Out[101]:
$$\begin{aligned} v\otimes \mathrm{d}f\,{}^{x}{}_{x} &= -\frac{2\,x}{x^6+y^6+3\,(x^2+1)y^4+3\,x^4+3\,(x^4+2\,x^2+1)y^2+3\,x^2+1}\\ v\otimes \mathrm{d}f\,{}^{x}{}_{y} &= -\frac{2\,y}{x^6+y^6+3\,(x^2+1)y^4+3\,x^4+3\,(x^4+2\,x^2+1)y^2+3\,x^2+1}\\ v\otimes \mathrm{d}f\,{}^{y}{}_{x} &= \frac{4\,x}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1}\\ v\otimes \mathrm{d}f\,{}^{y}{}_{y} &= \frac{4\,y}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1} \end{aligned}$$

The parent of t is the set $\mathcal{T}^{(1,1)}(M)$ of all type-$(1,1)$ tensor fields on M, considered as a $C^\infty(M)$-module:

In [102]:
```
print(t.parent())
t.parent()
```

```
Module T^(1,1)(M) of type-(1,1) tensors fields on the 2-dimensional
differentiable manifold M
```

Out[102]: $\mathcal{T}^{(1,1)}\left(M\right)$

In [103]:
```
t.parent().base_ring()
```

Out[103]: $C^\infty\left(M\right)$

As for vector fields, since M is not parallelizable, the $C^\infty(M)$-module $\mathcal{T}^{(1,1)}(M)$ is not free and the tensor fields are described by their restrictions to parallelizable subdomains:

In [104]:
```
t._restrictions
```

Out[104]: $\{V : v\otimes \mathrm{d}f, U : v\otimes \mathrm{d}f\}$

These restrictions form free modules:

In [105]:
```
print(t._restrictions[U].parent())
```

```
Free module T^(1,1)(U) of type-(1,1) tensors fields on the Open subset U of
the 2-dimensional differentiable manifold M
```

In [106]:
```
t._restrictions[U].parent().base_ring()
```

Out[106]: $C^\infty(U)$

4. Riemannian metric

4.1. **Defining a metric.** The standard metric on $M = \mathbb{S}^2$ is that induced by the Euclidean metric of $\mathbb{R}^3$. Let us start by defining the latter:

In [107]:
```
h = R3.metric('h')
h[0,0], h[1,1], h[2, 2] = 1, 1, 1
h.display()
```

Out[107]: $h = \mathrm{d}X \otimes \mathrm{d}X + \mathrm{d}Y \otimes \mathrm{d}Y + \mathrm{d}Z \otimes \mathrm{d}Z$

The metric g on M is the pullback of h associated with the embedding Φ introduced in Sec. 2.2:

In [108]:
```
g = M.metric('g')
g.set( Phi.pullback(h) )
print(g)
```

```
Riemannian metric g on the 2-dimensional differentiable manifold M
```

Note that we could have defined g intrinsically, i.e. by providing its components in the two vector frames `eU` and `eV`, as we did for the metric h on $\mathbb{R}^3$. Instead, we have chosen to get it as the pullback by Φ of h, as an example of pullback associated with some differential map.

The metric is a symmetric tensor field of type (0,2):

In [109]:
```
g.tensor_type()
```

Out[109]: $(0, 2)$

The expression of the metric in terms of the default frame on M (`eU`):

In [110]:
```
g.display()
```

Out[110]:
$$g = \left(\frac{4}{x^4 + y^4 + 2\,(x^2+1)y^2 + 2\,x^2 + 1}\right) \mathrm{d}x \otimes \mathrm{d}x + \left(\frac{4}{x^4 + y^4 + 2\,(x^2+1)y^2 + 2\,x^2 + 1}\right) \mathrm{d}y \otimes \mathrm{d}y$$

We may factorize the metric components to get a better display:

In [111]:
```
g[0,0].factor(); g[1,1].factor()
```

Out[111]: $\dfrac{4}{\left(x^2 + y^2 + 1\right)^2}$

In [112]:
```
g.display()
```

Out[112]: $g = \dfrac{4}{\left(x^2 + y^2 + 1\right)^2} \mathrm{d}x \otimes \mathrm{d}x + \dfrac{4}{\left(x^2 + y^2 + 1\right)^2} \mathrm{d}y \otimes \mathrm{d}y$

A matrix view of the components of g in the manifold's default frame:

In [113]:
```
g[:]
```

Out[113]:
$$\begin{pmatrix} \frac{4}{(x^2+y^2+1)^2} & 0 \\ 0 & \frac{4}{(x^2+y^2+1)^2} \end{pmatrix}$$

Display in terms of the vector frame $(V, (\partial_{x'}, \partial_{y'}))$:

In [114]:
```
g.display(eV)
```

Out[114]:
$$g = \left(\frac{4}{x'^4 + y'^4 + 2\left(x'^2+1\right)y'^2 + 2\,x'^2 + 1}\right) \mathrm{d}x' \otimes \mathrm{d}x' + \left(\frac{4}{x'^4 + y'^4 + 2\left(x'^2+1\right)y'^2 + 2\,x'^2 + 1}\right) \mathrm{d}y' \otimes \mathrm{d}y'$$

The metric acts on vector field pairs, resulting in a scalar field:

In [115]:
```
print(g(v,v))
```

```
Scalar field g(v,v) on the 2-dimensional differentiable manifold M
```

In [116]:
```
g(v,v).parent()
```

Out[116]:
$C^\infty(M)$

In [117]:
```
g(v,v).display()
```

Out[117]:
$$\begin{array}{lcll} g(v,v): & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x,y) & \longmapsto & \frac{4\left(4\,x^4+4\,y^4+8\left(x^2+1\right)y^2+8\,x^2+5\right)}{x^8+y^8+4\left(x^2+1\right)y^6+4\,x^6+6\left(x^4+2\,x^2+1\right)y^4+6\,x^4+4\left(x^6+3\,x^4+3\,x^2+1\right)y^2+4\,x^2+1} \\ \text{on } V: & (x',y') & \longmapsto & \frac{4\left(5\,x'^8+5\,y'^8+4\left(5\,x'^2+2\right)y'^6+8\,x'^6+2\left(15\,x'^4+12\,x'^2+2\right)y'^4+4\,x'^4+4\left(5\,x'^6+6\,x'^4+2\,x'^2\right.\right.}{x'^8+y'^8+4\left(x'^2+1\right)y'^6+4\,x'^6+6\left(x'^4+2\,x'^2+1\right)y'^4+6\,x'^4+4\left(x'^6+3\,x'^4+3\,x'^2+1\right)y'^2+4\,x'} \end{array}$$

4.2. **Levi-Civita connection.** The Levi-Civita connection associated with the metric g is

In [118]:
```
nab = g.connection()
print(nab)
nab
```

```
Levi-Civita connection nabla_g associated with the Riemannian metric g on
the 2-dimensional differentiable manifold M
```

Out[118]:
∇_g

The nonzero Christoffel symbols of g (skipping those that can be deduced by symmetry on the last two indices) w.r.t. the chart `XU`:

In [119]:
```
g.christoffel_symbols_display(chart=XU)
```

Out[119]:
$$\begin{aligned} \Gamma^{x}{}_{x\,x} &= -\frac{2\,x}{x^2+y^2+1} \\ \Gamma^{x}{}_{x\,y} &= -\frac{2\,y}{x^2+y^2+1} \\ \Gamma^{x}{}_{y\,y} &= \frac{2\,x}{x^2+y^2+1} \\ \Gamma^{y}{}_{x\,x} &= \frac{2\,y}{x^2+y^2+1} \\ \Gamma^{y}{}_{x\,y} &= -\frac{2\,x}{x^2+y^2+1} \\ \Gamma^{y}{}_{y\,y} &= -\frac{2\,y}{x^2+y^2+1} \end{aligned}$$

∇_g acting on the vector field $\boldsymbol{v}$:

In [120]:
```
Dv = nab(v)
print(Dv)
```

```
Tensor field nabla_g(v) of type (1,1) on the 2-dimensional differentiable
manifold M
```

In [121]:
```
Dv.display()
```

Out[121]:
$$\nabla_g v = \left(\frac{4\left(y^3+\left(x^2+1\right)y-x\right)}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial x}\otimes \mathrm{d}x + \left(-\frac{4\left(x^3+xy^2+x+y\right)}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial x}\otimes \mathrm{d}y + \left(\frac{2\left(2\,x^3+2\,xy^2+2\,x+y\right)}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial y}\otimes \mathrm{d}x + \left(\frac{2\left(2\,y^3+2\left(x^2+1\right)y-x\right)}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial y}\otimes \mathrm{d}y$$

4.3. **Curvature.** The Riemann curvature tensor of the metric g is

In [122]:
```
Riem = g.riemann()
print(Riem)
Riem.display()
```

```
Tensor field Riem(g) of type (1,3) on the 2-dimensional differentiable
manifold M
```

Out[122]:
$$\mathrm{Riem}\,(g) = \left(\frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial x}\otimes \mathrm{d}y\otimes \mathrm{d}x\otimes \mathrm{d}y + \left(-\frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial x}\otimes \mathrm{d}y\otimes \mathrm{d}y\otimes \mathrm{d}x + \left(-\frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial y}\otimes \mathrm{d}x\otimes \mathrm{d}x\otimes \mathrm{d}y + \left(\frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\right)\frac{\partial}{\partial y}\otimes \mathrm{d}x\otimes \mathrm{d}y\otimes \mathrm{d}x$$

The components of the Riemann tensor in the default frame on M are

In [123]:
```
Riem.display_comp()
```

Out[123]:
$$\begin{aligned}
\mathrm{Riem}\,(g)^{x}{}_{y\,x\,y} &= \frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\\
\mathrm{Riem}\,(g)^{x}{}_{y\,y\,x} &= -\frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\\
\mathrm{Riem}\,(g)^{y}{}_{x\,x\,y} &= -\frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}\\
\mathrm{Riem}\,(g)^{y}{}_{x\,y\,x} &= \frac{4}{x^4+y^4+2\left(x^2+1\right)y^2+2\,x^2+1}
\end{aligned}$$

The parent of the Riemann tensor is the $C^\infty(M)$-module of type-(1,3) tensor fields on M:

In [124]:
```
print(Riem.parent())
```

```
Module T^(1,3)(M) of type-(1,3) tensors fields on the 2-dimensional
differentiable manifold M
```

The Riemann tensor is antisymmetric on its two last indices (i.e. the indices at position 2 and 3, the first index being at position 0):

In [125]:
```
Riem.symmetries()
```

Out[125]:
```
no symmetry; antisymmetry: (2, 3)
```

The Riemann tensor of the Euclidean metric h on $\mathbb{R}^3$ is identically zero, i.e. h is a flat metric:

In [126]:
```
h.riemann().display()
```

Out[126]: $\mathrm{Riem}\,(h) = 0$

The Ricci tensor is

In [127]:
```
Ric = g.ricci()
Ric.display()
```

Out[127]: $$\mathrm{Ric}\,(g) = \left(\frac{4}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1}\right) \mathrm{d}x\otimes\mathrm{d}x + \left(\frac{4}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1}\right) \mathrm{d}y\otimes\mathrm{d}y$$

while the Ricci scalar is

In [128]:
```
R = g.ricci_scalar()
R.display()
```

Out[128]:
$$\begin{array}{lccc} \mathrm{r}\,(g): & M & \longrightarrow & \mathbb{R} \\ \text{on } U: & (x,y) & \longmapsto & 2 \\ \text{on } V: & (x',y') & \longmapsto & 2 \end{array}$$

We recover the fact that $(\mathbb{S}^2, g)$ is a Riemannian manifold of constant positive curvature.

In dimension 2, the Riemann curvature tensor is entirely determined by the Ricci scalar R according to

$$R^i{}_{jlk} = \frac{R}{2}\left(\delta^i{}_k g_{jl} - \delta^i{}_l g_{jk}\right) \tag{4.1}$$

Let us check this formula here, under the form $R^i{}_{jlk} = -R g_{j[k}\delta^i{}_{l]}$:

In [129]:
```
delta = M.tangent_identity_field()
Riem == - R*(g*delta).antisymmetrize(2,3)
```

Out[129]: `True`

Similarly the relation $\mathrm{Ric} = (R/2)\, g$ must hold:

In [130]:
```
Ric == (R/2)*g
```

Out[130]: `True`

4.4. **Volume form.** The ***volume form*** (or ***Levi-Civita tensor***) associated with the metric g and for which the vector frame (∂_x, ∂_y) is right-handed is the following 2-form:

In [131]:
```
eps = g.volume_form()
print(eps)
eps.display()
```

Out[131]: $$\epsilon_g = \left(\frac{4}{x^4+y^4+2\,(x^2+1)y^2+2\,x^2+1}\right) \mathrm{d}x\wedge\mathrm{d}y$$

The exterior derivative of ϵ_g is a 3-form:

In [132]:
```
print(eps.exterior_derivative())
```

```
3-form deps_g on the 2-dimensional differentiable manifold M
```

Of course, since the dimension of M is 2, all 3-forms vanish identically:

In [133]:
```
eps.exterior_derivative().display()
```

Out[133]: $\mathrm{d}\epsilon_g = 0$

CHAPTER 5

Conclusion and perspectives

We have presented some aspects of symbolic tensor calculus as implemented in SageMath. The implementation is independent of the symbolic backend (i.e. the tool used to performed symbolic calculus on coordinate representations of scalar fields), the latter being involved only in the last stage of the diagram shown in Fig. 3.3.

The implementation has been performed via the SageManifolds project, the home page of which we refer to for details and material complementary to what has been shown here (in particular many more examples):

`https://sagemanifolds.obspm.fr/`

This project resulted in approximately 85,000 lines of Python code (including comments and doctests), which have been submitted to SageMath community as a sequence of ~ 50 tickets[25] at the time of this writing (October 2018), the first ticket having been accepted in March 2015. These tickets have been written and reviewed by a dozen of contributors.[26] As a result, all code is fully included in SageMath 8.4 and does not require any separate installation. The following features have been already implemented:

- differentiable manifolds: tangent spaces, vector frames, tensor fields, curves, pullback and pushforward operators;
- standard tensor calculus (tensor product, contraction, symmetrization, etc.), even on non-parallelizable manifolds;
- all monoterm tensor symmetries taken into account;
- Lie derivatives of tensor fields;
- differential forms: exterior and interior products, exterior derivative, Hodge duality;
- multivector fields: exterior and interior products, Schouten-Nijenhuis bracket;
- affine connections (curvature, torsion);
- pseudo-Riemannian metrics;
- computation of geodesics (numerical integration via SageMath/GSL);
- some plotting capabilities (charts, points, curves, vector fields);
- extrinsic geometry of pseudo-Riemannian submanifolds;
- parallelization (on tensor components) of CPU demanding computations, via the Python library `multiprocessing`;
- the possibility to use SymPy as the symbolic backend, instead of SageMath's default, which is Pynac (with Maxima for simplifications).

Only a subset of the above functionalities have been presented in these lectures. In particular, the exterior calculus on differential forms and multivector fields has not been touched, nor the computation of geodesics.

The SageManifolds project is still ongoing and future prospects include

- adding more symbolic backends (Giac, FriCAS, ...);
- computing integrals on submanifolds;
- adding more plotting capabilities;
- introducing new functionalities: symplectic forms, fibre bundles, spinors, variational calculus, etc.;
- connecting with numerical relativity: using SageMath to explore numerically-generated spacetimes; this will be done by introducing *numerical* backends, instead of *symbolic* ones, in the last stage of the Fig. 3.3 diagram.

[25] Cf. the meta-ticket `https://trac.sagemath.org/ticket/18528`.

[26] Cf. the list at `https://sagemanifolds.obspm.fr/authors.html`.

Manifold tutorial

This notebook provides a short introduction to differentiable manifolds in SageMath. The tools described below have been implemented through the SageManifolds project.

If you are new to SageMath, you may first take a look at this first contact tutorial.

This notebook is valid for version 9.2 or higher of SageMath:

In [1]:
```
version()
```

Out[1]:
```
'SageMath version 10.5, Release Date: 2024-12-04'
```

First we set up the notebook to display mathematical objects using LaTeX rendering:

In [2]:
```
%display latex
```

Defining a manifold

As an example let us define a differentiable manifold of dimension 3 over $\mathbb{R}$:

In [3]:
```
M = Manifold(3, 'M', latex_name=r'\mathcal{M}', start_index=1)
```

- The first argument, `3`, is the manifold dimension; it can be any positive integer.
- The second argument, `'M'`, is a string defining the manifold's name; it may be different from the symbol set on the left-hand side of the = sign (here `M`): the latter stands for the Python variable that refers to the manifold object in the computer memory, while the string `'M'` is the mathematical symbol chosen for the manifold.
- The optional argument `latex_name=r'\mathcal{M}'` sets the LaTeX symbol to display the manifold. Note the letter 'r' in front on the first quote: it indicates that the string is a *raw* one, so that the backslash character in `\mathcal` is considered as an ordinary character (otherwise, the backslash is used to escape some special characters). If the argument `latex_name` is not provided by the user, it is set to the string used as the second argument (here `'M'`)
- The optional argument `start_index=1` defines the range of indices to be used for tensor components on the manifold: setting it to 1 means that indices will range in $\{1, 2, 3\}$. The default value is `start_index=0`.

Note that the default base field is $\mathbb{R}$. If we would have used the optional argument `field='complex'`, we would have defined a manifold over $\mathbb{C}$. See the list of all options for more details.

If we ask for `M`, it is displayed via its LaTeX symbol:

In [4]:
```
M
```

Out[4]: $\mathcal{M}$

If we use the function `print()` instead, we get a short description of the object:

In [5]:
```
print(M)
```

```
3-dimensional differentiable manifold M
```

Via the function `type()`, we get the type of the Python object corresponding to M (here the Python class `DifferentiableManifold_with_category`):

In [6]:
```
print(type(M))
```

```
<class 'sage.manifolds.differentiable.manifold.DifferentiableManifold_with_categor
y'>
```

We may also ask for the category of `M` and see that it is the category of smooth manifolds over $\mathbb{R}$:

In [7]:
```
category(M)
```

Out[7]: $\mathbf{Smooth}_{\mathbf{R}}$

The indices on the manifold are generated by the method `irange()`, to be used in loops:

In [8]:
```
[i for i in M.irange()]
```

Out[8]: $[1, 2, 3]$

If the parameter `start_index` had not been specified, the default range of the indices would have been $\{0, 1, 2\}$ instead:

In [9]:
```
M0 = Manifold(3, 'M', latex_name=r'\mathcal{M}')
[i for i in M0.irange()]
```

Out[9]: $[0, 1, 2]$

Defining a chart on the manifold

Let us assume that the manifold $\mathcal{M}$ can be covered by a single chart (other cases are discussed below); the chart is declared as follows:

In [10]:
```
X.<x,y,z> = M.chart()
```

The writing `.<x,y,z>` in the left-hand side means that the Python variables `x`, `y` and `z` are set to the three coordinates of the chart. This allows one to refer subsequently to the coordinates by their names.

In this example, the function `chart()` has no arguments, which implies that the coordinate symbols will be `x`, `y` and `z` (i.e. exactly the characters set in the `<...>` operator) and that each coordinate range is $(-\infty, +\infty)$. For other cases, an argument must be passed to `chart()` to specify the coordinate symbols and range, as well as the LaTeX symbol of a coordinate if the latter is different from the coordinate name (an example will be provided below).

The chart is displayed as a pair formed by the open set covered by it (here the whole manifold) and the coordinates:

In [11]:
```
print(X)
```

```
Chart (M, (x, y, z))
```

In [12]:
```
X
```

Out[12]: $(\mathcal{M}, (x, y, z))$

The coordinates can be accessed individually, by means of their indices, following the convention defined by `start_index=1` in the manifold's definition:

In [13]:
```
X[1]
```

Out[13]: x

In [14]:
```
X[2]
```

Out[14]: y

In [15]:
```
X[3]
```

Out[15]: z

The full set of coordinates is obtained by means of the operator `[:]` :

In [16]:
```
X[:]
```

Out[16]: (x, y, z)

Thanks to the operator `<x,y,z>` used in the chart declaration, each coordinate can be accessed directly via its name:

In [17]:
```
z is X[3]
```

Out[17]: True

Coordinates are SageMath symbolic expressions:

In [18]:
```
type(z)
```

Out[18]:
```
<class 'sage.symbolic.expression.Expression'>
```

Functions of the chart coordinates

Real-valued functions of the chart coordinates (mathematically speaking, *functions defined on the chart codomain*) are generated via the method `function()` acting on the chart:

In [19]:
```
f = X.function(x+y^2+z^3)
f
```

Out[19]: $z^3 + y^2 + x$

In [20]:
```
f.display()
```

Out[20]: $(x, y, z) \mapsto z^3 + y^2 + x$

In [21]:
```
f(1,2,3)
```

Out[21]: 32

They belong to the class `ChartFunction` (actually the subclass `ChartFunctionRing_with_category.element_class`):

In [22]:
```
print(type(f))
```

```
<class 'sage.manifolds.chart_func.ChartFunctionRing_with_category.element_class'>
```

and differ from SageMath standard symbolic functions by automatic simplifications in all operations. For instance, adding the two symbolic functions

In [23]:
```
f0(x,y,z) = cos(x)^2; g0(x,y,z) = sin(x)^2
```

results in

In [24]:
```
f0 + g0
```

Out[24]: $(x, y, z) \mapsto \cos\left(x\right)^2 + \sin\left(x\right)^2$

while the sum of the corresponding functions in the class `ChartFunction` is automatically simplified:

In [25]:
```
f1 = X.function(cos(x)^2); g1 = X.function(sin(x)^2)
f1 + g1
```

Out[25]: 1

To get the same output with symbolic functions, one has to invoke the method `simplify_trig()`:

In [26]:
```
(f0 + g0).simplify_trig()
```

Out[26]: $(x, y, z) \mapsto 1$

Another difference regards the display; if we ask for the symbolic function `f0`, we get

In [27]:
```
f0
```

Out[27]: $(x, y, z) \mapsto \cos\left(x\right)^2$

while if we ask for the chart function `f1`, we get only the coordinate expression:

In [28]:
```
f1
```

Out[28]: $\cos\left(x\right)^2$

To get an output similar to that of `f0`, one should call the method `display()`:

In [29]:
```
f1.display()
```

Out[29]: $(x, y, z) \mapsto \cos\left(x\right)^2$

Note that the method `expr()` returns the underlying symbolic expression:

In [30]:
```
f1.expr()
```

Out[30]: $\cos\left(x\right)^2$

In [31]:
```
print(type(f1.expr()))
```

```
<class 'sage.symbolic.expression.Expression'>
```

Introducing a second chart on the manifold

Let us first consider an open subset of $\mathcal{M}$, for instance the complement U of the region defined by $\{y = 0, x \geq 0\}$ (note that `(y!=0, x<0)` stands for $y \neq 0$ OR $x < 0$; the condition $y \neq 0$ AND $x < 0$ would have been written `[y!=0, x<0]` instead):

In [32]:
```
U = M.open_subset('U', coord_def={X: (y!=0, x<0)})
```

Let us call `X_U` the restriction of the chart `X` to the open subset U:

In [33]:
```
X_U = X.restrict(U)
```

```
X_U
```

Out[33]: $(U, (x, y, z))$

We introduce another chart on U, with spherical-type coordinates (r, θ, ϕ):

In [34]:
```
Y.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
Y
```

Out[34]: $(U, (r, \theta, \phi))$

The method `chart()` is now used with an argument; it is a string, which contains specific LaTeX symbols, hence the prefix 'r' to it (for *raw* string). It also contains the coordinate ranges, since they are different from the default value, which is $(-\infty, +\infty)$. For a given coordinate, the various fields are separated by the character ':' and a space character separates the coordinates. Note that for the coordinate r, there are only two fields, since the LaTeX symbol has not to be specified. The LaTeX symbols are used for the outputs:

In [35]:
```
th, ph
```

Out[35]: (θ, ϕ)

In [36]:
```
Y[2], Y[3]
```

Out[36]: (θ, ϕ)

The declared coordinate ranges are now known to Sage, as we may check by means of the command `assumptions()`:

In [37]:
```
assumptions()
```

Out[37]: $[\mathtt{x\ is\ real}, \mathtt{y\ is\ real}, \mathtt{z\ is\ real}, \mathtt{r\ is\ real}, r > 0, \mathtt{th\ is\ real}, \theta > 0, \theta < \pi, \mathtt{ph\ is\ real}$

They are used in simplifications:

In [38]:
```
simplify(abs(r))
```

Out[38]: r

In [39]:
```
simplify(abs(x)) # no simplification occurs since x can take any value in R
```

Out[39]: $|x|$

After having been declared, the chart `Y` can be fully specified by its relation to the chart `X_U`, via a transition map:

In [40]:
```
transit_Y_to_X = Y.transition_map(X_U, [r*sin(th)*cos(ph), r*sin(th)*sin(ph),
                                        r*cos(th)])
transit_Y_to_X
```

Out[40]: $(U, (r, \theta, \phi)) \to (U, (x, y, z))$

In [41]:
```
transit_Y_to_X.display()
```

Out[41]:
$$\begin{cases} x &= r\cos(\phi)\sin(\theta) \\ y &= r\sin(\phi)\sin(\theta) \\ z &= r\cos(\theta) \end{cases}$$

The inverse of the transition map can be specified by means of the method `set_inverse()`:

In [42]:
```
transit_Y_to_X.set_inverse(sqrt(x^2+y^2+z^2), atan2(sqrt(x^2+y^2),z),
                           atan2(y, x))
```

```
Check of the inverse coordinate transformation:
  r == r  *passed*
  th == arctan2(r*sin(th), r*cos(th))  **failed**
  ph == arctan2(r*sin(ph)*sin(th), r*cos(ph)*sin(th))  **failed**
  x == x  *passed*
  y == y  *passed*
  z == z  *passed*
NB: a failed report can reflect a mere lack of simplification.
```

A check of the provided inverse is performed by composing it with the original transition map, on the left and on the right respectively. As indicated, the reported failure for `th` and `ph` is actually due to a lack of simplification of expressions involving `arctan2`.

We have then

In [43]:
```
transit_Y_to_X.inverse().display()
```

Out[43]:
$$\begin{cases} r &= \sqrt{x^2+y^2+z^2} \\ \theta &= \arctan\left(\sqrt{x^2+y^2}, z\right) \\ \phi &= \arctan(y, x) \end{cases}$$

At this stage, the manifold's **atlas** (the "user atlas", not the maximal atlas!) contains three charts:

In [44]:
```
M.atlas()
```

Out[44]: $[(\mathcal{M}, (x, y, z)), (U, (x, y, z)), (U, (r, \theta, \phi))]$

The first chart defined on the manifold is considered as the manifold's default chart (this can be changed by the method `set_default_chart()`):

In [45]:
```
M.default_chart()
```

Out[45]: $(\mathcal{M}, (x, y, z))$

Each open subset has its own atlas (since an open subset of a manifold is a manifold by itself):

In [46]:
```
U.atlas()
```

Out[46]: $[(U, (x, y, z)), (U, (r, \theta, \phi))]$

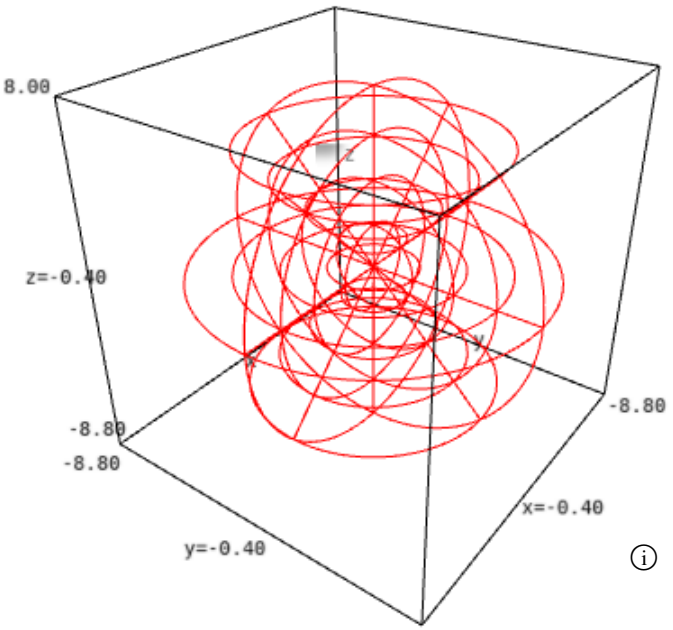
In [47]:
```
U.default_chart()
```

Out[47]: $(U, (x, y, z))$

We can draw the chart Y in terms of the chart X via the command `Y.plot(X)`, which shows the lines of constant coordinates from the Y chart in a "Cartesian frame" based on the X coordinates:

In [48]:
```
Y.plot(X)
```

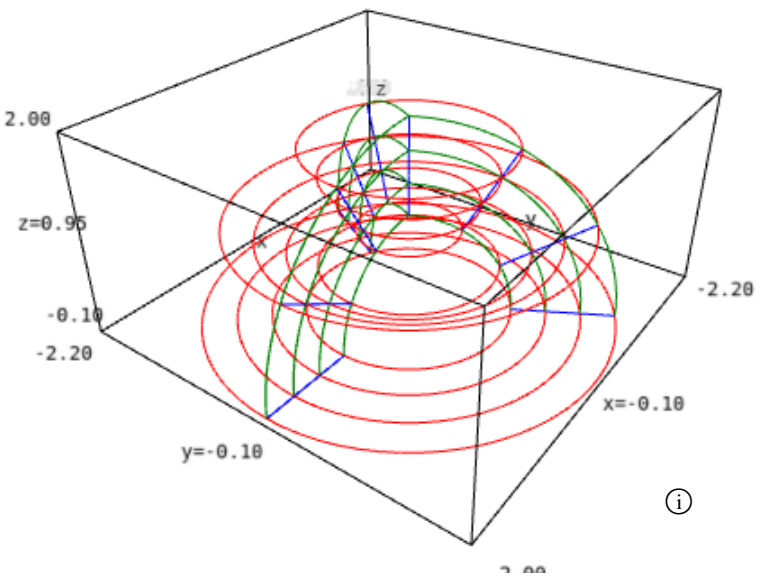
Out[48]:



The method `plot()` allows for many options, to control the number of coordinate lines to be drawn, their style and color, as well as the coordinate ranges (see the list of all options):

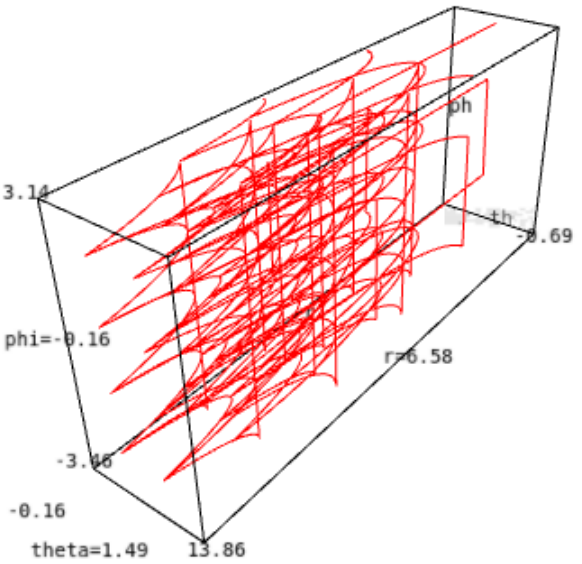
In [49]:
```
Y.plot(X, ranges={r:(1,2), th:(0,pi/2)}, number_values=4,
       color={r:'blue', th:'green', ph:'red'}, aspect_ratio=1)
```

Out[49]:



Conversely, the chart $X|_U$ can be plotted in terms of the chart Y (this is not possible for the whole chart X since its domain is larger than that of chart Y):

In [50]:
```
graph = X_U.plot(Y)
show(graph, axes_labels=['r','theta','phi'])
```



Points on the manifold

A point on $\mathcal{M}$ is defined by its coordinates in a given chart:

In [51]:
```
p = M.point((1,2,-1), chart=X, name='p')
print(p)
p
```

```
Point p on the 3-dimensional differentiable manifold M
```

Out[51]: p

Since $X = (\mathcal{M}, (x, y, z))$ is the manifold's default chart, its name can be omitted:

In [52]:
```
p = M.point((1,2,-1), name='p')
print(p)
p
```

```
Point p on the 3-dimensional differentiable manifold M
```

Out[52]: p

Of course, p belongs to $\mathcal{M}$:

In [53]:
```
p in M
```

Out[53]: True

It is also in U:

In [54]:
```
p in U
```

Out[54]: True

Indeed the coordinates of p have $y \neq 0$:

In [55]:
```
p.coord(X)
```

Out[55]: $(1, 2, -1)$

Note in passing that since X is the default chart on $\mathcal{M}$, its name can be omitted in the arguments of coord():

In [56]:
```
p.coord()
```

Out[56]: $(1, 2, -1)$

The coordinates of p can also be obtained by letting the chart act on the point (from the very definition of a chart!):

In [57]:
```
X(p)
```

Out[57]: $(1, 2, -1)$

Let q be a point with $y = 0$ and $x \geq 0$:

In [58]:
```
q = M.point((1,0,2), name='q')
```

This time, the point does not belong to U:

In [59]:
```
q in U
```

Out[59]: False

Accordingly, we cannot ask for the coordinates of q in the chart $Y = (U, (r, \theta, \phi))$:

In [60]:
```
try:
    q.coord(Y)
except ValueError as exc:
    print("Error: " + str(exc))
```

```
Error: the point does not belong to the domain of Chart (U, (r, th, ph))
```

but we can for point p:

In [61]:
```
p.coord(Y)
```

Out[61]: $\left(\sqrt{3}\sqrt{2}, \pi - \arctan\left(\sqrt{5}\right), \arctan(2)\right)$

In [62]:
```
Y(p)
```

Out[62]: $\left(\sqrt{3}\sqrt{2}, \pi - \arctan\left(\sqrt{5}\right), \arctan(2)\right)$

Points can be compared:

In [63]:
```
q == p
```

Out[63]: False

In [64]:
```
p1 = U.point((sqrt(3)*sqrt(2), pi-atan(sqrt(5)), atan(2)), chart=Y)
p1 == p
```

Out[64]: True

In SageMath's terminology, points are **elements**, whose **parents** are the manifold on which they have been defined:

In [65]:
```
p.parent()
```

Out[65]: $\mathcal{M}$

In [66]:
```
q.parent()
```

Out[66]: $\mathcal{M}$

In [67]:
```
p1.parent()
```

Out[67]: U

Scalar fields

A **scalar field** is a differentiable map $U \to \mathbb{R}$, where U is an open subset of $\mathcal{M}$.

A scalar field is defined by its expressions in terms of charts covering its domain (in general more than one chart is necessary to cover all the domain):

In [68]:
```
f = U.scalar_field({X_U: x+y^2+z^3}, name='f')
print(f)
```

```
Scalar field f on the Open subset U of the 3-dimensional differentiable manifold M
```

The coordinate expressions of the scalar field are passed as a Python dictionary, with the charts as keys, hence the writing `{X_U: x+y^2+z^3}` . Since in the present case, there is only one chart in the dictionary, an alternative writing is

In [69]:
```
f = U.scalar_field(x+y^2+z^3, chart=X_U, name='f')
print(f)
```

```
Scalar field f on the Open subset U of the 3-dimensional differentiable manifold M
```

Since `X_U` is the domain's default chart, it can be omitted in the above declaration:

In [70]:
```
f = U.scalar_field(x+y^2+z^3, name='f')
print(f)
```

```
Scalar field f on the Open subset U of the 3-dimensional differentiable manifold M
```

As a map $U \subset \mathcal{M} \longrightarrow \mathbb{R}$, a scalar field acts on points, not on coordinates:

In [71]:
```
f(p)
```

Out[71]: 4

The method `display()` provides the expression of the scalar field in terms of a given chart:

In [72]:
```
f.display(X_U)
```

Out[72]:
$$\begin{array}{llcl} f: & U & \longrightarrow & \mathbb{R} \\ & (x, y, z) & \longmapsto & z^3 + y^2 + x \end{array}$$

If no argument is provided, the method `display()` shows the coordinate expression of the scalar field in all the charts defined on the domain (except for *subcharts*, i.e. the restrictions of some chart to a subdomain):

In [73]:
```
f.display()
```

Out[73]:
$$\begin{array}{llcl} f: & U & \longrightarrow & \mathbb{R} \\ & (x, y, z) & \longmapsto & z^3 + y^2 + x \\ & (r, \theta, \phi) & \longmapsto & r^3 \cos\left(\theta\right)^3 + r^2 \sin\left(\phi\right)^2 \sin\left(\theta\right)^2 + r \cos(\phi) \sin(\theta) \end{array}$$

Note that the expression of f in terms of the coordinates (r, θ, ϕ) has not been provided by the user but has been automatically computed by means of the change-of-coordinate formula declared

above in the transition map.

In [74]:
```
f.display(Y)
```

Out[74]:
$$\begin{array}{llcl} f: & U & \longrightarrow & \mathbb{R} \\ & (r, \theta, \phi) & \longmapsto & r^3 \cos\left(\theta\right)^3 + r^2 \sin\left(\phi\right)^2 \sin\left(\theta\right)^2 + r \cos(\phi) \sin(\theta) \end{array}$$

In each chart, the scalar field is represented by a function of the chart coordinates (an object of the type `ChartFunction` described above), which is accessible via the method `coord_function()`:

In [75]:
```
f.coord_function(X_U)
```

Out[75]: $z^3 + y^2 + x$

In [76]:
```
f.coord_function(X_U).display()
```

Out[76]: $(x, y, z) \mapsto z^3 + y^2 + x$

In [77]:
```
f.coord_function(Y)
```

Out[77]: $r^3 \cos\left(\theta\right)^3 + r^2 \sin\left(\phi\right)^2 \sin\left(\theta\right)^2 + r \cos(\phi) \sin(\theta)$

In [78]:
```
f.coord_function(Y).display()
```

Out[78]: $(r, \theta, \phi) \mapsto r^3 \cos\left(\theta\right)^3 + r^2 \sin\left(\phi\right)^2 \sin\left(\theta\right)^2 + r \cos(\phi) \sin(\theta)$

The "raw" symbolic expression is returned by the method `expr()`:

In [79]:
```
f.expr(X_U)
```

Out[79]: $z^3 + y^2 + x$

In [80]:
```
f.expr(Y)
```

Out[80]: $r^3 \cos\left(\theta\right)^3 + r^2 \sin\left(\phi\right)^2 \sin\left(\theta\right)^2 + r \cos(\phi) \sin(\theta)$

In [81]:
```
f.expr(Y) is f.coord_function(Y).expr()
```

Out[81]: True

A scalar field can also be defined by some unspecified function of the coordinates:

In [82]:
```
h = U.scalar_field(function('H')(x, y, z), name='h')
print(h)
```

```
Scalar field h on the Open subset U of the 3-dimensional differentiable manifold M
```

In [83]:
```
h.display()
```

Out[83]:
$$\begin{array}{llcl} h: & U & \longrightarrow & \mathbb{R} \\ & (x, y, z) & \longmapsto & H\left(x, y, z\right) \\ & (r, \theta, \phi) & \longmapsto & H\left(r \cos(\phi) \sin(\theta), r \sin(\phi) \sin(\theta), r \cos(\theta)\right) \end{array}$$

In [84]:
```
h.display(Y)
```

Out[84]:
$$\begin{array}{llcl} h: & U & \longrightarrow & \mathbb{R} \\ & (r, \theta, \phi) & \longmapsto & H\left(r \cos(\phi) \sin(\theta), r \sin(\phi) \sin(\theta), r \cos(\theta)\right) \end{array}$$

In [85]:
```
h(p) # remember that p is the point of coordinates (1,2,-1) in the chart X_U
```

Out[85]: $H\left(1, 2, -1\right)$

The parent of f is the set $C^\infty(U)$ of all smooth scalar fields on U, which is a commutative algebra over $\mathbb{R}$:

In [86]:
```
CU = f.parent()
CU
```

Out[86]: $C^\infty(U)$

In [87]:
```
print(CU)
```

```
Algebra of differentiable scalar fields on the Open subset U of the 3-dimensional di
fferentiable manifold M
```

In [88]:
```
CU.category()
```

Out[88]: **JoinCategory**

The base ring of the algebra is the field $\mathbb{R}$, which is represented here by SageMath's Symbolic Ring (SR):

In [89]:
```
CU.base_ring()
```

Out[89]: SR

Arithmetic operations on scalar fields are defined through the algebra structure:

In [90]:
```
s = f + 2*h
print(s)
```

```
Scalar field on the Open subset U of the 3-dimensional differentiable manifold M
```

In [91]:
```
s.display()
```

Out[91]:

$$\begin{array}{lcl} U & \longrightarrow & \mathbb{R} \\ (x, y, z) & \longmapsto & z^3 + y^2 + x + 2\,H\left(x, y, z\right) \\ (r, \theta, \phi) & \longmapsto & r^3 \cos\left(\theta\right)^3 + r^2 \sin\left(\phi\right)^2 \sin\left(\theta\right)^2 + r\cos(\phi)\sin(\theta) + 2\,H\left(r\cos(\phi)\sin(\theta), r\sin(\right. \end{array}$$

Tangent spaces

The tangent vector space to the manifold at point p is obtained as follows:

In [92]:
```
Tp = M.tangent_space(p)
Tp
```

Out[92]: $T_p\,\mathcal{M}$

In [93]:
```
print(Tp)
```

```
Tangent space at Point p on the 3-dimensional differentiable manifold M
```

$T_p\,\mathcal{M}$ is a 2-dimensional vector space over $\mathbb{R}$ (represented here by SageMath's Symbolic Ring (SR)):

In [94]:
```
print(Tp.category())
```

```
Category of finite dimensional vector spaces over Symbolic Ring
```

In [95]:
```
Tp.dim()
```

Out[95]: 3

$T_p\,\mathcal{M}$ is automatically endowed with vector bases deduced from the vector frames defined around the point:

In [96]:
```
Tp.bases()
```

Out[96]:
$$\left[\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right), \left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)\right]$$

For the tangent space at the point q, on the contrary, there is only one pre-defined basis, since q is not in the domain U of the frame associated with coordinates (r, θ, ϕ):

In [97]:
```
Tq = M.tangent_space(q)
Tq.bases()
```

Out[97]:
$$\left[\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right]$$

A random element:

In [98]:
```
v = Tp.an_element()
print(v)
```

```
Tangent vector at Point p on the 3-dimensional differentiable manifold M
```

In [99]:
```
v.display()
```

Out[99]:
$$\frac{\partial}{\partial x} + 2\frac{\partial}{\partial y} + 3\frac{\partial}{\partial z}$$

In [100…
```
u = Tq.an_element()
print(u)
```

```
Tangent vector at Point q on the 3-dimensional differentiable manifold M
```

In [101…
```
u.display()
```

Out[101…
$$\frac{\partial}{\partial x} + 2\frac{\partial}{\partial y} + 3\frac{\partial}{\partial z}$$

Note that, despite what the above simplified writing may suggest (the mention of the point p or q is omitted in the basis vectors), u and v are different vectors, for they belong to different vector spaces:

In [102…
```
v.parent()
```

Out[102…
$$T_p\,\mathcal{M}$$

In [103…
```
u.parent()
```

Out[103…
$$T_q\,\mathcal{M}$$

In particular, it is not possible to add u and v:

In [104…
```
try:
    s = u + v
except TypeError as exc:
    print("Error: " + str(exc))
```

```
Error: unsupported operand parent(s) for +: 'Tangent space at Point q on the 3-dimen
sional differentiable manifold M' and 'Tangent space at Point p on the 3-dimensional
differentiable manifold M'
```

Vector Fields

Each chart defines a vector frame on the chart domain: the so-called **coordinate basis**:

In [105…]
```
X.frame()
```

Out[105…]
$$\left(\mathcal{M}, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right)$$

In [106…]
```
X.frame().domain()  # this frame is defined on the whole manifold
```

Out[106…]
$$\mathcal{M}$$

In [107…]
```
Y.frame()
```

Out[107…]
$$\left(U, \left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)\right)$$

In [108…]
```
Y.frame().domain() # this frame is defined only on U
```

Out[108…]
$$U$$

The list of frames defined on a given open subset is returned by the method `frames()` :

In [109…]
```
M.frames()
```

Out[109…]
$$\left[\left(\mathcal{M}, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right), \left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right), \left(U, \left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)\right)\right]$$

In [110…]
```
U.frames()
```

Out[110…]
$$\left[\left(U, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right), \left(U, \left(\frac{\partial}{\partial r}, \frac{\partial}{\partial \theta}, \frac{\partial}{\partial \phi}\right)\right)\right]$$

In [111…]
```
M.default_frame()
```

Out[111…]
$$\left(\mathcal{M}, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right)$$

Unless otherwise specified (via the command `set_default_frame()`), the default frame is that associated with the default chart:

In [112…]
```
M.default_frame() is M.default_chart().frame()
```

Out[112…]
$$\text{True}$$

In [113…]
```
U.default_frame() is U.default_chart().frame()
```

Out[113…]
$$\text{True}$$

Individual elements of a frame can be accessed by means of their indices:

In [114…]
```
e = U.default_frame()
e2 = e[2]
e2
```

Out[114…]
$$\frac{\partial}{\partial y}$$

In [115…]
```
print(e2)
```

```
Vector field ∂/∂y on the Open subset U of the 3-dimensional differentiable manifold
M
```

We may define a new vector field as follows:

In [116…
```
v = e[2] + 2*x*e[3]
print(v)
```

```
Vector field on the Open subset U of the 3-dimensional differentiable manifold M
```

In [117…
```
v.display()
```

Out[117…
$$\frac{\partial}{\partial y} + 2\,x\frac{\partial}{\partial z}$$

A vector field can be defined by its components with respect to a given vector frame. When the latter is not specified, the open set's default frame is of course assumed:

In [118…
```
v = U.vector_field(name='v') # vector field defined on the open set U
v[1] = 1+y
v[2] = -x
v[3] = x*y*z
v.display()
```

Out[118…
$$v = (y+1)\,\frac{\partial}{\partial x} - x\frac{\partial}{\partial y} + xyz\frac{\partial}{\partial z}$$

It is possible to initialize the components of the vector field while declaring it, so that the above is equivalent to

In [119…
```
v = U.vector_field(1+y, -x, x*y*z, name='v')
v.display()
```

Out[119…
$$v = (y+1)\,\frac{\partial}{\partial x} - x\frac{\partial}{\partial y} + xyz\frac{\partial}{\partial z}$$

Vector fields on U are Sage *element* objects, whose *parent* is the set $\mathfrak{X}(U)$ of vector fields defined on U:

In [120…
```
v.parent()
```

Out[120…
$$\mathfrak{X}\left(U\right)$$

The set $\mathfrak{X}(U)$ is a module over the commutative algebra $C^\infty(U)$ of scalar fields on U:

In [121…
```
print(v.parent())
```

```
Free module X(U) of vector fields on the Open subset U of the 3-dimensional differen
tiable manifold M
```

In [122…
```
print(v.parent().category())
```

```
Category of finite dimensional modules over Algebra of differentiable scalar fields
on the Open subset U of the 3-dimensional differentiable manifold M
```

In [123…
```
v.parent().base_ring()
```

Out[123…
$$C^\infty\left(U\right)$$

A vector field acts on scalar fields:

In [124…
```
f.display()
```

Out[124…
$$\begin{array}{llcl} f: & U & \longrightarrow & \mathbb{R} \\ & (x, y, z) & \longmapsto & z^3 + y^2 + x \\ & (r, \theta, \phi) & \longmapsto & r^3\cos\left(\theta\right)^3 + r^2\sin\left(\phi\right)^2\sin\left(\theta\right)^2 + r\cos(\phi)\sin(\theta) \end{array}$$

In [125…
```
s = v(f)
print(s)
```

```
Scalar field v(f) on the Open subset U of the 3-dimensional differentiable manifold
M
```

In [126…]
```
s.display()
```

Out[126…]
$$\begin{aligned} v\,(f):\ U &\longrightarrow \mathbb{R} \\ (x, y, z) &\longmapsto 3\,xyz^3 - (2\,x - 1)y + 1 \\ (r, \theta, \phi) &\longmapsto r\sin(\phi)\sin(\theta) + \left(3\,r^5\cos(\phi)\cos\left(\theta\right)^3\sin(\phi) - 2\,r^2\cos(\phi)\sin(\phi)\right)\sin\left(\theta\right) \end{aligned}$$

In [127…]
```
e[3].display()
```

Out[127…]
$$\frac{\partial}{\partial z} = \frac{\partial}{\partial z}$$

In [128…]
```
e[3](f).display()
```

Out[128…]
$$\begin{aligned} \frac{\partial}{\partial z}(f):\ U &\longrightarrow \mathbb{R} \\ (x, y, z) &\longmapsto 3\,z^2 \\ (r, \theta, \phi) &\longmapsto 3\,r^2\cos\left(\theta\right)^2 \end{aligned}$$

Unset components are assumed to be zero:

In [129…]
```
w = U.vector_field(name='w')
w[2] = 3
w.display()
```

Out[129…]
$$w = 3\frac{\partial}{\partial y}$$

A vector field on U can be expanded in the vector frame associated with the chart (r, θ, ϕ):

In [130…]
```
v.display(Y.frame())
```

Out[130…]
$$v = \left(\frac{xyz^2 + x}{\sqrt{x^2 + y^2 + z^2}}\right)\frac{\partial}{\partial r} + \left(-\frac{\left(x^3y + xy^3 - x\right)\sqrt{x^2 + y^2}z}{x^4 + 2\,x^2y^2 + y^4 + (x^2 + y^2)z^2}\right)\frac{\partial}{\partial \theta} + \left(-\frac{x^2 + y^2 + y}{x^2 + y^2}\right)\frac{\partial}{\partial}$$

By default, the components are expressed in terms of the default coordinates (x, y, z). To express them in terms of the coordinates (r, θ, ϕ), one should add the corresponding chart as the second argument of the method `display()` :

In [131…]
```
v.display(Y.frame(), Y)
```

Out[131…]
$$v = \left(r^3\cos(\phi)\cos\left(\theta\right)^2\sin(\phi)\sin\left(\theta\right)^2 + \cos(\phi)\sin(\theta)\right)\frac{\partial}{\partial r} + \left(-\frac{r^3\cos(\phi)\cos(\theta)\sin(\phi)\sin\left(\theta\right)^3}{r}\right.$$

As a shortcut, for a coordinate frame, one may provide the name of the chart only:

In [132…]
```
v.display(Y)
```

Out[132…]
$$v = \left(r^3\cos(\phi)\cos\left(\theta\right)^2\sin(\phi)\sin\left(\theta\right)^2 + \cos(\phi)\sin(\theta)\right)\frac{\partial}{\partial r} + \left(-\frac{r^3\cos(\phi)\cos(\theta)\sin(\phi)\sin\left(\theta\right)^3}{r}\right.$$

In [133…]
```
for i in M.irange():
    show(e[i].display(Y))
```

$$\frac{\partial}{\partial x} = \cos(\phi)\sin(\theta)\frac{\partial}{\partial r} + \frac{\cos(\phi)\cos(\theta)}{r}\frac{\partial}{\partial \theta} - \frac{\sin(\phi)}{r\sin(\theta)}\frac{\partial}{\partial \phi}$$

$$\frac{\partial}{\partial y} = \sin(\phi)\sin(\theta)\frac{\partial}{\partial r} + \frac{\cos(\theta)\sin(\phi)}{r}\frac{\partial}{\partial \theta} + \frac{\cos(\phi)}{r\sin(\theta)}\frac{\partial}{\partial \phi}$$

$$\frac{\partial}{\partial z} = \cos(\theta)\frac{\partial}{\partial r} - \frac{\sin(\theta)}{r}\frac{\partial}{\partial \theta}$$

The components of a tensor field w.r.t. the default frame can also be obtained as a list, thanks to the operator `[:]` :

In [134…]
```
v[:]
```

Out[134…] $[y+1, -x, xyz]$

An alternative is to use the method `display_comp()` :

In [135…]
```
v.display_comp()
```

Out[135…]
$$\begin{aligned} v^x &= y+1 \\ v^y &= -x \\ v^z &= xyz \end{aligned}$$

To obtain the components w.r.t. another frame, one may go through the method `comp()` and specify the frame:

In [136…]
```
v.comp(Y.frame())[:]
```

Out[136…] $$\left[\frac{xyz^2+x}{\sqrt{x^2+y^2+z^2}}, -\frac{\left(x^3y+xy^3-x\right)\sqrt{x^2+y^2}z}{x^4+2\,x^2y^2+y^4+(x^2+y^2)z^2}, -\frac{x^2+y^2+y}{x^2+y^2}\right]$$

However a shortcut is to provide the frame as the first argument of the square brackets:

In [137…]
```
v[Y.frame(), :]
```

Out[137…] $$\left[\frac{xyz^2+x}{\sqrt{x^2+y^2+z^2}}, -\frac{\left(x^3y+xy^3-x\right)\sqrt{x^2+y^2}z}{x^4+2\,x^2y^2+y^4+(x^2+y^2)z^2}, -\frac{x^2+y^2+y}{x^2+y^2}\right]$$

In [138…]
```
v.display_comp(Y.frame())
```

Out[138…]
$$\begin{aligned} v^r &= \frac{xyz^2+x}{\sqrt{x^2+y^2+z^2}} \\ v^\theta &= -\frac{\left(x^3y+xy^3-x\right)\sqrt{x^2+y^2}z}{x^4+2\,x^2y^2+y^4+(x^2+y^2)z^2} \\ v^\phi &= -\frac{x^2+y^2+y}{x^2+y^2} \end{aligned}$$

Components are shown expressed in terms of the default's coordinates; to get them in terms of the coordinates (r, θ, ϕ) instead, add the chart name as the last argument in the square brackets:

In [139…]
```
v[Y.frame(), :, Y]
```

Out[139…] $$\left[r^3\cos(\phi)\cos\left(\theta\right)^2\sin(\phi)\sin\left(\theta\right)^2 + \cos(\phi)\sin(\theta), -\frac{r^3\cos(\phi)\cos(\theta)\sin(\phi)\sin\left(\theta\right)^3 - \cos(\phi)\cos(\right.$$

or specify the chart in `display_comp()` :

In [140…]
```
v.display_comp(Y.frame(), chart=Y)
```

Out[140…]
$$\begin{aligned} v^r &= r^3\cos(\phi)\cos\left(\theta\right)^2\sin(\phi)\sin\left(\theta\right)^2 + \cos(\phi)\sin(\theta) \\ v^\theta &= -\frac{r^3\cos(\phi)\cos(\theta)\sin(\phi)\sin\left(\theta\right)^3 - \cos(\phi)\cos(\theta)}{r} \\ v^\phi &= -\frac{r\sin(\theta)+\sin(\phi)}{r\sin(\theta)} \end{aligned}$$

To get a vector component as a scalar field instead of a coordinate expression, use double square brackets:

In [141…
```
print(v[[1]])
```

```
Scalar field on the Open subset U of the 3-dimensional differentiable manifold M
```

In [142…
```
v[[1]].display()
```

Out[142…
$$\begin{array}{lcl} U & \longrightarrow & \mathbb{R} \\ (x, y, z) & \longmapsto & y + 1 \\ (r, \theta, \phi) & \longmapsto & r \sin(\phi) \sin(\theta) + 1 \end{array}$$

In [143…
```
v[[1]].expr(X_U)
```

Out[143…
$$y + 1$$

A vector field can be defined with components being unspecified functions of the coordinates:

In [144…
```
u = U.vector_field(function('u_x')(x,y,z),
                   function('u_y')(x,y,z),
                   function('u_z')(x,y,z), name='u')
u.display()
```

Out[144…
$$u = u_x\,(x, y, z)\,\frac{\partial}{\partial x} + u_y\,(x, y, z)\,\frac{\partial}{\partial y} + u_z\,(x, y, z)\,\frac{\partial}{\partial z}$$

In [145…
```
s = v + u
s.set_name('s')
s.display()
```

Out[145…
$$s = (y + u_x\,(x, y, z) + 1)\,\frac{\partial}{\partial x} + \left(-x + u_y\,(x, y, z)\right)\,\frac{\partial}{\partial y} + (xyz + u_z\,(x, y, z))\,\frac{\partial}{\partial z}$$

Values of vector field at a given point

The value of a vector field at some point of the manifold is obtained via the method `at()`:

In [146…
```
vp = v.at(p)
print(vp)
```

```
Tangent vector v at Point p on the 3-dimensional differentiable manifold M
```

In [147…
```
vp.display()
```

Out[147…
$$v = 3\frac{\partial}{\partial x} - \frac{\partial}{\partial y} - 2\frac{\partial}{\partial z}$$

Indeed, recall that, w.r.t. chart `X_U` $=(x, y, z)$, the coordinates of the point p and the components of the vector field v are

In [148…
```
p.coord(X_U)
```

Out[148…
$$(1, 2, -1)$$

In [149…
```
v.display(X_U.frame(), X_U)
```

Out[149…
$$v = (y + 1)\,\frac{\partial}{\partial x} - x\frac{\partial}{\partial y} + xyz\frac{\partial}{\partial z}$$

Note that to simplify the writing, the symbol used to denote the value of the vector field at point p is the same as that of the vector field itself (namely v); this can be changed by the method `set_name()`:

In [150…
```
vp.set_name(latex_name='v|_p')
vp.display()
```

Out[150…
$$v|_p = 3\frac{\partial}{\partial x} - \frac{\partial}{\partial y} - 2\frac{\partial}{\partial z}$$

Of course, $v|_p$ belongs to the tangent space at p:

In [151…
```
vp.parent()
```

Out[151…
$$T_p\,\mathcal{M}$$

In [152…
```
vp in M.tangent_space(p)
```

Out[152…
$$\mathrm{True}$$

In [153…
```
up = u.at(p)
print(up)
```

```
Tangent vector u at Point p on the 3-dimensional differentiable manifold M
```

In [154…
```
up.display()
```

Out[154…
$$u = u_x\,(1,2,-1)\,\frac{\partial}{\partial x} + u_y\,(1,2,-1)\,\frac{\partial}{\partial y} + u_z\,(1,2,-1)\,\frac{\partial}{\partial z}$$

1-forms

A **1-form** on $\mathcal{M}$ is a field of linear forms. For instance, it can be the *differential of a scalar field*:

In [155…
```
df = f.differential()
print(df)
```

```
1-form df on the Open subset U of the 3-dimensional differentiable manifold M
```

An equivalent writing is

In [156…
```
df = diff(f)
```

The method `display()` shows the expansion on the default coframe:

In [157…
```
df.display()
```

Out[157…
$$\mathrm{d}f = \mathrm{d}x + 2\,y\mathrm{d}y + 3\,z^2\mathrm{d}z$$

In the above writing, the 1-form is expanded over the basis $(\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)$ associated with the chart (x, y, z). This basis can be accessed via the method `coframe()`:

In [158…
```
dX = X.coframe()
dX
```

Out[158…
$$(\mathcal{M}, (\mathrm{d}x, \mathrm{d}y, \mathrm{d}z))$$

The list of all coframes defined on a given manifold open subset is returned by the method `coframes()`:

In [159…
```
M.coframes()
```

Out[159… $[(\mathcal{M}, (\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)), (U, (\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)), (U, (\mathrm{d}r, \mathrm{d}\theta, \mathrm{d}\phi))]$

As for a vector field, the value of the differential form at some point on the manifold is obtained by the method `at()` :

In [160…
```
dfp = df.at(p)
print(dfp)
```

```
Linear form df on the Tangent space at Point p on the 3-dimensional differentiable m
anifold M
```

In [161…
```
dfp.display()
```

Out[161… $\mathrm{d}f = \mathrm{d}x + 4\mathrm{d}y + 3\mathrm{d}z$

Recall that

In [162…
```
p.coord()
```

Out[162… $(1, 2, -1)$

The linear form $\mathrm{d}f|_p$ belongs to the dual of the tangent vector space at p:

In [163…
```
dfp.parent()
```

Out[163… $T_p\,\mathcal{M}^*$

In [164…
```
dfp.parent() is M.tangent_space(p).dual()
```

Out[164… True

As such, it is acting on vectors at p, yielding a real number:

In [165…
```
print(vp)
vp.display()
```

```
Tangent vector v at Point p on the 3-dimensional differentiable manifold M
```

Out[165… $v|_p = 3\frac{\partial}{\partial x} - \frac{\partial}{\partial y} - 2\frac{\partial}{\partial z}$

In [166…
```
dfp(vp)
```

Out[166… -7

In [167…
```
print(up)
up.display()
```

```
Tangent vector u at Point p on the 3-dimensional differentiable manifold M
```

Out[167… $u = u_x\,(1,2,-1)\,\frac{\partial}{\partial x} + u_y\,(1,2,-1)\,\frac{\partial}{\partial y} + u_z\,(1,2,-1)\,\frac{\partial}{\partial z}$

In [168…
```
dfp(up)
```

Out[168… $u_x\,(1,2,-1) + 4\,u_y\,(1,2,-1) + 3\,u_z\,(1,2,-1)$

The differential 1-form of the unspecified scalar field h:

In [169…
```
dh = h.differential()
dh.display()
```

Out[169…] $$\mathrm{d}h = \frac{\partial H}{\partial x}\mathrm{d}x + \frac{\partial H}{\partial y}\mathrm{d}y + \frac{\partial H}{\partial z}\mathrm{d}z$$

A 1-form can also be defined from scratch:

In [170…]
```
om = U.one_form(name='omega', latex_name=r'\omega')
print(om)
```

```
1-form omega on the Open subset U of the 3-dimensional differentiable manifold M
```

It can be specified by providing its components in a given coframe:

In [171…]
```
om[:] = [x^2+y^2, z, x-z]    # components in the default coframe (dx,dy,dz)
om.display()
```

Out[171…] $$\omega = \left(x^2 + y^2\right) \mathrm{d}x + z\mathrm{d}y + (x - z)\,\mathrm{d}z$$

It is also possible to initialize the components of the 1-form while declaring it, so that the above is equivalent to

In [172…]
```
om = U.one_form(x^2+y^2, z, x-z, name='omega',
                latex_name=r'\omega')
om.display()
```

Out[172…] $$\omega = \left(x^2 + y^2\right) \mathrm{d}x + z\mathrm{d}y + (x - z)\,\mathrm{d}z$$

Of course, one may set the components in a frame different from the default one:

In [173…]
```
om[Y.frame(), :, Y] = [r*sin(th)*cos(ph), 0, r*sin(th)*sin(ph)]
om.display(Y.frame(), Y)
```

Out[173…] $$\omega = r\cos(\phi)\sin(\theta)\mathrm{d}r + r\sin(\phi)\sin(\theta)\mathrm{d}\phi$$

The components in the coframe $(\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)$ are updated automatically:

In [174…]
```
om.display()
```

Out[174…] $$\omega = \left(\frac{x^4 + x^2y^2 - \sqrt{x^2 + y^2 + z^2}y^2}{\sqrt{x^2 + y^2 + z^2}(x^2 + y^2)}\right) \mathrm{d}x + \left(\frac{x^3y + xy^3 + \sqrt{x^2 + y^2 + z^2}xy}{\sqrt{x^2 + y^2 + z^2}(x^2 + y^2)}\right) \mathrm{d}y + \left(\frac{z}{\sqrt{x^2 +}\right.$$

Let us revert to the values set previously:

In [175…]
```
om[:] = [x^2+y^2, z, x-z]
om.display()
```

Out[175…] $$\omega = \left(x^2 + y^2\right) \mathrm{d}x + z\mathrm{d}y + (x - z)\,\mathrm{d}z$$

This time, the components in the coframe $(\mathrm{d}r, \mathrm{d}\theta, \mathrm{d}\phi)$ are those that are updated:

In [176…]
```
om.display(Y.frame(), Y)
```

Out[176…] $$\omega = \left(r^2\cos(\phi)\sin\left(\theta\right)^3 + r(\cos(\phi) + \sin(\phi))\cos(\theta)\sin(\theta) - r\cos\left(\theta\right)^2\right) \mathrm{d}r$$
$$+ \left(r^2\cos\left(\theta\right)^2\sin(\phi) + r^2\cos(\theta)\sin(\theta) + \left(r^3\cos(\phi)\cos(\theta) - r^2\cos(\phi)\right)\sin\left(\theta\right)^2\right) \mathrm{d}\theta + \left(-r^3\sin(\right.$$

A 1-form acts on vector fields, resulting in a scalar field:

In [177…]
```
print(om(v))
om(v).display()
```

```
Scalar field omega(v) on the Open subset U of the 3-dimensional differentiable manif
old M
```

Out[177…
$$\begin{aligned}\omega(v): U &\longrightarrow \mathbb{R}\\ (x,y,z) &\longmapsto -xyz^2+x^2y+y^3+x^2+y^2+\left(x^2y-x\right)z\\ (r,\theta,\phi) &\longmapsto -r^2\cos(\phi)\cos(\theta)\sin(\theta)+\left(r^4\cos(\phi)^2\cos(\theta)\sin(\phi)+r^3\sin(\phi)\right)\sin(\theta\end{aligned}$$

In [178…
```
print(df(v))
df(v).display()
```

```
Scalar field df(v) on the Open subset U of the 3-dimensional differentiable manifold
M
```

Out[178…
$$\begin{aligned}\mathrm{d}f(v): U &\longrightarrow \mathbb{R}\\ (x,y,z) &\longmapsto 3\,xyz^3-(2\,x-1)y+1\\ (r,\theta,\phi) &\longmapsto r\sin(\phi)\sin(\theta)+\left(3\,r^5\cos(\phi)\cos(\theta)^3\sin(\phi)-2\,r^2\cos(\phi)\sin(\phi)\right)\sin(\end{aligned}$$

In [179…
```
om(u).display()
```

Out[179…
$$\begin{aligned}\omega(u): U &\longrightarrow \mathbb{R}\\ (x,y,z) &\longmapsto x^2u_x(x,y,z)+y^2u_x(x,y,z)+z\big(u_y(x,y,z)-u_z(x,y,z)\big)+xu_z(x,y\\ (r,\theta,\phi) &\longmapsto r^2\sin(\theta)^2u_x(r\cos(\phi)\sin(\theta),r\sin(\phi)\sin(\theta),r\cos(\theta))+r\cos(\theta)u_y(r\,\mathrm{co}\\ &\quad+(r\cos(\phi)\sin(\theta)-r\cos(\theta))u_z(r\cos(\phi)\sin(\theta),r\sin(\end{aligned}$$

In the case of a differential 1-form, the following identity holds:

In [180…
```
df(v) == v(f)
```

Out[180… True

1-forms are Sage *element* objects, whose *parent* is the $C^\infty(U)$-module $\Omega^1(U)$ of all 1-forms defined on U:

In [181…
```
df.parent()
```

Out[181… $\Omega^1(U)$

In [182…
```
print(df.parent())
```

```
Free module Omega^1(U) of 1-forms on the Open subset U of the 3-dimensional differen
tiable manifold M
```

In [183…
```
print(om.parent())
```

```
Free module Omega^1(U) of 1-forms on the Open subset U of the 3-dimensional differen
tiable manifold M
```

$\Omega^1(U)$ is actually the dual of the free module $\mathfrak{X}(U)$:

In [184…
```
df.parent() is v.parent().dual()
```

Out[184… True

Differential forms and exterior calculus

The **exterior product** of two 1-forms is taken via the method `wedge()` and results in a 2-form:

In [185…
```
a = om.wedge(df)
print(a)
a.display()
```

```
2-form omega∧df on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[185…
$$\omega\wedge\mathrm{d}f=\left(2\,x^2y+2\,y^3-z\right)\mathrm{d}x\wedge\mathrm{d}y+\left(3\left(x^2+y^2\right)z^2-x+z\right)\mathrm{d}x\wedge\mathrm{d}z+\left(3\,z^3-2\,xy+2\,yz\right)\mathrm{d}$$

A matrix view of the components:

In [186…]
```
a[:]
```

Out[186…]
$$\begin{pmatrix} 0 & 2\,x^2y + 2\,y^3 - z & 3\left(x^2 + y^2\right)z^2 - x + z \\ -2\,x^2y - 2\,y^3 + z & 0 & 3\,z^3 - 2\,xy + 2\,yz \\ -3\left(x^2 + y^2\right)z^2 + x - z & -3\,z^3 + 2\,xy - 2\,yz & 0 \end{pmatrix}$$

Displaying only the non-vanishing components, skipping the redundant ones (i.e. those that can be deduced by antisymmetry):

In [187…]
```
a.display_comp(only_nonredundant=True)
```

Out[187…]
$$\begin{aligned} \omega \wedge \mathrm{d}f_{\,x\,y} &= 2\,x^2y + 2\,y^3 - z \\ \omega \wedge \mathrm{d}f_{\,x\,z} &= 3\left(x^2 + y^2\right)z^2 - x + z \\ \omega \wedge \mathrm{d}f_{\,y\,z} &= 3\,z^3 - 2\,xy + 2\,yz \end{aligned}$$

The 2-form $\omega \wedge \mathrm{d}f$ can be expanded on the $(\mathrm{d}r, \mathrm{d}\theta, \mathrm{d}\phi)$ coframe:

In [188…]
```
a.display(Y.frame(), Y)
```

Out[188…]
$$\begin{aligned} \omega \wedge \mathrm{d}f = {} & \left(3\,r^5\cos(\phi)\sin(\theta)^4 - \left(3\,r^5\cos(\phi) - 3\,r^4\cos(\theta)\sin(\phi) - 2\,r^3\cos(\phi)\sin(\phi)^2\right)\sin(\theta)^2\right. \\ & \left. - \left(2\,r^3\cos(\theta)\sin(\phi)^2 - r^2\cos(\phi)^2\right)\sin(\theta)\right)\mathrm{d}r \wedge \mathrm{d}\theta \\ & + \left(2\,r^4\sin(\phi)\sin(\theta)^5 + \left(3\,r^5\cos(\theta)^3\sin(\phi) + 2\,r^3\cos(\phi)^2\cos(\theta)\sin(\phi)\right)\sin(\theta)^3 - \left(2\,r^3\cos(\phi)\right.\right. \\ & \left.\left.(\theta)^2 - \left(3\,r^4\cos(\phi)\cos(\theta)^4 - r^2\cos(\theta)^2\sin(\phi)\right)\sin(\theta)\right)\mathrm{d}r \wedge \mathrm{d}\phi \right. \\ & + \left(-r^3\cos(\theta)^2\sin(\theta) - \left(3\,r^6\cos(\theta)^2\sin(\phi) + 2\,r^4\cos(\phi)^2\sin(\phi) - 2\,r^5\cos(\theta)\sin(\phi)\right)\sin(\theta)^4\right. \\ & \left. + \left(3\,r^5\cos(\phi)\cos(\theta)^3 - r^3\cos(\theta)\sin(\phi)\right)\sin(\theta)^2\right)\mathrm{d}\theta \wedge \mathrm{d}\phi \end{aligned}$$

As a 2-form, $A := \omega \wedge \mathrm{d}f$ can be applied to a pair of vectors and is antisymmetric:

In [189…]
```
a.set_name('A')
print(a(u,v))
a(u,v).display()
```

```
Scalar field A(u,v) on the Open subset U of the 3-dimensional differentiable manifol
d M
```

Out[189…
$$A\,(u,v):\quad U \longrightarrow \mathbb{R}$$
$$(x,y,z) \longmapsto 3\,xyz^4u_y\,(x,y,z) - 2\,x^2y^2u_y\,(x,y,z) - \\ - \left(3\,y^3u_z\,(x,y,z) - (2$$

$$- \left(2\,x^2y^2u_y\,(x,y,z) + \left(x^2u_x\,(x,y,\right.\right.$$
$$(r,\theta,\phi) \longmapsto \left(r^4\cos(\phi)\cos(\theta)^2\sin(\phi)\sin(\theta)^2 + \left(\sin(\phi)^3 - \sin(\phi)\right)r^4\cos(\theta)\sin\right.$$
$$+ \left(3\,r^6\cos(\phi)\cos(\theta)^4\sin(\phi)\sin(\theta)^2 + r^2\cos(\theta)\sin(\phi)\sin(\theta) + 2\left(\right.\right.$$
$$\left.\left.\left(r^5\cos(\phi)\cos(\theta)^2\sin(\phi)^2 - r^3\sin(\phi)\right)\sin(\theta)^3 + r\cos(\theta)\right)\right)$$

$$- \left(\left(\left(3\,r^5\cos(\theta)^2\sin(\phi) - 2\left(\sin(\phi)^3 - \sin(\phi)\right)r^3\right)\sin(\theta)^3 + \left(3\,r^4\right.\right.\right.$$
$$\left.\left.\left. - \left(3\,r^4\cos(\phi)\cos(\theta)^3 - r^2\cos(\theta)\sin(\phi) + r\cos(\phi)\right)\sin(\theta)\right)\right)$$

In [190…
```
a(u,v) == - a(v,u)
```

Out[190… True

In [191…
```
a.symmetries()
```

```
no symmetry; antisymmetry: (0, 1)
```

The **exterior derivative** of a differential form:

In [192…
```
dom = om.exterior_derivative()
print(dom)
dom.display()
```

```
2-form domega on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[192… $\mathrm{d}\omega = -2\,y\mathrm{d}x\wedge\mathrm{d}y + \mathrm{d}x\wedge\mathrm{d}z - \mathrm{d}y\wedge\mathrm{d}z$

Instead of invoking the method `exterior_derivative()`, one can use the function `diff()` (available in SageMath 9.2 or higher):

In [193…
```
dom = diff(om)
```

In [194…
```
da = diff(a)
print(da)
da.display()
```

```
3-form dA on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[194… $\mathrm{d}A = \left(-6\,yz^2 - 2\,y - 1\right)\mathrm{d}x\wedge\mathrm{d}y\wedge\mathrm{d}z$

The exterior derivative is nilpotent:

In [195…
```
ddf = diff(df)
ddf.display()
```

Out[195… $\mathrm{d}\mathrm{d}f = 0$

In [196…
```
ddom = diff(dom)
ddom.display()
```

Out[196… $\mathrm{d}\mathrm{d}\omega = 0$

Lie derivative

The Lie derivative of any tensor field with respect to a vector field is computed by the method `lie_derivative()`, with the vector field as the argument:

In [197…
```
lv_om = om.lie_derivative(v)
print(lv_om)
lv_om.display()
```

```
1-form on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[197…
$$\left(-yz^2 + (xy - 1)z + 2\,x\right)\,\mathrm{d}x + \left(-xz^2 + x^2 + y^2 + \left(x^2 + xy\right)z\right)\,\mathrm{d}y + \left(-2\,xyz + \left(x^2 + 1\right)y + 1\right)$$

In [198…
```
lu_dh = dh.lie_derivative(u)
print(lu_dh)
lu_dh.display()
```

```
1-form on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[198…
$$\left(u_x\,(x,y,z)\,\frac{\partial^2\,H}{\partial x^2} + u_y\,(x,y,z)\,\frac{\partial^2\,H}{\partial x\partial y} + u_z\,(x,y,z)\,\frac{\partial^2\,H}{\partial x\partial z} + \frac{\partial\,H}{\partial x}\,\frac{\partial\,u_x}{\partial x} + \frac{\partial\,H}{\partial y}\,\frac{\partial\,u_y}{\partial x} + \frac{\partial\,H}{\partial z}\,\frac{\partial\,u}{\partial x}\right.$$
$$+ \left(u_x\,(x,y,z)\,\frac{\partial^2\,H}{\partial x\partial y} + u_y\,(x,y,z)\,\frac{\partial^2\,H}{\partial y^2} + u_z\,(x,y,z)\,\frac{\partial^2\,H}{\partial y\partial z} + \frac{\partial\,H}{\partial x}\,\frac{\partial\,u_x}{\partial y} + \frac{\partial\,H}{\partial y}\,\frac{\partial\,u_y}{\partial y} + \frac{\partial\,H}{\partial z}\right.$$
$$+ \left(u_x\,(x,y,z)\,\frac{\partial^2\,H}{\partial x\partial z} + u_y\,(x,y,z)\,\frac{\partial^2\,H}{\partial y\partial z} + u_z\,(x,y,z)\,\frac{\partial^2\,H}{\partial z^2} + \frac{\partial\,H}{\partial x}\,\frac{\partial\,u_x}{\partial z} + \frac{\partial\,H}{\partial y}\,\frac{\partial\,u_y}{\partial z} + \frac{\partial\,H}{\partial z}\right.$$

Let us check **Cartan identity** on the 1-form ω:

$$\mathcal{L}_v\omega = v\cdot\mathrm{d}\omega + \mathrm{d}\langle\omega, v\rangle$$

and on the 2-form A:

$$\mathcal{L}_v A = v\cdot\mathrm{d}A + \mathrm{d}(v\cdot A)$$

In [199…
```
om.lie_derivative(v) == v.contract(diff(om)) + diff(om(v))
```

Out[199…
$$\mathrm{True}$$

In [200…
```
a.lie_derivative(v) == v.contract(diff(a)) + diff(v.contract(a))
```

Out[200…
$$\mathrm{True}$$

The Lie derivative of a vector field along another one is the **commutator** of the two vectors fields:

In [201…
```
v.lie_derivative(u)(f) == u(v(f)) - v(u(f))
```

Out[201…
$$\mathrm{True}$$

Tensor fields of arbitrary rank

Up to now, we have encountered tensor fields

- of type (0,0) (i.e. scalar fields),
- of type (1,0) (i.e. vector fields),
- of type (0,1) (i.e. 1-forms),
- of type (0,2) and antisymmetric (i.e. 2-forms).

More generally, tensor fields of any type (p, q) can be introduced in SageMath. For instance a tensor field of type (1,2) on the open subset U is declared as follows:

In [202…
```
t = U.tensor_field(1, 2, name='T')
print(t)
```

```
Tensor field T of type (1,2) on the Open subset U of the 3-dimensional differentiabl
e manifold M
```

As for vectors or 1-forms, the tensor's components with respect to the domain's default frame are set by means of square brackets:

In [203…
```
t[1,2,1] = 1 + x^2
t[3,2,1] = x*y*z
```

Unset components are zero:

In [204…
```
t.display()
```

Out[204…
$$T = \left(x^2 + 1\right) \frac{\partial}{\partial x} \otimes \mathrm{d}y \otimes \mathrm{d}x + xyz \frac{\partial}{\partial z} \otimes \mathrm{d}y \otimes \mathrm{d}x$$

In [205…
```
t[:]
```

Out[205…
$$\left[\left[[0, 0, 0], \left[x^2 + 1, 0, 0\right], [0, 0, 0]\right], [[0, 0, 0], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [xyz, 0, 0], [0, 0, 0]]\right]$$

Display of the nonzero components:

In [206…
```
t.display_comp()
```

Out[206…
$$\begin{aligned} T^{x}{}_{y\,x} &= x^2 + 1 \\ T^{z}{}_{y\,x} &= xyz \end{aligned}$$

Double square brackets return the component (still w.r.t. the default frame) as a scalar field, while single square brackets return the expression of this scalar field in terms of the domain's default coordinates:

In [207…
```
print(t[[1,2,1]])
t[[1,2,1]].display()
```

```
Scalar field on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[207…
$$\begin{aligned} U &\longrightarrow \mathbb{R} \\ (x, y, z) &\longmapsto x^2 + 1 \\ (r, \theta, \phi) &\longmapsto r^2 \cos\left(\phi\right)^2 \sin\left(\theta\right)^2 + 1 \end{aligned}$$

In [208…
```
print(t[1,2,1])
t[1,2,1]
```

```
x^2 + 1
```

Out[208…
$$x^2 + 1$$

A tensor field of type (1,2) maps a 3-tuple (1-form, vector field, vector field) to a scalar field:

In [209…
```
print(t(om, u, v))
t(om, u, v).display()
```

```
Scalar field T(omega,u,v) on the Open subset U of the 3-dimensional differentiable m
anifold M
```

Out[209…
$$T(\omega,u,v): \quad U \longrightarrow \mathbb{R}$$
$$(x,y,z) \longmapsto \left(x^2+1\right)y^3u_y(x,y,z)+\left(x^2+1\right)y^2u_y(x,y,z)-\left(xy^2u_y(x,y,z)\right.$$
$$+$$
$$(r,\theta,\phi) \longmapsto \left(r^5\cos(\phi)^2\sin(\phi)\sin(\theta)^5-\left(\left(\cos(\phi)^4-\cos(\phi)^2\right)r^5\cos(\theta)-r^4\right.\right.$$
$$+\left(\left(\cos(\phi)^3-\cos(\phi)\right)r^5\cos(\theta)^2+r^4\cos(\phi)^2\cos(\theta)\sin(\phi)+r^3\right.$$
$$($$

As for vectors and differential forms, the tensor components can be taken in any frame defined on the manifold:

In [210…
```
t[Y.frame(), 1,1,1, Y]
```

Out[210…
$$r^2\cos(\phi)^4\sin(\phi)\sin(\theta)^5+\left(\cos(\phi)^4-\cos(\phi)^2\right)r^3\sin(\theta)^6-\left(\cos(\phi)^4-\cos(\phi)^2\right)r^3\sin(\theta)^4+$$

Tensor calculus

The **tensor product** $\otimes$ is denoted by `*` :

In [211…
```
print(v.tensor_type())
print(a.tensor_type())
```

```
(1, 0)
(0, 2)
```

In [212…
```
b = v*a
print(b)
b
```

```
Tensor field v⊗A of type (1,2) on the Open subset U of the 3-dimensional differentia
ble manifold M
```

Out[212…
$$v\otimes A$$

The tensor product preserves the (anti)symmetries: since A is a 2-form, it is antisymmetric with respect to its two arguments (positions 0 and 1); as a result, b is antisymmetric with respect to its last two arguments (positions 1 and 2):

In [213…
```
a.symmetries()
```

```
no symmetry; antisymmetry: (0, 1)
```

In [214…
```
b.symmetries()
```

```
no symmetry; antisymmetry: (1, 2)
```

Standard **tensor arithmetics** is implemented:

In [215…
```
s = - t + 2*f* b
print(s)
```

```
Tensor field of type (1,2) on the Open subset U of the 3-dimensional differentiable
manifold M
```

Tensor contractions are dealt with by the methods `trace()` and `contract()` : for instance, let us contract the tensor T w.r.t. its first two arguments (positions 0 and 1), i.e. let us form the tensor c of components $c_i = T^k_{\ ki}$:

In [216…
```
c = t.trace(0,1)
print(c)
```

```
1-form on the Open subset U of the 3-dimensional differentiable manifold M
```

An alternative to the writing `trace(0,1)` is to use the **index notation** to denote the contraction: the indices are given in a string inside the `[]` operator, with `'^'` in front of the contravariant indices and `'_'` in front of the covariant ones:

In [217…
```
c1 = t['^k_ki']
print(c1)
c1 == c
```

```
1-form on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[217… True

The contraction is performed on the repeated index (here `k`); the letter denoting the remaining index (here `i`) is arbitrary:

In [218…
```
t['^k_kj'] == c
```

Out[218… True

In [219…
```
t['^b_ba'] == c
```

Out[219… True

It can even be replaced by a dot:

In [220…
```
t['^k_k.'] == c
```

Out[220… True

LaTeX notations are allowed:

In [221…
```
t['^{k}_{ki}'] == c
```

Out[221… True

as well as Greek letters (only for SageMath 9.2 or higher):

In [222…
```
t['^μ_μα'] == c
```

Out[222… True

The contraction $T^i_{jk} v^k$ of the tensor fields T and v is taken as follows (2 refers to the last index position of T and 0 to the only index position of v):

In [223…
```
tv = t.contract(2, v, 0)
print(tv)
```

```
Tensor field of type (1,1) on the Open subset U of the 3-dimensional differentiable
manifold M
```

Since 2 corresponds to the last index position of T and 0 to the first index position of v, a shortcut for the above is

In [224…
```
tv1 = t.contract(v)
print(tv1)
```

```
Tensor field of type (1,1) on the Open subset U of the 3-dimensional differentiable
manifold M
```

In [225…
```
tv1 == tv
```

Out[225… True

Instead of `contract()`, the **index notation**, combined with the * operator, can be used to denote the contraction:

In [226…
```
t['^i_jk']*v['^k'] == tv
```

Out[226… True

The non-repeated indices can be replaced by dots:

In [227…
```
t['^._.k']*v['^k'] == tv
```

Out[227… True

Metric structures

A **Riemannian metric** on the manifold $\mathcal{M}$ is declared as follows:

In [228…
```
g = M.riemannian_metric('g')
print(g)
```

```
Riemannian metric g on the 3-dimensional differentiable manifold M
```

It is a symmetric tensor field of type (0,2):

In [229…
```
g.parent()
```

Out[229… $\mathcal{T}^{(0,2)}\left(\mathcal{M}\right)$

In [230…
```
print(g.parent())
```

```
Free module T^(0,2)(M) of type-(0,2) tensors fields on the 3-dimensional differentia
ble manifold M
```

In [231…
```
g.symmetries()
```

```
symmetry: (0, 1); no antisymmetry
```

The metric is initialized by its components with respect to some vector frame. For instance, using the default frame of $\mathcal{M}$:

In [232…
```
g[1,1], g[2,2], g[3,3] = 1, 1, 1
g.display()
```

Out[232… $g = \mathrm{d}x \otimes \mathrm{d}x + \mathrm{d}y \otimes \mathrm{d}y + \mathrm{d}z \otimes \mathrm{d}z$

The components w.r.t. another vector frame are obtained as for any tensor field:

In [233…
```
g.display(Y.frame(), Y)
```

Out[233… $g = \mathrm{d}r \otimes \mathrm{d}r + r^2 \mathrm{d}\theta \otimes \mathrm{d}\theta + r^2 \sin\left(\theta\right)^2 \mathrm{d}\phi \otimes \mathrm{d}\phi$

For a coordinate frame, as a shortcut, one may provide only the name of the chart:

In [234…
```
g.display(Y)
```

Out[234… $g = \mathrm{d}r \otimes \mathrm{d}r + r^2 \mathrm{d}\theta \otimes \mathrm{d}\theta + r^2 \sin\left(\theta\right)^2 \mathrm{d}\phi \otimes \mathrm{d}\phi$

Of course, the metric acts on vector pairs:

In [235…
```
print(g(u,v))
g(u,v).display()
```

```
Scalar field g(u,v) on the Open subset U of the 3-dimensional differentiable manifol
d M
```

Out[235…

$$\begin{aligned} g\,(u,v):\ U &\longrightarrow \mathbb{R} \\ (x,y,z) &\longmapsto xyzu_z\,(x,y,z) + yu_x\,(x,y,z) - xu_y\,(x,y,z) + u_x\,(x,y,z) \\ (r,\theta,\phi) &\longmapsto r^3\cos(\phi)\cos(\theta)\sin(\phi)\sin\,(\theta)^2 u_z\,(r\cos(\phi)\sin(\theta), r\sin(\phi)\sin(\theta), r\cos \\ &\quad + (r\sin(\phi)\sin(\theta)+1)u_x\,(r\cos(\phi) \end{aligned}$$

The **Levi-Civita connection** associated to the metric g:

In [236…
```
nabla = g.connection()
print(nabla)
nabla
```

```
Levi-Civita connection nabla_g associated with the Riemannian metric g on the 3-dime
nsional differentiable manifold M
```

Out[236… ∇_g

The Christoffel symbols with respect to the manifold's default coordinates:

In [237…
```
nabla.coef()[:]
```

Out[237… $[[[0,0,0],[0,0,0],[0,0,0]],[[0,0,0],[0,0,0],[0,0,0]],[[0,0,0],[0,0,0],[0,0,0]]]$

The Christoffel symbols with respect to the coordinates (r,θ,ϕ):

In [238…
```
nabla.coef(Y.frame())[:, Y]
```

Out[238… $\left[\left[[0,0,0],[0,-r,0],\left[0,0,-r\sin\,(\theta)^2\right]\right],\left[\left[0,\frac{1}{r},0\right],\left[\frac{1}{r},0,0\right],[0,0,-\cos(\theta)\sin(\theta)]\right],\left[\left[0,0,\frac{1}{r}\right.\right.\right.$

A nice view is obtained via the method `display()` (by default, only the nonzero connection coefficients are shown):

In [239…
```
nabla.display(frame=Y.frame(), chart=Y)
```

Out[239…

$$\begin{aligned} \Gamma^{r}{}_{\theta\,\theta} &= -r \\ \Gamma^{r}{}_{\phi\,\phi} &= -r\sin\,(\theta)^2 \\ \Gamma^{\theta}{}_{r\,\theta} &= \frac{1}{r} \\ \Gamma^{\theta}{}_{\theta\,r} &= \frac{1}{r} \\ \Gamma^{\theta}{}_{\phi\,\phi} &= -\cos(\theta)\sin(\theta) \\ \Gamma^{\phi}{}_{r\,\phi} &= \frac{1}{r} \\ \Gamma^{\phi}{}_{\theta\,\phi} &= \frac{\cos(\theta)}{\sin(\theta)} \\ \Gamma^{\phi}{}_{\phi\,r} &= \frac{1}{r} \\ \Gamma^{\phi}{}_{\phi\,\theta} &= \frac{\cos(\theta)}{\sin(\theta)} \end{aligned}$$

One may also use the method `christoffel_symbols_display()` of the metric, which (by default) displays only the non-redundant Christoffel symbols:

In [240…
```
g.christoffel_symbols_display(Y)
```

Out[240…
$$\begin{aligned}
\Gamma^{r}{}_{\theta\,\theta} &= -r \\
\Gamma^{r}{}_{\phi\,\phi} &= -r\sin(\theta)^2 \\
\Gamma^{\theta}{}_{r\,\theta} &= \frac{1}{r} \\
\Gamma^{\theta}{}_{\phi\,\phi} &= -\cos(\theta)\sin(\theta) \\
\Gamma^{\phi}{}_{r\,\phi} &= \frac{1}{r} \\
\Gamma^{\phi}{}_{\theta\,\phi} &= \frac{\cos(\theta)}{\sin(\theta)}
\end{aligned}$$

The connection acting as a covariant derivative:

In [241…
```
nab_v = nabla(v)
print(nab_v)
nab_v.display()
```

```
Tensor field nabla_g(v) of type (1,1) on the Open subset U of the 3-dimensional diff
erentiable manifold M
```

Out[241…
$$\nabla_g v = \frac{\partial}{\partial x}\otimes \mathrm{d}y - \frac{\partial}{\partial y}\otimes \mathrm{d}x + yz\frac{\partial}{\partial z}\otimes \mathrm{d}x + xz\frac{\partial}{\partial z}\otimes \mathrm{d}y + xy\frac{\partial}{\partial z}\otimes \mathrm{d}z$$

Being a Levi-Civita connection, ∇_g is torsion.free:

In [242…
```
print(nabla.torsion())
nabla.torsion().display()
```

```
Tensor field of type (1,2) on the 3-dimensional differentiable manifold M
```

Out[242…
$$0$$

In the present case, it is also flat:

In [243…
```
print(nabla.riemann())
nabla.riemann().display()
```

```
Tensor field Riem(g) of type (1,3) on the 3-dimensional differentiable manifold M
```

Out[243…
$$\mathrm{Riem}\,(g) = 0$$

Let us consider a non-flat metric, by changing g_{rr} to $1/(1+r^2)$:

In [244…
```
g[Y.frame(), 1,1, Y] = 1/(1+r^2)
g.display(Y.frame(), Y)
```

Out[244…
$$g = \left(\frac{1}{r^2+1}\right)\mathrm{d}r\otimes\mathrm{d}r + r^2\mathrm{d}\theta\otimes\mathrm{d}\theta + r^2\sin(\theta)^2\mathrm{d}\phi\otimes\mathrm{d}\phi$$

For convenience, we change the default chart on the domain U to Y=$(U, (r,\theta,\phi))$:

In [245…
```
U.set_default_chart(Y)
```

In this way, we do not have to specify Y when asking for coordinate expressions in terms of (r,θ,ϕ):

In [246…
```
g.display(Y.frame())
```

Out[246…
$$g = \left(\frac{1}{r^2+1}\right)\mathrm{d}r\otimes\mathrm{d}r + r^2\mathrm{d}\theta\otimes\mathrm{d}\theta + r^2\sin(\theta)^2\mathrm{d}\phi\otimes\mathrm{d}\phi$$

We recognize the metric of the hyperbolic space $\mathbb{H}^3$. Its expression in terms of the chart $(U, (x,y,z))$ is

In [247…
```
g.display(X_U.frame(), X_U)
```

Out[247…

$$g = \left(\frac{y^2+z^2+1}{x^2+y^2+z^2+1}\right) \mathrm{d}x\otimes\mathrm{d}x + \left(-\frac{xy}{x^2+y^2+z^2+1}\right) \mathrm{d}x\otimes\mathrm{d}y + \left(-\frac{xz}{x^2+y^2+z^2+1}\right)$$
$$+ \left(\frac{x^2+z^2+1}{x^2+y^2+z^2+1}\right) \mathrm{d}y\otimes\mathrm{d}y + \left(-\frac{yz}{x^2+y^2+z^2+1}\right) \mathrm{d}y\otimes\mathrm{d}z + \left(-\frac{xz}{x^2+y^2+z^2+1}\right) \mathrm{d}z$$
$$+ \left(\frac{x^2+y^2+1}{x^2+y^2+z^2+1}\right) \mathrm{d}z\otimes\mathrm{d}z$$

A matrix view of the components may be more appropriate:

In [248…

```
g[X_U.frame(), :, X_U]
```

Out[248…

$$\begin{pmatrix} \frac{y^2+z^2+1}{x^2+y^2+z^2+1} & -\frac{xy}{x^2+y^2+z^2+1} & -\frac{xz}{x^2+y^2+z^2+1} \\ -\frac{xy}{x^2+y^2+z^2+1} & \frac{x^2+z^2+1}{x^2+y^2+z^2+1} & -\frac{yz}{x^2+y^2+z^2+1} \\ -\frac{xz}{x^2+y^2+z^2+1} & -\frac{yz}{x^2+y^2+z^2+1} & \frac{x^2+y^2+1}{x^2+y^2+z^2+1} \end{pmatrix}$$

We extend these components, a priori defined only on U, to the whole manifold $\mathcal{M}$, by demanding the same coordinate expressions in the frame associated to the chart X=$(\mathcal{M}, (x, y, z))$:

In [249…

```
g.add_comp_by_continuation(X.frame(), U, X)
g.display()
```

Out[249…

$$g = \left(\frac{y^2+z^2+1}{x^2+y^2+z^2+1}\right) \mathrm{d}x\otimes\mathrm{d}x + \left(-\frac{xy}{x^2+y^2+z^2+1}\right) \mathrm{d}x\otimes\mathrm{d}y + \left(-\frac{xz}{x^2+y^2+z^2+1}\right)$$
$$+ \left(\frac{x^2+z^2+1}{x^2+y^2+z^2+1}\right) \mathrm{d}y\otimes\mathrm{d}y + \left(-\frac{yz}{x^2+y^2+z^2+1}\right) \mathrm{d}y\otimes\mathrm{d}z + \left(-\frac{xz}{x^2+y^2+z^2+1}\right) \mathrm{d}z$$
$$+ \left(\frac{x^2+y^2+1}{x^2+y^2+z^2+1}\right) \mathrm{d}z\otimes\mathrm{d}z$$

The Levi-Civita connection is automatically recomputed, after the change in g:

In [250…

```
nabla = g.connection()
```

In particular, the Christoffel symbols are different:

In [251…

```
nabla.display(only_nonredundant=True)
```

Out[251…

$$\Gamma^{x}{}_{x\,x} = -\frac{xy^2+xz^2+x}{x^2+y^2+z^2+1}$$
$$\Gamma^{x}{}_{x\,y} = \frac{x^2y}{x^2+y^2+z^2+1}$$
$$\Gamma^{x}{}_{x\,z} = \frac{x^2z}{x^2+y^2+z^2+1}$$
$$\Gamma^{x}{}_{y\,y} = -\frac{x^3+xz^2+x}{x^2+y^2+z^2+1}$$
$$\Gamma^{x}{}_{y\,z} = \frac{xyz}{x^2+y^2+z^2+1}$$
$$\Gamma^{x}{}_{z\,z} = -\frac{x^3+xy^2+x}{x^2+y^2+z^2+1}$$
$$\Gamma^{y}{}_{x\,x} = -\frac{y^3+yz^2+y}{x^2+y^2+z^2+1}$$
$$\Gamma^{y}{}_{x\,y} = \frac{xy^2}{x^2+y^2+z^2+1}$$
$$\Gamma^{y}{}_{x\,z} = \frac{xyz}{x^2+y^2+z^2+1}$$
$$\Gamma^{y}{}_{y\,y} = -\frac{yz^2+\left(x^2+1\right)y}{x^2+y^2+z^2+1}$$
$$\Gamma^{y}{}_{y\,z} = \frac{y^2z}{x^2+y^2+z^2+1}$$
$$\Gamma^{y}{}_{z\,z} = -\frac{y^3+\left(x^2+1\right)y}{x^2+y^2+z^2+1}$$
$$\Gamma^{z}{}_{x\,x} = -\frac{z^3+\left(y^2+1\right)z}{x^2+y^2+z^2+1}$$
$$\Gamma^{z}{}_{x\,y} = \frac{xyz}{x^2+y^2+z^2+1}$$
$$\Gamma^{z}{}_{x\,z} = \frac{xz^2}{x^2+y^2+z^2+1}$$
$$\Gamma^{z}{}_{y\,y} = -\frac{z^3+\left(x^2+1\right)z}{x^2+y^2+z^2+1}$$
$$\Gamma^{z}{}_{y\,z} = \frac{yz^2}{x^2+y^2+z^2+1}$$
$$\Gamma^{z}{}_{z\,z} = -\frac{\left(x^2+y^2+1\right)z}{x^2+y^2+z^2+1}$$

In [252…
```
nabla.display(frame=Y.frame(), chart=Y, only_nonredundant=True)
```

Out[252…

$$\Gamma^{r}{}_{r\,r} = -\frac{r}{r^2+1}$$
$$\Gamma^{r}{}_{\theta\,\theta} = -r^3 - r$$
$$\Gamma^{r}{}_{\phi\,\phi} = -\left(r^3+r\right)\sin\left(\theta\right)^2$$
$$\Gamma^{\theta}{}_{r\,\theta} = \frac{1}{r}$$
$$\Gamma^{\theta}{}_{\phi\,\phi} = -\cos(\theta)\sin(\theta)$$
$$\Gamma^{\phi}{}_{r\,\phi} = \frac{1}{r}$$
$$\Gamma^{\phi}{}_{\theta\,\phi} = \frac{\cos(\theta)}{\sin(\theta)}$$

The **Riemann tensor** is now

In [253…
```
Riem = nabla.riemann()
print(Riem)
Riem.display(Y.frame())
```

```
Tensor field Riem(g) of type (1,3) on the 3-dimensional differentiable manifold M
```

Out[253…
$$\mathrm{Riem}\,(g) = -r^2\,\frac{\partial}{\partial r}\otimes \mathrm{d}\theta\otimes \mathrm{d}r\otimes \mathrm{d}\theta + r^2\,\frac{\partial}{\partial r}\otimes \mathrm{d}\theta\otimes \mathrm{d}\theta\otimes \mathrm{d}r - r^2\sin(\theta)^2\,\frac{\partial}{\partial r}\otimes \mathrm{d}\phi\otimes \mathrm{d}r\otimes \mathrm{d}\phi +$$
$$\otimes \mathrm{d}r\otimes \mathrm{d}\theta + \left(-\frac{1}{r^2+1}\right)\frac{\partial}{\partial \theta}\otimes \mathrm{d}r\otimes \mathrm{d}\theta\otimes \mathrm{d}r - r^2\sin(\theta)^2\,\frac{\partial}{\partial \theta}\otimes \mathrm{d}\phi\otimes \mathrm{d}\theta\otimes \mathrm{d}\phi + r^2\sin(\theta)^2\,\frac{\partial}{\partial \theta}\otimes$$
$$+ \left(-\frac{1}{r^2+1}\right)\frac{\partial}{\partial \phi}\otimes \mathrm{d}r\otimes \mathrm{d}\phi\otimes \mathrm{d}r + r^2\,\frac{\partial}{\partial \phi}\otimes \mathrm{d}\theta\otimes \mathrm{d}\theta\otimes \mathrm{d}\phi - r^2\,\frac{\partial}{\partial \phi}\otimes \mathrm{d}\theta\otimes \mathrm{d}\phi\otimes \mathrm{d}\theta$$

Note that it can be accessed directely via the metric, without any explicit mention of the connection:

In [254…
```
g.riemann() is nabla.riemann()
```

Out[254…
$$\mathrm{True}$$

The **Ricci tensor** is

In [255…
```
Ric = g.ricci()
print(Ric)
Ric.display(Y.frame())
```

```
Field of symmetric bilinear forms Ric(g) on the 3-dimensional differentiable manifol
d M
```

Out[255…
$$\mathrm{Ric}\,(g) = \left(-\frac{2}{r^2+1}\right)\mathrm{d}r\otimes \mathrm{d}r - 2\,r^2\mathrm{d}\theta\otimes \mathrm{d}\theta - 2\,r^2\sin(\theta)^2\mathrm{d}\phi\otimes \mathrm{d}\phi$$

The **Weyl tensor** is:

In [256…
```
C = g.weyl()
print(C)
C.display()
```

```
Tensor field C(g) of type (1,3) on the 3-dimensional differentiable manifold M
```

Out[256…
$$\mathrm{C}\,(g) = 0$$

The Weyl tensor vanishes identically because the dimension of $\mathcal{M}$ is 3.

Finally, the **Ricci scalar** is

In [257…
```
R = g.ricci_scalar()
print(R)
R.display()
```

```
Scalar field r(g) on the 3-dimensional differentiable manifold M
```

Out[257…
$$\begin{aligned} \mathrm{r}\,(g):\quad \mathcal{M} &\longrightarrow \mathbb{R} \\ (x,y,z) &\longmapsto -6 \\ \text{on } U:\quad (r,\theta,\phi) &\longmapsto -6 \end{aligned}$$

We recover that $\mathbb{H}^3$ is a Riemannian manifold of constant negative curvature.

Tensor transformations induced by a metric

The most important tensor transformation induced by the metric g is the so-called **musical isomorphism**, or **index raising** and **index lowering**:

In [258…
```
print(t)
```

```
Tensor field T of type (1,2) on the Open subset U of the 3-dimensional differentiabl
e manifold M
```

In [259…
```
t.display()
```

Out[259…
$$T = \left(r^2 \cos(\phi)^2 \sin(\theta)^2 + 1\right) \frac{\partial}{\partial x} \otimes \mathrm{d}y \otimes \mathrm{d}x + r^3 \cos(\phi)\cos(\theta)\sin(\phi)\sin(\theta)^2 \frac{\partial}{\partial z} \otimes \mathrm{d}y \otimes \mathrm{d}x$$

In [260…
```
t.display(X_U.frame(), X_U)
```

Out[260…
$$T = \left(x^2 + 1\right) \frac{\partial}{\partial x} \otimes \mathrm{d}y \otimes \mathrm{d}x + xyz \frac{\partial}{\partial z} \otimes \mathrm{d}y \otimes \mathrm{d}x$$

Raising the last index (position 2) of T with g:

In [261…
```
s = t.up(g, 2)
print(s)
```

```
Tensor field of type (2,1) on the Open subset U of the 3-dimensional differentiable
manifold M
```

Note that the raised index becomes the *last* one among the contravariant indices, i.e. the tensor s returned by the method `up` is

$$s^{ab}{}_c = g^{bi} T^a{}_{ic}$$

See the up() documentation for more details.

Raising all the covariant indices of T (i.e. those at the positions 1 and 2):

In [262…
```
s = t.up(g)
print(s)
```

```
Tensor field of type (3,0) on the Open subset U of the 3-dimensional differentiable
manifold M
```

Lowering all contravariant indices of T (i.e. the index at position 0):

In [263…
```
s = t.down(g)
print(s)
```

```
Tensor field of type (0,3) on the Open subset U of the 3-dimensional differentiable
manifold M
```

Note that the lowered index becomes the *first* one among the covariant indices, i.e. the tensor s returned by the method `down` is

$$s_{abc} = g_{ai} T^i{}_{bc}$$

See the down() documentation for more details.

Hodge duality

The volume 3-form (Levi-Civita tensor) associated with the metric g is

In [264…
```
epsilon = g.volume_form()
print(epsilon)
epsilon.display()
```

```
3-form eps_g on the 3-dimensional differentiable manifold M
```

Out[264…
$$\epsilon_g = \left(\frac{1}{\sqrt{x^2 + y^2 + z^2 + 1}}\right) \mathrm{d}x \wedge \mathrm{d}y \wedge \mathrm{d}z$$

In [265…
```
epsilon.display(Y.frame())
```

Out[265…
$$\epsilon_g = \left(\frac{r^2 \sin(\theta)}{\sqrt{r^2+1}} \right) \mathrm{d}r \wedge \mathrm{d}\theta \wedge \mathrm{d}\phi$$

In [266…
```
print(f)
f.display()
```

```
Scalar field f on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[266…
$$\begin{aligned} f : \ U & \longrightarrow \mathbb{R} \\ (x,y,z) & \longmapsto z^3 + y^2 + x \\ (r,\theta,\phi) & \longmapsto r^3 \cos(\theta)^3 + r^2 \sin(\phi)^2 \sin(\theta)^2 + r\cos(\phi)\sin(\theta) \end{aligned}$$

In [267…
```
sf = f.hodge_dual(g.restrict(U))
print(sf)
sf.display()
```

```
3-form *f on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[267…
$$\star f = \left(\frac{r^3 \cos(\theta)^3 + r^2 \sin(\phi)^2 \sin(\theta)^2 + r\cos(\phi)\sin(\theta)}{\sqrt{r^2+1}} \right) \mathrm{d}x \wedge \mathrm{d}y \wedge \mathrm{d}z$$

We check the classical formula $\star f = f\, \epsilon_g$, or, more precisely, $\star f = f\, \epsilon_g|_U$ (for f is defined on U only):

In [268…
```
sf == f * epsilon.restrict(U)
```

Out[268…
$$\mathrm{True}$$

The Hodge dual of a 1-form is a 2-form:

In [269…
```
print(om)
om.display()
```

```
1-form omega on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[269…
$$\omega = r^2 \sin(\theta)^2 \mathrm{d}x + r\cos(\theta)\mathrm{d}y + (r\cos(\phi)\sin(\theta) - r\cos(\theta))\,\mathrm{d}z$$

In [270…
```
som = om.hodge_dual(g)
print(som)
som.display()
```

```
2-form *omega on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[270…
$$\star\omega = \left(\frac{r^4 \cos(\phi)\cos(\theta)\sin(\theta)^3 - r^3\cos(\theta)^3 - r\cos(\theta) + \left(r^3(\cos(\phi)+\sin(\phi))\cos(\theta)^2 + r\cos(\right.}{\sqrt{r^2+1}} \right.$$
$$+ \left(-\frac{r^4\cos(\phi)\sin(\phi)\sin(\theta)^4 - r^3\cos(\theta)^2\sin(\phi)\sin(\theta) + \left(\cos(\phi)\sin(\phi) + \sin(\phi)^2\right)r^3\cos(\theta)\,\mathrm{si}}{\sqrt{r^2+1}} \right.$$
$$+ \left(\frac{r^4\cos(\phi)^2\sin(\theta)^4 - r^3\cos(\phi)\cos(\theta)^2\sin(\theta) + \left(\left(\cos(\phi)^2 + \cos(\phi)\sin(\phi)\right)r^3\cos(\theta) + r^2\right)\,\mathrm{s}}{\sqrt{r^2+1}} \right.$$

The Hodge dual of a 2-form is a 1-form:

In [271…
```
print(a)
```

```
2-form A on the Open subset U of the 3-dimensional differentiable manifold M
```

In [272…
```
sa = a.hodge_dual(g)
print(sa)
sa.display()
```

```
1-form *A on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[272…

$$
\begin{aligned}
\star A = & \left(\frac{3\,r^5\cos(\theta)^5 + 3\,r^3\cos(\theta)^3 + \left(3\,r^6\cos(\phi)\cos(\theta)^2\sin(\phi) - 2\,r^5\cos(\phi)\cos(\theta)\sin(\phi) - 2\,r^4\cos(\theta)\sin(\phi)^3 + \left(\sin(\phi)^3 - \sin(\phi)\right)r^3\right)\sin(\theta)^3 + \left(3\,r^5\cos(\theta)^3\sin(\phi)^2 - 2\,r^4\cos(\phi)\cos(\theta)^2\sin(\phi) + r^3\cos(\phi)\cos(\theta)\sin(\phi) - 2\,r^2\cos(\right.}{\left. + \left(2\,r^4\cos(\theta)^3\sin(\phi) + r^3\cos(\phi)\cos(\theta)^2 + 2\,r^2\cos(\theta)\sin(\phi)\right)\sin(\theta)}{\sqrt{r^2+1}} \right) \\
& + \left(-\frac{r^3\cos(\theta)^3 + \left(3\,r^6\cos(\phi)^2\cos(\theta)^2 - 2\left(\cos(\phi)^2 - 1\right)r^5\cos(\theta) + 2\left(\cos(\phi)^4 - \cos(\phi)^2\right)r\right.}{\left. - \left(r^3\cos(\phi)^3 + 2\left(\cos(\phi)^3 - \cos(\phi)\right)r^4\cos(\theta)\right)\sin(\theta)^3 + \left(3\,r^6\cos(\theta)^4 + 3\,r^5\cos(\phi)\cos\right.}{\left. + r\cos(\theta) - \left(r^3(\cos(\phi)+\sin(\phi))\cos(\theta)^2 + r\cos(\phi)\right)\sin(\theta)}{\sqrt{r^2+1}} \right) \\
& + \left(\frac{2\,r^5\sin(\phi)\sin(\theta)^5 + \left(3\,r^6\cos(\theta)^3\sin(\phi) + 2\,r^4\cos(\phi)^2\cos(\theta)\sin(\phi) + 2\,r^3\sin(\phi)\right)\sin(\theta)^3}{- \left(2\,r^4\cos(\phi)\cos(\theta)^2\sin(\phi) + (\cos(\phi)\sin(\phi)+1)r^3\cos(\theta)\right)\sin(\theta)^2 - r\cos(\theta) - \left(3\,r^5\cos\right.}{\sqrt{r^2+1}} \right)
\end{aligned}
$$

Finally, the Hodge dual of a 3-form is a 0-form, i.e. a scalar field:

In [273…
```
print(da)
da.display()
```

```
3-form dA on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[273…

$$\mathrm{d}A = \left(-2\left(3\,r^3\cos(\theta)^2\sin(\phi) + r\sin(\phi)\right)\sin(\theta) - 1\right)\mathrm{d}x\wedge\mathrm{d}y\wedge\mathrm{d}z$$

In [274…
```
sda = da.hodge_dual(g)
print(sda)
sda.display()
```

```
Scalar field *dA on the Open subset U of the 3-dimensional differentiable manifold M
```

Out[274…

$$
\begin{aligned}
\star\mathrm{d}A : \ U &\longrightarrow \mathbb{R} \\
(x,y,z) &\longmapsto -\left(6\,yz^2 + 2\,y + 1\right)\sqrt{x^2+y^2+z^2+1} \\
(r,\theta,\phi) &\longmapsto -\sqrt{r^2+1}\left(2\left(3\,r^3\cos(\theta)^2\sin(\phi) + r\sin(\phi)\right)\sin(\theta) + 1\right)
\end{aligned}
$$

In dimension 3 and for a Riemannian metric, the Hodge star is idempotent:

In [275…
```
sf.hodge_dual(g) == f
```

Out[275…

True

In [276…
```
som.hodge_dual(g) == om
```

Out[276…
True

In [277…
```
sa.hodge_dual(g) == a
```

Out[277…
True

In [278…
```
sda.hodge_dual(g.restrict(U)) == da
```

Out[278…
True

Getting help

To get the list of functions (methods) that can be called on a object, type the name of the object, followed by a dot and press the TAB key, e.g. `sa.<TAB>` .

To get information on an object or a method, use the question mark:

In [279…
```
nabla?
```